Towards Simulation-Based Performance Modeling of UML specifications

Simonetta Balsamo Mattia Grosso Moreno Marzolla

Dipartimento di Informatica
Università Ca' Foscari di Venezia
via Torino 155, 30172 Mestre (VE), Italy
e-mail: {balsamo|mgrosso|marzolla}@dsi.unive.it

Technical Report CS-2003-2

Abstract

Quantitative analysis of software systems is being recognized as an important issue in the software development process. Performance analysis can help to address quantitative system analysis from the early stages of the software development life cycle, e.g, to compare design alternatives or to identify system bottlenecks. Early identification of performance problems is desirable as the cost of design change increases with the later phases in the software development cycle. In this paper we focus on simulation-oriented performance models of software systems. We introduce an approach for translating UML software specifications into simulation models. The proposed technique defines the annotation of an UML system specification by using the UML profile for Schedulability, Performance and Time specification. This allows the designer to easily add quantitative informations to the software specification in a standard and consistent way. Then we derive a process-oriented simulation model whose execution provides performance results that can be directly interpreted at the UML software specification level.

Keywords Discrete-event simulation, Software Architectures, Simulation-based performance model, Unified Modeling Language.

1 Introduction

In recent years it has been recognized that software development processes should be supported by a suitable mechanism for early assessment of software performance. Early identification of unsatisfactory performance of Software Architectures (SA) can greatly reduce the cost of design change [38, 39]. The reason is that correcting a design flaw is more expensive the later the change is applied during the software development process. This is particularly true if a waterfall-style model [40] of software development is employed, as any change during the development process requires to start back from the beginning. However, this is still a relevant issue whenever a different software development process is used.

Both quantitative and qualitative analysis can be performed at the software architectural design level. Qualitative analysis deals with functional properties of the software system such as deadlock-freedom or security. Qualitative analysis is carried out by measurement or by modeling the software system to derive quantitative figures of merit, such as, for example, the execution profile of the software, memory or network utilization. We focus on models of software systems at the SA level. These include Queuing Networks [23], Stochastic Petri Nets [28] and Stochastic Process Algebra [18].

In this paper we consider quantitative evaluation of the performances of SA at the design level based on simulation models. We consider SA specifications expressed in terms of Unified Modeling Language (UML) [29] diagrams. We propose to annotate the UML diagrams using a subset of annotations defined in the *UML Profile for Schedulability, Performance and Time Specification* [30] (hereafter referred as *UML performance profile*).

Simulation is a powerful modeling technique that allows general system models, that is simulation models can represent arbitrarily complex real-world situations, which can be too complex or even impossible to represent by analytical models. Examples of complex systems are computer systems with asynchronous communications, fork and join and simultaneous resource possessions, for which analytical models often become difficult to analyze. We propose an approach for software performance analysis based on simulation to take advantage of this modeling technique. We define a simulation model of an UML software architecture specification introducing an almost one-to-one correspondence between behaviors expressed in the UML model and the entities or processes in the simulation model.

The advantage of the proposed approach is twofold. First, this correspondence between the system and the model helps the feedback process to report performance results obtained by simulation back into the original SA. Second, this allow to define a simple translation algorithm from the SA to the simulation performance model that can be fully automated.

This paper is organized as follows. Section 2 presents some background on software performance modeling, and Section 3 we discuss some critical issues. Section 4 presents the proposed methodology introduced the annotation of the UML diagrams and the translation algorithm from SA to the simulation model. Model validation is discussed in Section 5. Conclusions and future works are presented in Section 6.

2 Software Performances Modeling

Performance analysis of software systems can be carried out by measurement or by modeling techniques. Direct measurement of an actual implementation provides an accurate assessment of the performance of a software system. This is relatively easy to do, but requires system availability or to build a system implementation before the measurement can take place. Implementing a complex system is usually a time-consuming, error-prone and expensive task. We focus on software system at the SA design level. When SA exhibits performance-related problems, it is unlikely that such problems will be fixes once the architecture has been deployed, given the high costs associated with changing the design. Hence it is useful or even necessary to evaluate early performance measures at the architectural design level, by developing and evaluating a model of SA. Performance model evaluation can is obtained by analytical, simulative or hybrid techniques.

Most of the research in the area of Software Performance Engineering (SPE) is based on developing analytical models of SA [4, 5]. However, such models can usually be built only by imposing some structural restrictions on the original SA, depending on the specific modeling formalism which has been chosen; the reason is that analytical models have often a limited expressiveness. While it is sometimes possible to simplify the model of the system in order to make it analytically tractable, there are many cases in which the significant aspects of the software system can not be effectively represented into the performance model. Examples are concurrency, simultaneous resource possession and fork and join systems.

For these reasons we propose a software performance evaluation approach based on simulation models. Simulation models can be arbitrarily detailed, in that, informally, they impose no restrictions on what they can model. The analyst has the maximum degree of freedom in selecting the aspects of the real system, that is the SA in our context, to model, and at which level of detail. This freedom comes at some cost: the drawback of simulation is that very complex models may require a lot of time and computational resources in order to be executed. The results also require sophisticated statistical techniques in order to be correctly understood [21]. While it is true that any given system can be represented at an arbitrarily high level of detail by a simulation model, the analyst often can ignore the exact details of the system being simulated. This is certainly the case with SA, since they are defined at a high level of abstraction, and many details are postponed until the implementation phase. However, while the software architect may ignore the inner details of the system being designed, he could have some more or less detailed knowledge of part of the architecture (for example if some pieces are taken from an existing, already implemented system). Whenever additional informations are available, they should be used to obtain better and more realistic performance measures. A performance model which is able to represent the SA at an arbitrary level of detail could be extremely useful in this situation.

In order to easily apply SPE techniques it is convenient to refer to a common notation for software specification. Many system notations are design-oriented and do not provide built-in facilities to express informations necessary to derive a performance model. Two main approaches have been employed to overcome these limitations. The first is introducing performance-oriented formalisms applied on informations specifically added to software models [38]. The second is the extension of modeling languages with additional notations [7, 19, 27]. While both approaches produced encouraging results, they are not widespread since they require software engineers to learn new and non-standard modeling formalisms and notations.

After its adoption in 1997 as an official Object Management Group (OMG) standard, the *Unified Modeling Language* (UML) [29] gained wide acceptance among software engineers thanks to its flexibility and ease of use. UML is a language for specifying, visualizing, constructing, and documenting software and non-software systems. It is particularly useful for, although not limited to, designing Object-Oriented systems. UML provides users with a visual modeling notation to develop and exchange models, and it defines extensibility and specialization mechanisms. It supports specifications independent of particular programming languages and development processes. Moreover, it supports higher-level development concepts such as components, collaborations, frameworks and patterns.

Given its wide acceptance, many software engineers are already acquainted with at least the basics of the UML notation. For this reason recently several SPE approaches consider UML as a starting notation on which existing and new performance evaluation techniques can be applied. This can be done because UML provides extension mechanisms to add new concepts and notations for those situations which are not covered by the core language, and to specialize the concepts, notation, and constraints for particular application domains.

