On transforming UML models into performance models

Simonetta Balsamo and Marta Simeoni
Dip. Informatica, Università *Ca' Foscari* di Venezia, Italy*
e-mail: `balsamo,simeoni`@dsi.unive.it

Abstract

The derivation of performance models from software architecture specifications has recently become a challenging research issue. The aim is to integrate quantitative analysis into the software development process, since the early stage of the software design. This would allow in fact the derivation or validation of non-functional properties of the system, and the comparison of design alternatives also from the performance viewpoint.

In this paper we present and compare some approaches which consider UML as specification language, and propose transformation techniques for obtaining performance models of the specified systems. We aim at describing how the considered approaches make use of the different UML diagrams. From the comparison of the various transformation techniques we make some observations concerning their generality, the constraints on the software system, the set of employed UML diagrams, and the additional information specified in the diagrams.

1 Introduction

In the recent years there has been a significant effort to integrate quantitative analysis into the software development process (see [Smi90]). In this framework there is a growing interest in Software Architectures (SA) [SG96] and in relating SA specification to software system performance analysis. To this aim various approaches have been proposed to derive a performance model from the SA specification. The integration of a SAs specification language with a software performance evaluation model would allow the derivation or validation of non-functional properties of the system, and the comparison of design alternatives also from the performance viewpoint.
In this context, most of the recent contributions on the derivation of performance models from SA specifications consider the Unified Modeling Language (UML) as specification language. The choice is motivated by the fact that UML is nowadays a widely accepted notation for the specification of software systems. It provides various diagrams allowing the description of the different system views, which capture static and behavioral aspects, interactions among system components, and implementation details.
In this paper we present and compare some recently proposed approaches on the transformation of UML models of SAs into performance evaluation models. We aim at pointing out how the model transformation techniques make use of the UML diagrams.
We focus on some different and relevant features of the various transformation approaches. They consider different types of UML diagrams and derive different types of performance models, such as queueing networks, stochastic Petri nets, stochastic process algebras and simulation models. Each method presents a transformation from an UML specification to the performance model. However, this is not always presented as a formal or algorithmic approach. Some methods introduce

*This work has been partially supported by MURST Project Research Funds Saladin

specific constraints on the SA specified by the UML diagrams. From the comparison of the various transformation approaches we make some observations about their generality, the constraints on the software system, the set of employed UML diagrams, and the additional information specified in the diagrams.
The paper is organized as follows: after a brief introduction on UML diagrams and performance evaluation models in section 2, Section 3 presents a survey and comparison of various transformation approaches and Section 4 provides some concluding remarks.

2 Background

In this section we introduce the UML model and performance models, which provide the basis for the description and comparison of the model transformation approaches in the next section.
UML provides several kind of diagrams, which allow different aspects and properties of a system to be expressed. We recall their main characteristics; for a complete description see, for example, [BRJ99].
The static aspects of a system can be described via *static diagrams*, which are the Class and Use Case diagrams. The dynamic system behaviour can be described by means of *behavioural diagrams*, that are State and Activity diagrams. The interactions among system components can be represented using *interaction diagrams*, that are Sequence and Collaboration diagrams. The details about the phisical implementation and the dependencies among software components can be represented via *implementation diagrams*, that are Deployment and Component diagrams.
We consider transformation approaches that map UML specifications into performance models. Each approach refers to a certain type of performance model. We briefly introduce three main classes of stochastic performance models: Queueing Network models (QNM), Stochastic Timed Petri nets (STPN) and Stochastic Process Algebras (SPA) and their extensions [Kan92, ABC986, HP98]. Each performance model of these three classes can be analyzed by analytical methods or by simulation.

- Queuing Network (QN): QN models have been widely applied as system performance models to represent and analyze resource sharing systems. A QNM is a collection of interacting *service centers*, which represent system resources, and a set of *customers* which represent users sharing the resources. Its usual informal representation is a direct graph whose nodes are service centers and arcs represent the behavior of customers' service requests.