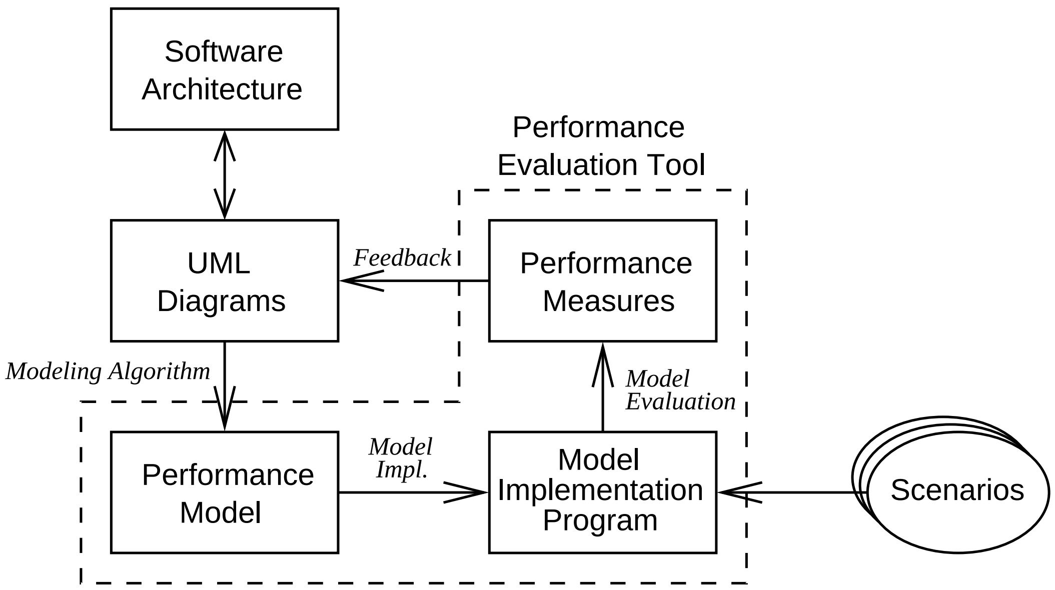

Figure 1: Phases of the modeling and performance evaluation process

We consider the the performance-oriented modeling process illustrated in Fig. 1. The starting point is a description of the SA. We consider a description as a set of UML diagrams annotated with quantitative informations in order to derive a simulation-based performance model. The model is obtained using a suitable *Modeling Algorithm* and then it is then implemented in a simulation program, which is eventually executed. Simulation results are a set of performance measures, that can be used to provide a feedback at the original SA design level. The feedback should pinpoint performance problems on the SA, and possibly provide suggestions to the software designer about how the problem can be solved. The modeling cycle shown in Fig. 1 can be iterated until a SA with satisfactory performances is developed. We focus on the first phases of the modeling process, namely the choice which UML diagrams to use and to be modeled, and how we can annotate them with quantitative performance informations useful to develop the simulation model.

Several approaches and methodologies have been recently proposed on deriving performance models from UML descriptions of SA. Most of them propose the derivation of analytical performance models. We shall now briefly discuss some references. A thorough discussion and bibliography can be found in [5, 4].

Approaches using analytic models King and Pooley [22] propose a methodology for deriving performance models based on *Generalized Stochastic Petri Nets* (GSPN) [28] from

UML collaboration and statechart diagrams. They propose to use a combination of UML diagrams – state diagrams embedded into collaboration diagrams – to better express the global state of a system. In [33], Pooley and King describe how the various kinds of UML diagrams can be used for performance evaluation purposes; to demonstrate their approach, a queuing network model is derived from a UML description of an ATM system. The approach adds textual notations to UML diagrams to include useful information for performance evaluation (e.g., time labels in sequence diagrams). Such annotations are used to produce more complete models of software systems. It should be noted that their approach can be applied regardless of the particular performance model derived.

In [8], Bernardi et al. derive a GSPN model from UML state and sequence diagrams. They define two levels of modeling: a class level and an instance level. The class level is represented by state diagrams, and is used to describe the behavior of single entities of a system. The instance level uses sequence diagrams to show patterns of interaction among objects (instances of classes). The GSPN model is then created merging together the information provided by the two kinds of diagrams.

Cortellessa and Mirandola present in [14] a methodology for translating UML sequence diagrams, use case diagrams and deployment diagrams into performance model based on *Extended Queuing Networks* (EQN) [24], using an intermediate transformation into *Execution Graphs* [38]. System performance evaluation is presented as an incremental process integrated in the software development life cycle, by using information of different kinds of UML diagrams from the early stages of the development process. The level of detail of the model is extended as the software development proceeds. This allows incremental building of a performance model of the system, which can be used to improve or modify the SA. The UML diagrams are annotated with quantitative informations, which are necessary to set the parameters of the model. Actors are annotated with the frequency they may appear in the system. Associations between actors and use cases are annotated with the probabilities that each actor executes each use case. Sequence diagrams are annotated with timing informations attached to events, and messages sent among objects are tagged with their sizes. Deployment diagrams represent various kinds of resources, and are annotated with suitable parameters such as bandwidth for network links, or speed of computational resources. Finally the three types of UML diagrams (use case, sequence and deployment diagrams) are used together to build a EQN model.

In [16], Gomaa and Menascé use UML diagrams to represent the interconnection pattern of a distributed software architecture. class diagrams are used to illustrate the static view of a system, while collaboration diagrams show the dynamic behavior. collaboration diagrams are extended with new elements showing interconnections and communication ports, and are added with performance annotations written in XML. Such annotations refer to routing probability between objects, average message processing time, average message size and average arrival rate of requests. Then they derive a performance model based on Queuing Networks.

Gu and Petriu [17] and Petriu and Shen [32] derive performance models based on *Layered Queuing Network* (LQN) models [37] from a description of SA. They use UML activity diagrams, annotated as defined in the UML Performance Profile. Diagrams and annotations are saved in XML files in the XMI format [31] and then translated in LQN models through XSL transformations (XSLT) [43].

Lindemann et al. [26] develop an algorithm for deriving performance models based on *Generalized Semi-Markov Processes* from UML state and activity Diagrams. These diagrams are annotated with exponentially distributed or deterministic delays applied to events, and timed events trigger a state transition. Annotations are based on an extension the UML Performance Profile.

Kähkipuro [20] proposes a framework on UML notation for describing performance models of component-based distributed systems. The performance model is based on Augmented Queuing Networks (AQNs).

Bernardo et alt. [9] propose an architectural description language based on the stochastic Process Algebra EMPA [10] that provides an integration of a formal specification language and performance models. They introduce the architectural description language ÆMPA, whose syntax is given both as a graphical and textual notation. The semantics of ÆMPA is given in terms of EMPA specifications.

Approaches using Simulation-based models A few works concentrate on simulation-based performance models for software systems. Arief and Speirs [1, 2, 3] develop an automatic tool for deriving simulation models from UML class and sequence diagrams. Their approach consists on transforming the UML diagrams into a simulation model described as an XML document. This model can then be translated into different kinds of simulation programs, possibly written in different languages. In this way the performance model is decoupled from its actual implementation.

De Miguel et al. [15] introduce UML extensions for the representation and automatic evaluation of temporal requirements and resource usage, particularly targeted at real-time systems. The extensions are expressed in term of stereotypes, tagged values and stereotyped constraints, and are introduced in a commercial UML CASE tool, which generates OPNET simulation models starting from annotated UML diagrams.

3 Some Critical Issues

We shall discuss now some critical issues, that are, limitations and open problems in many of the existing SPE methodologies applied to the analysis of software architectures.

Difficulty in interpreting the results Most of the SPE methodologies work by transforming the original SA into a different, performance-oriented representation. Performance-oriented models may be based on Queuing Networks, Stochastic Petri Nets and Markov Processes. Some integrated functional and performance models are based on Stochastic Process Algebras, that integrates specification and performance models into a unique framework. However, these approaches require the software designer to express the SA in a SPA specification. Informally, we observe that the software performance process produces a target performance-oriented representation usually less expressive than the original software model. Indeed, many aspects of the original design are discarded during the modeling process. Therefore it may be difficult to translate performance measures obtained by the performance model into the original SA. The use of a simulation-based performance model is expected to alleviate this problem. The transformation from software design model to the performance model should be simpler and more immediate than with analytical methods, and simulation results should be immediately interpreted at the software architectural design level.

Non-standard UML annotations Another limitation of several software performance approaches based on UML is on how the diagrams are annotated. Various approaches proposed different annotations of UML diagrams [1, 16, 20]. Standard UML extension mechanisms are *stereotypes* and *tagged values* [29]. A stereotype is a model element that defines values (based on tag definitions), constraints and optionally a new graphical representation. Each model element that is labeled by one or more particular stereotypes import these values and constraints in addition to its attributes, associations and superclasses. Tag definitions specify new kinds of properties that may be attached to model elements. The actual properties of individual model elements are specified using Tagged Values. These may either be simple datatype values or references to other model elements. The UML Performance Profile addresses this problem by defining a standard framework in which performance-oriented informations can be added to UML diagrams in a standard way.

Partial Use of the UML diagrams The current approaches for producing simulation-based performance models from UML diagrams consider only few of the diagram types defined by UML. However, software architects describe SA by using several UML diagrams at the same time. Different kinds of diagrams describe the same system from different points of view. Hence we conceive a detailed static and dynamic model of the software system by considering all these points of view. This is not an easy task, as there is not a single way of using the various UML diagrams. The user is allowed to combine them in many different ways. This allows the greatest degree of flexibility, at the price that the UML semantic is poorly and only very broadly defined, as discussed in [36].

Lack of model validation The existing SPE methodologies based on simulation performance models seldom consider the problem of model validation. This is one of the most difficult problems in simulation performance modeling and analysis [25] and is particularly difficult in SPE. The reason is that the starting point of SPE is itself a conceptual model of a software system. The goal of SPE is forecasting software system performance before the system is actually built. Thus, many of the standard verification, validation and testing techniques cannot be applied, as they rely on comparing the results obtained by the performance model with those obtained by the running system [6]. We discuss the problem of simulation model validation in software performance modeling in Section 5.

4 Proposed Methodology

In this section we propose a methodology for software performance modeling based on simulation. Referring to the modeling scheme illustrated in Fig. 1, we propose a modeling algorithm for automatic translation of annotated UML diagrams into a simulation-based performance models. Then we auotmatically derive a simulation program from the simulation model. Performance results are reported back into the original UML diagrams to show possible performance problems. The phases of the proposed approach can be included into an existing *Computer Aided Software Engineering* (CASE) tool, to better integrate software performance evaluation techniques in the software modeling design process.

First, we introduce the proposed annotations of the considered UML diagrams. Then we describe the translation algorithm from the UML specification of the SA to the process-oriented simulation model, and we discuss the implementation into a simulation program.

4.1 Annotating UML diagrams

We introduce a notation to add performance-oriented informations into UML diagrams based on a subset of the *UML Performance Profile* [30], which has recently been released as an OMG Specification. We briefly describe the UML Performance Profile in Appendix A, and a complete description can be found in [30].

We derive the simulation model from UML diagrams. Some approaches of simulation-based software performance [14, 33] identify the following diagrams as potentially useful for quantitative analysis of SA:

- Use case diagrams
- Class ciagrams
- Sequence diagrams/interaction diagrams
- Deployment diagrams
- State and activity diagrams

With respect to [33] we use a different notation to annotate UML diagrams, that is the annotation described in the UML Performance Profile. As in [14] we use use case diagrams to represent workloads applied to the system. Moreover we let the user associate a fragment of code to each workload in order to initialize parameters associated with the simulation execution of that workload.

Use case diagrams are used to describe at a very high level the interaction between the software system and one or more *actors* requiring service. The UML Performance Profile does not make any use of use case diagrams, but they can be used to represent a *Performance Context*, in the terminology introduced in the previous section. Similarly to the proposals in [33] and [14], Actors represent workloads applied to the system, and use cases represent scenarios. Thus, actors can be stereotyped with $\ll$PAclosedLoad$\gg$ or $\ll$PAopenLoad$\gg$, whose tagged values are illustrated in Table 1.

We denote with boldface strings the additions to the standard definitions of the UML Performance Profile. The *PAfreq* tag (of type *Real* represents the frequency with which the actor interacts with the system.

Moreover we propose to introduce a new stereotype, $\ll$ PAscenario $\gg$ which is not defined in the UML Performance Profile, since a scenario is implicitly defined by its first step, stereotyped as a $\ll$ PAstep $\gg$). This new stereotype can be associated to Use Cases and contains a tag *PAprob* of type *Real*, representing the probability that the given scenario occurs. Since a scenario is usually characterized by its own set of values for some user-defined variables, we propose to add another tag *PAconfiguration*, of type *String*, which

Stereotype	**Base Class**	**Tags**
≪PAclosedLoad≫	Message, Stimulus, Action, Action State, SubactivityState, ActionExecution, Operation, Method, Reception, **Actor**	PArespTime, PApriority, PApopulation, PAextDelay, **PAfreq**
≪PAopenLoad≫	Message, Stimulus, Action State, SubactivityState, Action, ActionExecution, Operation, Method, Reception, **Actor**	PArespTime, PApriority, PAoccurrence, **PAfreq**

Table 1: Proposed extension of the *workload*

contains an appropriate set of Perl statements for initializing the value of the scenario's variables.

Stereotype	**Base Class**	**Tags**
≪PAscenario≫	Actor	PAprob, PAconfiguration

Table 2: Proposed addition of the *scenario*

We assume that the normalizing probabilities constraint holds, i.e. the the probabilities associated to the scenarios which can be generated by the same actor must sum to 1.