  Extended Queuing Network (EQN) models have been introduced in order to represent several interesting features of real systems, such as synchronization and concurrency constraints, finite capacity queue and memory constraints, simultaneous resource possession.

  Another extension of QNM is the Layered Queuing Network (LQN) models that allow client-server communication pattern in concurrent and/or distributed software systems [RS95, WNPM95]. The main difference between LQN and QNM is that in LQN a server may become client (customer) to other servers while serving its own clients requests. A LQN model is represented as an acyclic graph whose nodes are software entities and hardware devices and whose arcs denote service requests.

- Stochastic Timed Petri Net (STPN): Petri nets have long been used for modeling synchronization behaviour of concurrent systems. A Petri net is defined by a set of *places*, a set of *transitions*, an *input function* relating places to transitions, an *output function* relating transition to places, and a *marking* function, associating to each place a nonnegative integer number. Petri nets are are usually represented in a graphical way: places are represented by circles, transitions by bars, input function by arcs directed from places to transitions, output function by arcs directed from transitions to bars, and marking by bullets, called *tokens*, depicted inside the corresponding places. Tokens distributed among places define the state of the net. The dynamic behaviour of a Petri net is described by the sequence of transition *firings*. Firing rules define whether a transition is *enabled* or not.

Stochastic Timed Petri Nets are Petri nets where exponentially distributed random variables, representing time duration, are associated with transitions. The firing time of a transition represents the time taken by the activity represented by the transition itself.

- Stochastic Process Algebras (SPA): Process Algebras are a widely known modeling technique for the functional analysis of concurrent systems. These are described as collections of entities, or *agents*, executing atomic *actions*, which are used to describe concurrent sequential behaviors, and synchronization or communications between them. Process algebras provide a formal model of concurrent systems, which is abstract (the internal behaviour of the system components can be disregarded) and compositional (systems can be modeled in terms of the interactions of their subsystems). SPAs are extensions of process algebras which add quantification on their models, making them suitable for performance modeling. The extension consists of associating a random variable, representing time duration, with every action.

Besides being a possible solution technique for the introduced performance models, simulation can be considered as a performance evaluation tool by itself. We can specify a simulation model by an appropriate simulation language and apply simulation to derive software system performance evaluation.

3 Model transformation approaches

In this section we present and compare some recently proposed approaches concerning the tranformation of UML models of SAs into performance models. The approaches are presented by pointing out their relevant characteristics w.r.t the model transformation point of view, such as:

- the degree of generality of the proposed transformation technique, which can be a systematic methodology, or an informal presentation, or even just a case study;
- the constraints on the SA to be modeled;
- the UML diagrams employed for the SAs specification;
- the additional information associated with the UML diagrams needed for a complete definition and analysis of the performance model. In particular, we consider how the various approaches specify this additional information and whether they introduce the use of UML extensions or simple annotations on the diagrams.
- the target performance model adopted by the approaches.

We proceed by classifying the approaches w.r.t. the degree of generality of the proposed transformation technique: each of the following subsections presents and compares the methods belonging to the same class.

3.1 General methodologies

Table 1 shows some features of three approaches which propose a general methodology for transforming UML models into performance models. They do not assume any constraint on the SAs to be modeled, and provide algorithms to generate of the performance model. These methods refer to the combination of different tools and environment for system performance evaluation. However, none of these approaches has been yet fully implemented as a complete transformation tool.

Approach	UML Diagrams	Annotations/ Extensions	Perf. Model
[AABI00]	Sequence	—	QN
[CM00]	Deployment, Sequence, Use Case	Annotations	EQN
[WS98]	Deployment, Sequence, Class	—	EQN

Table 1: General methodologies

The methods proposed in [WS98] and [CM00] are based on the Software Performance Engineering (SPE) methodology introduced by Smith in [Smi90]. It is based on two models: the *software execution model* and the *system execution model*. The former is based on execution graphs and represents the software execution behaviour, the latter is based on EQN models and represents the hardware platform. The analysis of the software model gives information concerning the resource requirements of the software system. The obtained results, together with information about the hardware devices, are the input parameters of the system execution model, which represents the model of the whole software/hardware system.
In [CM00] SAs are specified by using three different UML diagrams: Deployment, Sequence, and Use Case. The proposed methodology derives the software execution model from the Use Case and Sequence diagrams and the system execution model from the Deployment diagrams. Moreover, Deployment diagrams allow the tailoring of the software model w.r.t information concerning the overhead delay due to the communication between software components. Both Use Case and Deployment diagrams are enriched with performance annotations concerning workload distribution and parameters of devices, respectively.
The methodology in [CM00] is a more formal extension of the approach presented in [WS98]. In the latter the emphasis is in the construction and analysis of the software execution model, which is considered the target model of the specified SA and is obtained from the Sequence diagrams. The Class and Deployment diagrams contribute to complete the description of the SA, but are not involved in the transformation process.
In [AABI00], SAs are specified by means of Message Sequence Charts that are Sequence diagrams, in the UML terminology. The proposed transformation methodology is based on a systematic analysis of the Sequence diagrams which allows to single out the the real degree of parallelism among the SAs components and their dynamic dependencies. This information is then used to automatically build a QNM corresponding to the SA description. While in the two previous approaches the EQN model is derived by the SPE methodology, this apporach defines the QNM by the anlysis of a Labeled Transition System associated to the Sequence diagram.
A common aspect of [WS98] and [AABI00] approaches is that they are only interested in analyzing the SA, without specifying (or using information regarding) the underlying hardware platform.

3.2 Methodologies based on architectural patterns

SAs can be described by means of *architectural patterns*, which identify frequently used architectural solutions. Each pattern is described by its structure (what are the components) and its behaviour (how they interact). The approaches in table 2 define UML models of architectural patterns and show their corresponding performance models. Since there is a direct correspondence between a single pattern and its performance model, the target model is immediately given, without applying any particular transformation technique.

Approach	UML Diagrams	Annotations/ Extensions	Architectural Constraints	Perf. Model
[GM00]	Collaboration, Class	Annotations	client/server systems: component interconnection patterns for synchronous and asynchronous communication with a multi-threaded server	EQN
[PW99, Petriu00]	UML-Collaboration, Deployment, Use Case, Activity	Annotations	patterns for client/server, pipe and filters, broker, layers, critical section, master-slave	LQN

Table 2: Approaches based on architectural patterns

The architectural assumptions of the two approaches are quite different: [PW99, Petriu00] considers a significant set of architectural patterns (pipe and filters, client/server, broker, layers, critical section and master-slave), while [GM00] considers the case of client/server systems and model the patterns for synchronous and asynchronous communication with a multi-threaded server.
[GM00] specifies the considered patterns using Class and Collaboration diagrams, and shows for each pattern the corresponding performance model. The UML diagrams are enriched with performance annotations concerning workload intensity and service demand parameters. They are specified using an XML notation.
Similarly, the approach presented in [PW99, Petriu00] specifies the considered patterns by using UML-Collaborations (combined Class and Sequence diagrams showing explicitely the collaborating objects) and immediately derives their performance models. Moreover [PW99, Petriu00] proposes a systematic approach to build performance models of complex SAs based on combinations of the considered patterns. The approach follows the SPE methodology and generates the software and system execution models by applying graph transformation techniques. SAs are specified using UML-Collaborations, Deployment and Use Case diagrams. The Sequence diagram part of the UML-Collaboration is used to obtain the software execution model (which is represented as a UML Activity diagram); the Class part is used to obtain the system execution model (which is represented as a LQN model). Use Case diagrams provide information on the workloads, and Deployment diagrams allow for the allocation of software components to hardware sites.