Class diagrams We propose their use for providing informations about the computational complexity or the execution cost of object's methods. To this aim we apply the *PAdemand* tag to the relevant methods. The value of the tag represents the host execution demand of the method. The exact definition of the syntax for this type of annotations is a critical point. If the software architect already knows the implementation details for some of the methods, it should be highly desirable that such detailed information could be inserted into the annotations. The detailed information could be some sophisticated cost model or maybe an algorithmic description of an exact cost model for the method. The cost of the method could also depend on the history of invocation of other methods in the same or in other classes. As an example, the expected cost of an item insertion into a sorted, doubly linked list is a function of the size of the list. The size depends on how many calls there have been to the methods used to insert and remove items to the list object. It turns out that expressing the computational cost of the methods by means of stochastic variables is not adequate in these cases. However, given that the UML Performance Profile uses TVL as the language used to describe tag's values, it seems a sensible idea to use a more powerful subset of the Perl language (or the whole language) in place of TVL. See Section 4.3 for additional discussion on this issue.

Deployment diagrams show how objects are allocated to physical components (*nodes*) of the system. Deployment diagrams can be used to show where software components are allocated in a distributed system. Also, nodes can be connected with lines indicating the presence of a communication medium between them. Nodes and lines can be tagged with informations regarding their performances (such as CPU speed, amount of RAM, network bandwidth). The UML Performance Profile already defines the possibility of annotating nodes in the deployment diagram with the $\ll$ PAresource $\gg$ stereotype. This stereotype allows the modeler to indicate tags for the evaluated utilization of the resource, its scheduling policy, capacity, access time, response time, evaluated total waiting time, and throughput.

Sequence/collaboration diagrams. Sequence diagrams show the dynamic behavior of objects. Objects are shown as boxes with vertical dashed lines extending below them. Time flows downward along the lines; horizontal arrows represent messages sent from one object to another. Timing informations can be added by associating a $\ll$ PAstep $\gg$ stereotype to messages or to action states. If the user specifies the physical location of each object in the deployment diagram, then we can use the informations associated with the communication links between nodes in the deployment diagram to compute the delay incurred by each message, as it travels from the source to the destination object. Also, the duration of each action can be computed from its CPU demand and the performance of the computing node on which the action occurs. Collaboration diagrams can usually be substituted to sequence diagrams, as they are just a different mean of representing the same things.

State and activity diagrams are used to show the behavior of an object as a set of activities, with events causing the internal state of the object to change. A state change can be caused either by a message or by a change in some condition, or by a combination of the two. State diagrams can have a hierarchical top-down decomposition, in which states can be part of super-states. Activity diagrams are an extension of state diagrams, with explicit synchronization similar to Petri Net transitions [29]. As in collaboration/sequence diagrams, actions and states can be annotated with the $\ll$ PAstep $\gg$ stereotype, so adding various performance-related annotations to each action.

Examples of annotations are the total execution demand on its host resource, the response time, the probability to execute that particular step – if a step has multiple successor, and thus a choice has to be made –, the number of times the step has to be iterated and the total step duration. If an action refers to an execution of an object method, then the duration of the action can be computed from the method's annotations on the appropriate class diagram.

4.2 The modeling algorithm

The modeling algorithm we propose can be described by the following steps:

1. Starting from an UML representation of a software system, we annotate and use the UML diagrams as follows:

    - UML use case diagrams describe the interaction between the software system and one or more Actors requiring service. As proposed in [14, 33] we identify Actors to represent workloads applied to the system and Use Cases to represent scenarios. Actors can be stereotyped as $\ll$ PAopenLoad $\gg$ or $\ll$ PAclosedLoad $\gg$ to represent respectively open and closed population of users accessing the system. Use Cases are tagged with `PAprob` tags, whose value indicates the probability of executing that scenario.
    - Each activity of an activity diagram can be tagged with the following informations: the number of times the step has to be repeated (`PArep`); the delay between repetitions (`PAinterval`) of the same step; an additional delay for each

step representing user "think time" (`PAdelay`); the service demand of the step (`PAdemand`).

- Sequence diagrams can be annotated similarly to Activity diagrams, as they provide roughly the same information. Sequence diagrams have the advantage of explicitly showing the messages exchanged between objects. Objects communication can be expressed also in activity diagrams if we associate each activity to the object which is responsible for its execution.
- Deployment diagrams can provide additional details on the physical environment on which the SA runs. Deployment diagrams show how objects are allocated to physical components (typically processors). Also, they show how processors are interconnected together. Processors and network links are tagged with parameters describing their speed, capacity, scheduling policy and other details. By considering the allocation of active components of the SA to processors it is possible to make a more realistic estimation of action durations, as actions can be associated to the processing resource on which they execute.

2. Automatically derive the simulation model from the XMI description [31] of the UML diagrams.
3. Execute the simulation model, optionally asking the user to specify some parameters for the simulation, such as the desired confidence level for the estimation of the performance indices, the confidence interval width and the simulation length.
4. Insert back the simulation results into the UML model as tagged values associated with UML model elements.

4.3 Building the simulation

The schema of the proposed software performance approach described in Fig. 1 can be implemented as a performance evaluation tool for annotated UML diagrams. The tool can be either embedded into an existing CASE tool, or as an external product. Embedding is only feasible if the source code of the CASE tool is available, or where the CASE tool provides a public application programming interface for communicating with external applications that many commercial CASE applications offer. Embedding allows the integration of performance analysis into the CASE application. Then the user would easily get feedback during the whole design process that can be used to change the design where necessary. On the other hand, an embedded performance analysis tool is less portable than an external one. An external separate performance tool is a stand-alone application which reads a set of annotated UML diagrams from a file, simulates the SA outputting a new model annotated with the results of the simulation. Note that many CASE tools use a standard XML-based notation for representing UML models. Such notation is called XML Metamodel Interface (XMI) [31]. However, there are different implementations of XMI by the various UML tools, so there still exists a dependency of the performance tool on the CASE application used to develop the UML models.

The simulation model can be implemented as a discrete-event simulation program. The performance results which can be obtained from the analysis can be any measure (index) the user might be interested in. Examples of such measures are the resources utilization, system throughput or the total time required to execute a scenario. Note that for these indices, the UML performance profile already defines specific tagged values, whose values could be computed by the performance tool. However, given that a simulation model is very powerful, since it often can represent arbitrarily complex behaviors, it is expected that any user-defined performance measure should be computed by the simulation tool. For example, suppose that the modeler is interested in knowing the memory utilization of the system. Furthermore, suppose that the memory required by the system at a given time can be predicted very accurately, and depends on the sequence of operations performed on that component. Currently, the UML performance profile does not provide a direct definition for custom performance metrics and their computation. The Tag Value Language as defined in [30] does not provide an assignment operator, so it cannot be used for describing complex behaviors. This can be easily solved by using the full Perl language rather than TVL (which is a subset of Perl). The freely available Perl interpreter is shipped with a C library which can be linked with any program; the library provides functions for instantiating an interpreter and executing chunks of Perl code. Such chunks of code could be attached to UML elements stereotyped as $\ll$ PAstep $\gg$, defining a new tag (for example named **PAcode**) which represents a chunk of Perl code which is executed by the simulator when the associated UML element is considered.