3.3 Simulation methods

In table 3 we consider two approaches based on simulation models. Both the approaches are based on existing simulation packages: the information obtained from UML diagrams defines the input parameters for the generation of the corresponding simulation models.
The [DLHBP00] approach considers only real time systems, and proposes extensions of UML diagrams to express temporal requirements and resource usage. The extension are based on the use of stereotypes, tagged values and stereotyped constraints. SAs are specified using the extended UML diagrams (there are not restrictions on the type of diagrams to be used), which are then used as input for the automatic generation of the corresponding simulation models. The approach

Approach	UML Diagrams	Annotations/ Extensions	Architectural Constraints	Perf. Model
[AS00]	Sequence, Class	Annotations	—	Simulation
[DLHBP00]	all	Extensions	real time systems	Simulation

Table 3: Simulation methods

provide also a feedback mechanism: after the model has been analyzed and simulated, some results are included in the tagged values. This constitutes a relevant feature which ease the SA designer in obtaining feedback from the performance evaluation results.
The [AS00] approach is based on a UML tool specifically developed for allowing the simulation characteristics of a system to be captured. The tool allows the user to draw Class and Sequence diagrams and to specify the information needed for the automatic generation of the simulation model. The authors proposes an XML translation of the specified UML models, in order to store the information they convey in a structured way.

3.4 Case studies

Some approaches concerning the generation of performance models from UML models are simple proposals, presenting the ideas for the model transformation through an example or a case study. We consider the ones in table 4.

Approach	UML Diagrams	Annotations/ Extensions	Perf. Model
[Hoe00]	all	Extensions	QN
[KP99]	Use Case, combined State and Collaboration diagram	Annotations	STPN
[PK99]	Use Case, combined Deployment and Component diagram	Annotations	QN
[Poo99]	combined State and Collaboration diagram	Annotations	SPA

Table 4: Case studies

A common aspect of these approaches is the adding of performance information to the UML diagrams. In particular, [Hoe00] describes the various UML diagrams in terms of the information useful for performance evaluation they can express or that can be added. It proposes extensions based on the use of stereotypes and tagged values. It presents moreover some rules that can be used to propagate user requests through UML models and create input for performance calculation. These rules allow performance evaluation of UML models at different levels of abstraction.
[PK99] presents some preliminary ideas on how to obtain performance models from the UML specification of a system. Use Case diagrams are used to specify the workloads and the various classes of requests of the system being modeled. Implementation diagrams are used to define contention and to quantify the available system resources. The idea is to define a correspondence between combined Deployment and Component diagrams and QNM, by mapping Components and links to service centers.

[KP99] shows, through an example, how to generate STPN models from the UML specification of systems. It uses Use Case diagrams and combined diagrams consisting of a Collaboration diagram with State diagrams (i.e. statecharts) of all the collaborating objects embedded within them. The idea is to translate each State diagram (representing an object of the Collaboration diagram) into a Petri net: states and transitions in the State diagram are represented by places and transitions in the Petri net, respectively. The obtained Petri nets can be combined to obtain a unique STPN model that represents the whole system.
[Poo99] describes how to derive SPA models from UML specifications. Like in [KP99], the starting point is the specification of a system via a combined State and Collaboration diagram. The idea is to produce an SPA description of each object of the Collaboration diagram and to combine them into a unique model.

4 Concluding remarks

In this paper we have presented and compared some approaches on the transformation of UML models into performance models. There are some common features to most of the considered approaches. The first one concerns the use of the UML diagrams for the transformation process. Interaction diagrams turn out to be the most important ones. They model the software behaviour and information concerning the flow of control of software components. The Interaction diagrams contribute to the definition of the performance model for what concerns software components. Use Case diagrams are used to derive information to specify the workloads. Deployment diagrams and Class diagrams give information concerning the structure of hardware devices and software components, respectively. Some transformation approaches use these diagrams to obtain the structure of the corresponding performance models, such as for example the topology of the QNM. This is specially natural in those approaches that are based on architectural patterns.
Another common feature of the transformation approaches concerns the information to be added to the UML model to complete the definition and parameterization of the performance model. It turns out that the approaches generally observe that UML models do not provide all the necessary information for a complete definition of the performance model. Hence almost all the approaches propose UML extensions, or simple annotations on the diagrams.
Finally, we observe that most of the considered approaches are based on QN performance models, including their extensions like EQN and LQN. One reason is that QN models have been traditionally used for performance evaluation and prediction, and have proved to be a powerful and versatile tool. Moreover, various approaches follow the Software Performance Engineering (SPE) methodology (see [Smi90]), which is based on a software execution model and a QN model. On the other hand, some approaches lead to different types of performance models, such as stochastic Petri nets and stochastic process algebra that allow an integration of some functional and non-functional analysis. A detailed analysis and comparison of the approaches from the point of view of the properties of the performance models is out of the scope of this paper and is subject of ongoing research.