5 Model Validation

Validation of a simulation model deals with the accuracy of the modeling process, i.e. how accurately the simulation model represents the real system. Validation is usually an iterative process, where model and system behavior are compared. Discrepancies between these behaviors are used to tune the model until its accuracy is considered acceptable [25].

In the context of SPE, the “real system” is an abstract representation of a SA. Specifically, we consider here an UML representation of a SA. Such an abstract representation cannot be directly executed, so the standard validation techniques cannot be applied within this context.

Formal validation techniques [6] can be used to validate performance models when an already implemented system is not available. The idea is to develop a formal proof that the performance model is valid, according to the scope of the simulation. To get such a formal proof of correctness both the system to simulate and the simulation model must be translated into a formal and precise mathematical notation. The model is considered validated if it is possible to formally prove that it behaves in the same way as the system and satisfies the requirements specification with sufficient accuracy. Fig. 2 sketches the process of the formal validation techniques. Note that this validation method does not require the simulation model to be actually implemented.

As observed in [6], developing formal proofs of correctness in realistic cases is not always possible under the current state of the art. If the system to be modeled is a set of UML diagrams the problem is even worse, because UML has no formal semantics. There are some research addressing this open problem, as discussed in [13, 36]). However, giving a formal description of a simulation model is difficult, and currently there is no standard notation to do so. A few works describe a semantics for a subset of a simulation language [11, 12, 41]. Thus, the simulation model could be formally described by first translating it into a simulation program for which there is a formal semantics.

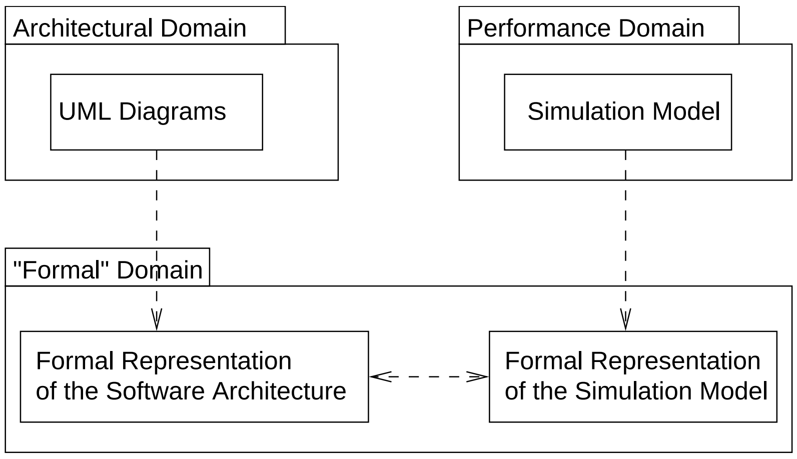
Architectural Domain
UML Diagrams
Performance Domain
Simulation Model
"Formal" Domain
Formal Representation of the Software Architecture
Formal Representation of the Simulation Model

Figure 2: Formal validation of a simulation model. The UML diagrams and the simulation model are translated into a common formal notation. These representations should be proved to be equivalent.

A simpler validation approach is to develop a set of test cases, each one being an UML representation of an already implemented SA. Then we can validate the simulation model with the implemented SA as shown in Fig. 3.

Both the approaches above have some drawbacks. Formal validation is difficult given the current state of the art, since UML descriptions of SA and simulation models have no standard, formal representation. On the other hand, validation of simulation models against testing, using standard techniques, gives no guarantee that the transformation from UML model into performance model always preserves properties of the original SA. Validation by example can be used just to show that the transformation can produce valid models, and not that the transformation must always produce valid models.

We outline another possible approach to model validation. We observe that transforming UML models into simulation models is very similar to defining a semantics of a set of annotated UML diagrams. This semantics should depend on what UML diagrams are used and how they are annotated; we assume annotations based on the UML performance profile. The transformation $\mathcal{S} : U \rightarrow M$ defines a mapping from the space of the annotated UML diagrams U into the space M of the simulation models. The mapping $\mathcal{S}$ could be defined according to the informal UML semantics defined in the standard [29], and according to the equally informal semantics of the annotations described in the UML performance profile [30]. Function $\mathcal{S}$ could be defined in an arbitrary way. However, if we treat $\mathcal{S}$ as a semantics function, we can impose a constraint on it by requiring the soundness property to hold. Suppose that we define an equivalence relation $\approx_U \subseteq U \times U$ and $\approx_M \subseteq M \times M$. Given that, we can define a soundness property for the transformation $\mathcal{S}$ as follows:

$$\text{for each } X, Y \in U : \mathcal{S}(X) \approx_M \mathcal{S}(Y) \Rightarrow X \approx_U Y \qquad (1)$$

This means that, for each pair of annotated sets of UML diagrams X and Y, if they map to equivalent performance models, then they represent similar SA.

Defining the equivalence relations $\approx_M$ and $\approx_U$ is very difficult. Two SA could be equivalent if they represent the same system at different levels of detail, or if they describe two

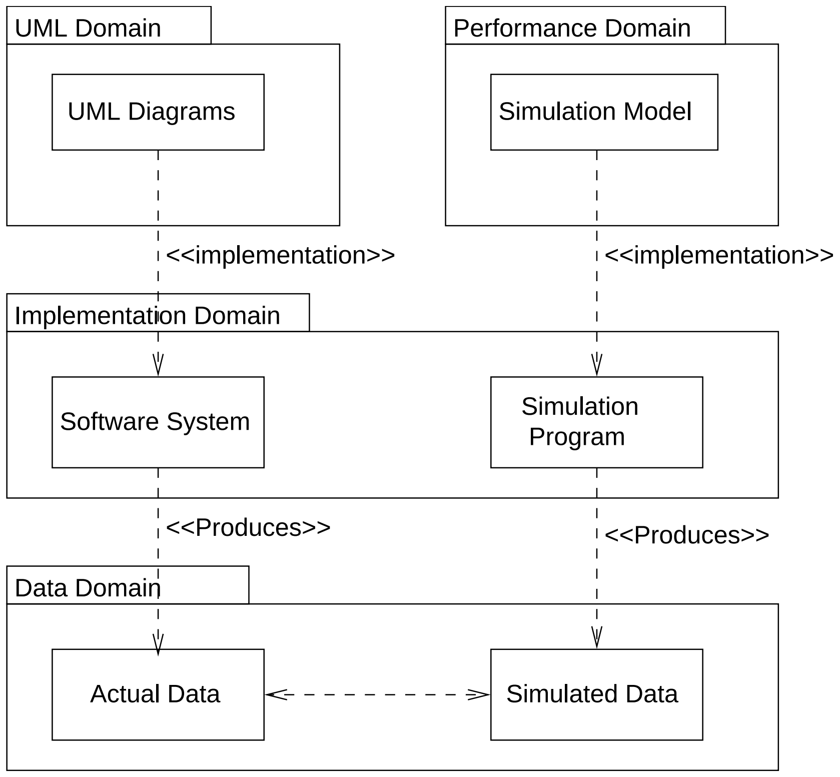

Figure 3: Implementation-based validation of simulation model.

systems providing the same functionalities, or if they are structurally equivalent, and so on. Two simulation models could be equivalent if they are structurally equivalent, or if they represent systems with the same performances under identical conditions. However, note that we could define an efficient performance model validation technique by using the definition of relations $\approx_M$ and $\approx_U$ so that the soundness property can be easily and possibly automatically verified.