References

[ABC986] M. Ajmone, G. Balbo, G. Conte "Performance Models of Multiprocessor Performance" *The MIT Press* (1986)

[AABI00] F. Andolfi, F. Aquilani, S. Balsamo, P. Inverardi "Deriving Performance Models of Software Architectures from Message Sequence Charts" In [ACM00]

[ACM00] *ACM Proceedings of Workshop on Software and Performance, WOSP2000, Ottawa, Canada* (2000)

[AS00] L.B. Arief, N.A. Speirs "A UML Tool for an Automatic Generation of Simulation Programs" In [ACM00], pp. 71–76

[BRJ99] G. Booch, J. Rumbaugh, I. Jacobson "The Unified Modeling Language User Guide" *Addison-Wesley* (1999)

[CM00] V. Cortellessa, R. Mirandola "Deriving a Queueing Network based Performance Model from UML Diagrams" In [ACM00], pp. 58–70

[DLHBP00] M. De Miguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, S. Piekarec "UML Extensions for the Specification and Evaluation of Latency Constraints in Architectural Models" In [ACM00], pp. 83–88

[GM00] H. Gomaa, D.A. Menascé "Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures" In [ACM00], pp. 117–126

[Hoe00] F. Hoeben "Using UML Models for Performance Calculation" In [ACM00], pp. 77–82

[HP98] J. Hillston, R. Pooley "Stochastic Process Algebras and their application to Performance Modelling" Proc. Tutorial, TOOLS'98, Palma de Mallorca, Spain, Sept. 1998.

[Kan92] K. Kant "Introduction to Computer System Performance Evaluation" *McGraw-Hill* (1992)

[KP99] P. King, R. Pooley "Derivation of Petri Net Performance Models from UML Specifications of Communication Software" *Proc. of XV UK Performance Engineering Workshop* (1999)

[Petriu00] D. Petriu "Deriving Performance Models from UML Models by Graph Transformations" Tutorial material in [ACM00]

[PK99] R. Pooley, P. King "The Unified Modeling Language and Performance Engineering" *Proc. of IEE Software* (1999)

[Poo99] R. Pooley "Using UML to Derive Stochastic Process Algebra Models" *Proc. of XV UK Performance Engineering Workshop* (1999)

[PW99] D. Petriu, X. Wang "From UML descriptions of High-Level Software Architectures to LQN Performance Models" *Proc. of AGTIVE'99, Springer Verlag LNCS 1779*, pp. 47–62 (1999)

[RS95] J.A. Rolia and K.C. Sevcik "The Method of Layers" *IEEE Transaction on Software Engineering*, Vol. 21/8, pp. 682–688 (1995)

[Smi90] C.U. Smith, "Performance Engineering of Software Systems" *Addison Wesley* (1990)

[SG96] M. Shaw, D. Garlan "Software Architecture: Perspectives on an Emerging Discipline" *Prentice Hall* (1996)

[WS98] L.G. Williams, C.U. Smith "Performance Evaluation of Software Architectures" *Proc. of WOSP'98, Santa Fe, New Mexico, USA*, pp. 164–177 (1998)

[WNPM95] C. Woodside, J. Neilson, S. Petriu, S. Mjumdar "The Stochastic rendezvous network model for performance of synchronous client-server-like distributed software" *IEEE Transaction on Computer*, Vol. 44, pp. 20–34 (1995)