6 Conclusions and Future Works

In this paper we have considered the problem of evaluating the performances of SA described in UML. We focused on the UML specification for Schedulability, Performance and Time specification, which defines a standard framework for schedulability and performance analysis. The UML profile is not specific for a performance model, so the modeler is free to use analytic, simulation or hybrid techniques. We consider simulation models as the more flexible and powerful. Moreover, we believe that an annotated set of UML diagrams maps almost directly into the appropriate simulation model in such a way that reporting the computed performance measures back into the original SA is almost trivial.

Following the approach in [33] we have considered the various classes of UML diagrams, identifying their possible use for performance analysis purposes. We suggest a simple extension to the UML performance profile, consisting in adopting use case diagrams to define scenarios to be simulated. Also, we suggest to use the full Perl language, rather than the TVL, as a means for specifying the values for the annotations of the diagrams. In this way it should be possible to express complex performance measures, other than the basic ones already included in the UML performance profile.

We have discussed the validation of the performance model as a crucial step in any software performance methodology based on simulation. The problem with the validation of simulation models derived from UML descriptions of SA is twofold. First, traditional verification and validation techniques cannot be employed, as the system to be modeled is an abstraction which is impossible to "run". Second, it is extremely difficult to express in a formal notation the simulation model, but this is necessary before trying to reason about properties and relations with the UML architecture. It should be noted that the UML performance profile defines only informally the meaning of the various tagged values to be added to the UML models.

Further research will focus on exploring this problem. Ongoing research concerns the implementation of a software performance simulation tool based on the extension of the UML performance profile discussed in this paper, and the development of an appropriate validation method.

A Overview of the UML Performance Profile

The profile has been defined using standard UML extension mechanisms, and provides the modeler with a set of packages. Each package defines the mapping between specific domain models (schedulability, time and performance characteristics) to UML stereotypes and tagged values definitions, which represent the UML viewpoint. The general structure of the profile is illustrated in Fig. 4.

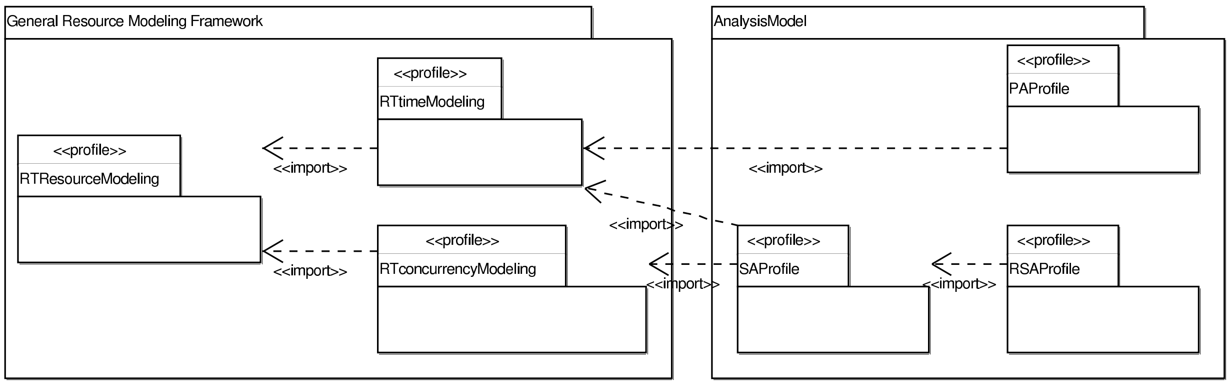

Figure 4: Structure of the UML Performance Profile.

The UML Performance Profile is partitioned into a number of sub-profiles. The core of the profile is the set of sub-profiles that represent the general resource modeling framework. These provide a common base for all the analysis sub-profiles in this specification. However, it is anticipated that future profiles dealing with other types of QoS (e.g., availability, fault tolerance, security) may need to reuse only a portion of this core. Hence, the general resource model is itself partitioned into three separate parts. The innermost part is the resource modeling sub-profile, which introduces the basic concepts of resources and QoS. These are general enough and independent of any concurrency and time-specific concepts. Since concurrency and time are at the core of the requirements behind this specification, they each have a separate sub-profile.

The modeler constructs an UML model that includes supplementary annotations required by the different model processors. The model is then passed to the model processor where it is analyzed and the results are fed back to the modeler as shown in Fig. 5. In particular, the results of the performance evaluation process are reported into the original UML model as tag values or as notes attached to the relevant UML elements.

The UML Performance Profile has several advantages. The model provided in the profile is minimal, with the expectation that further specialized profiles will be defined by users or tool vendors. However, even without such specializations the concepts are sufficient for basic performance analysis of complex systems. The profile tried to avoid any assumptions about whether the analysis method will be based on queuing theory or simulation. Hence, the profile is independent from the specific performance model used.

Fig. 6 illustrates the basic performance modeling framework, including the basic components and relationships.

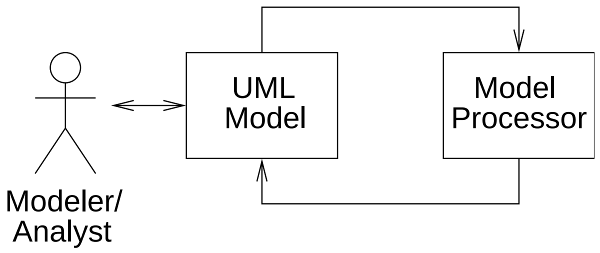

Figure 5: The modeling process.

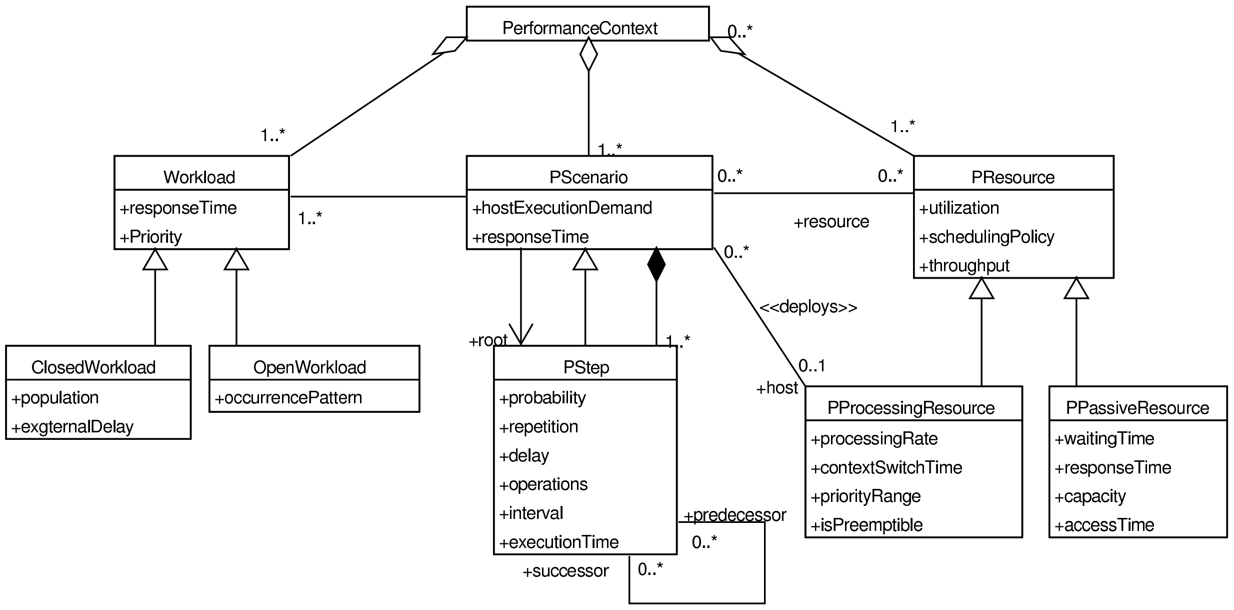

Figure 6: Performance Modeling Framework.

A *Performance Context* (PContext) represents a nonempty set of scenarios, which are used to analyze different situations. For example, Performance Contexts can be used to analyze the behavior of a system under *light* and *heavy* traffic, or with different customer arrival patterns, or assuming a different set of resources.

A *Scenario* (PScenario) is a sorted sequence of steps (PStep), each step representing the execution of some action which requires some amount of time. Multiple steps can be executed concurrently, and this is the reason why each step can have multiple predecessors (i.e., it occurs at a join point between multiple execution threads), or successors (i.e., it occurs at a fork point generating multiple execution threads). Note that a Step can itself be a Scenario (PStep inherits from PScenario), meaning that a single step can be a scenario made of finer-grained steps. Each Step has attributes indicating the probability of its execution, how many times it is to be repeated (and the delay between repetitions), the set of operations performed during the step and the total duration of the step. Each Scenario is associated with the workloads driving it, and with the set of resources required for its execution.

A *Resource* (PResource) represents an abstract view of a passive or active resource. Its attributes are the computed utilization of that resource, the policy used to control the access to the resource and the rate at which service can be provided. A *Processing Resource* (PProcessingResource) is a device (such as a processor) having processing steps allocated to it. A *Passive Resource* (PPassiveResource) may represent either a shared device or a logical entity (such as a memory buffer, or a shared queue).

A *Workload* (Workload) is used to represent either an *Open Workload* (OpenWorkload), in which the system services a stream of clients arriving with a given inter-arrival time, or a *Closed Workload* (ClosedWorkload) in which the system services a fixed population of clients which iterate between idle periods, and requests for service.

The UML Performance Profile defines two ways for mapping the above performance domain concepts into UML equivalents. The two approaches are based on Collaboration or Sequence Diagrams, and on Activity Graphs. For both approaches a set of stereotypes with associated tagged values are defined. These stereotypes are attached to the various model elements which should be considered for performance analysis purposes. The model processor is supposed to recognize these stereotypes and use the associated tagged values to obtain the model's parameters.

In the Collaboration-based approach a Performance Context is modeled as a $\ll$PAContext$\gg$ stereotype attached to a UML collaboration diagram. A Scenario is defined by applying a $\ll$PAStep$\gg$ stereotype to an action execution model element (or to a message or stimulus that directly causes that action). Workloads are defined with a $\ll$PAOpenLoad$\gg$ or $\ll$PAClosedLoad$\gg$ stereotype, attached to the first step of a scenario. Finally, resources are modeled with the two stereotypes $\ll$PAHost$\gg$ and $\ll$PAResource$\gg$. $\ll$PAHost$\gg$ is attached to the model element which actually executes the step. If multiple processing resources are present, each executing different roles, then the association of which resource executes which role is defined by a deployment diagram.

In the Activity-based approach, a Scenario is represented by an Activity Graph. This approach has the advantage connected with the hierarchical structure of the activity graphs, in which activities can be decomposed into a lower-level activity graph. The stereotypes used to annotate the UML model are the same seen above for the Collaboration-based approach, and their meaning is obviously the same. A Performance Context is modeled as an activity graph with an attached $\ll$PAContext$\gg$ stereotype. Scenarios are modeled by the set of activities of the graph, each action being stereotyped as a $\ll$PAStep$\gg$. Open and Closed workloads are represented with a $\ll$PAOpenLoad$\gg$ or $\ll$PAClosedLoad$\gg$ stereotype attached to the first step of the topmost activity graph, Modeling of active resources (processing hosts) can be done in two ways: by associating a $\ll$PAHost$\gg$ with the activity graph partition which are linked to a particular processing node, or with informations gathered from the deployment diagram.

Table 3 lists the tags associated with the various stereotypes. It also lists the elements to which each particular stereotype can be attached.

According to the UML Performance Profile, the values of the tags should be expressed with a special notation called *Tag Value Language* (TVL), which is a simple subset of the Perl language [42]. With TVL it is possible to put free variables (placeholders) inside tag values, assuming that the model preprocessor is able to instantiate these variables before running the model processor.

Stereotype	**Base Class**	**Tags**
≪PAclosedLoad≫	Message, Stimulus, Action, Action State, SubactivityState, ActionExecution, Operation, Method, Reception	PArespTime, PApriority, PA-population, PAextDelay
≪PAcontext≫	Collaboration, CollaborationInstanceSet, Activity-Graph	
≪PAHost≫	Clasifier, Node, Classifier-Role, Instance, Partition	PAutilization, PAsched-Policy, PArate, PActxSwT, PAprioRange, PApreemptable, PAthroughput
≪PAopenLoad≫	Message, Stimulus, Action State, SubactivityState, Action, ActionExecution, Operation, Method, Reception	PArespTime, PApriority, PAoccurrence
≪PAresource≫	Classifier, Node, Classifier-Role, Instance, Partition	PAutilization, PASchedPolicy, PAcapacity, PAaxTime, PArespTime
≪PAStep≫	Action, Stimulus, Action State, SubactivityState	PAdemand, PArespTime, PAprob, PArep

Table 3: Stereotypes and corresponding Tagged Values as defined in the UML Performance Profile

References

[1] L. B. Arief and N. A. Speirs, ‘Automatic generation of distributed system simulations from UML’, in *Proc. of ESM ’99, 13th European Simulation Multiconference* (Warsaw, Poland, June 1999), 85–91.

[2] L. B. Arief and N. A. Speirs, *Using SimML to bridge the transformation from UML to simulation*, in *Proc. of One Day Workshop on Software Performance and Prediction extracted from Design* (Heriot-Watt University, Edinburgh, Scotland, Nov. 1999).

[3] L. B. Arief and N. A. Speirs, ‘A UML tool for an automatic generation of simulation programs’, *in* Proceedings of WOSP 2000 [34], 71–76.

[4] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, ‘Software performance: state of the art and perspectives’, *Technical Report* **CS-2003-1,** (Dip. di Informatica, Università Ca’ Foscari di Venezia, Jan. 2003).

[5] S. Balsamo and M. Simeoni, *Deriving performance models from software architecture specifications*, in *Proc. of ESM 2001, 15th European Simulation Multiconference* (Prague, Czech Republic, June 6–9 2001).

[6] J. Banks (ed.), *Handbook of simulation* (Wiley–Interscience, 1998).

[7] A. Benzekri, A. Valderruten, O. Hjiej, and D. Gazal, ‘Deriving queueing networks performance models from annotated LOTOS specifications’, in *Computer Performance Evaluation: Modelling Techniques and Tools, 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, TOOLS ’92, Eds.* R. Pooley and J. Hillston, *Edits* **10** (Edinburgh University Press, Edinburgh, September 1992), 120–130.

[8] S. Bernardi, S. Donatelli, and J. Merseguer, *From UML sequence diagrams and statecharts to analysable Petri net models, in* Proceedings of WOSP 2002 [35].

[9] M. Bernardo, P. Ciancarini, and L. Donatiello, *Æmpa: A process algebraic description language for the performance analysis of software architectures, in* Proceedings of WOSP 2000 [34].

[10] M. Bernardo and R. Gorrieri, ‘A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time’, *Theoretical Computer Science* **202** (1998), 1–54.

[11] G. Birthwistle and C. Tofts, ‘Getting Demos models right. (I) Practice’, *Simulation Practice and Theory* **8** (2001), no. 6–7, 377–393.

[12] G. Birtwistle and C. Tofts, ‘Getting Demos models right. (II) ... and theory’, *Simulation Practice and Theory* **8** (2001), no. 6–7, 395–414.

[13] T. Clark, A. Evans, S. Kent, S. Brodsky, and S. Cook., ‘A feasibility study in rearchitecting UML as a family of languages using a precise OO meta-modeling approach - version 1.0’, *Technical Report* (Sep. 2000). http://www.cs.york.ac.uk/puml/mmf.pdf.

[14] V. Cortellessa and R. Mirandola, *PRIMA–UML: a performance validation incremental methodology on early UML diagrams, in* Proceedings of WOSP 2002 [35].

[15] M. De Miguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, and S. Piekarec, 'UML extensions for the specifications and evaluation of latency constraints in architectural models', *in* Proceedings of WOSP 2000 [34], 83–88.

[16] H. Gomaa and D. A. Menascé, 'Performance engineering of component-based distributed software systems', in *Performance Engineering – State of the Art and Current Trends*, in *Lecture Notes in Computer Science* **2047**, pp. 40–55 (Springer, 2001).

[17] G. Gu and D. C. Petriu, *XSLT transformation from UML models to LQN performance models, in* Proceedings of WOSP 2002 [35].

[18] H. Hermanns, U. Herzog, and J.-P. Katoen, 'Process algebra for performance evaluation', *Theoretical Computer Science* **274** (2002), no. 1-2, 43–87.

[19] D. Hogrefe, E. Heck, and B. Möller-Clostermann, 'Hierarchical performance evaluation based on formally specified architectures', *IEEE Transactions on Computers* **40** (1991), no. 4, 500–513.

[20] P. Kähkipuro, *UML-based performance modeling framework for component-based distributed systems*, in *Performance Engineering*, *Eds.* R. R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, *Lecture Notes in Computer Science* **2047** (Springer, 2001).

[21] K. Kant, *Introduction to computer system performance evaluation* (McGraw-Hill, Int. Editions, 1992).

[22] P. J. B. King and R. J. Pooley, 'Derivation of Petri net performance models from UML specifications of communications software', in *Computer Performance Evaluation / TOOLS*, *Eds.* B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith, *Lecture Notes in Computer Science* **1786** (Springer, 2000), 262–276.

[23] L. Kleinrock, *Queueing systems*, **1** (J. Wiley, 1975).

[24] S. S. Lavenberg, *Computer performance modeling handbook* (Academic Press, New York, USA, 1983).

[25] A. M. Law and W. D. Kelton, *Simulation modeling and analysis* (McGraw–Hill, 3rd ed., 2000).

[26] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. P. Waldhorst, *Performance analysis of time-enhanced UML diagrams based on stochastic processes, in* Proceedings of WOSP 2002 [35].

[27] N. M. Macintyre, S. R. Kershaw, and N. Xenios, 'A toolset to support the performance evaluation of systems described in SDL', in *Proc. of TOOLS '94, Vienna, Austria* (Speinger–Verlag, May 1994), 23–26.

[28] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modeling with generalized stochastic petri nets* (J. Wiley, 1995).

[29] Object Management Group (OMG), *Unified modeling language (UML), version 1.4* (Sep. 2001).

[30] Object Management Group (OMG), 'UML profile for schedulability, performance and time specification', *Final Adopted Specification* **ptc/02-03-02** (OMG, March 2002).

[31] Object Management Group (OMG), *XML Metadata Interchange (XMI) specification, version 1.2* (Jan. 2002).

[32] D. C. Petriu and H. Shen, *Applying the UML performance profile: graph grammar-based derivation of LQN models from UML specifications*, in *Computer Performance Evaluation / TOOLS*, *Eds.* T. Field, P. G. Harrison, J. Bradley, and U. Harder, in *Lecture Notes in Computer Science* **2324** (Springer, 2002).

[33] R. J. Pooley and P. J. B. King, 'The Unified Modeling Language and performance engineering', in *IEE Proceedings – Software*, **146** (February 1999), 2–10.

[34] *Proc. of the second international workshop on software and performance (WOSP 2000)* (ACM Press, September 17–20 2000).

[35] *Proc. of the third international workshop on software and performance (WOSP 2002)* (ACM Press, July 24–26 2002).

[36] G. Reggio, M. Cerioli, and E. Astesiano, 'Towards a rigorous semantics of UML supporting its multiview approach', in *FASE 2001*, *Ed.* H. Hussmann, *LNCS* **2029** (Springer–Verlag, 2001), 171–186.

[37] J. A. Rolia and K. C. Sevcik, 'The method of layers', *IEEE Transactions on Software Engineering* **21** (1995), no. 8, 689–700.

[38] C. U. Smith, *Performance engineering of software systems* (Addison-Wesley, 1990).

[39] C. U. Smith and L. Williams, *Performance solutions: A practical guide to creating responsive, scalable software* (Addison–Wesley, 2002).

[40] I. Sommerville, *Software engineering* (Addison–Wesley, 2000).

[41] C. Tofts and G. Birtwistle, 'A denotational semantics for a process-based simulation language', *ACM Transactions on Modeling and Computer Simulation (TOMACS)* **8** (1998), no. 3, 281–305.

[42] L. Wall, T. Christiansen, and J. Orwan, *Programming perl* (O'Reilly & Associates, third ed., Jul. 2000).

[43] World Wide Web Consortium (W3C), *XSL Transformations (XSLT), version 1.0* (Nov. 1999). http://www.w3.org/TR/1999/REC-xslt-19991116.