

## Chapter 12

# USING THE SHE METHOD FOR UML-BASED PERFORMANCE MODELING

B.D. Theelen, P.H.A. van der Putten and J.P.M. Voeten

*Information and Communication Systems Group, Faculty of Electrical Engineering, Eindhoven University of Technology. P.O. Box 513, 5600 MB Eindhoven, The Netherlands. E-mail: B.D.Theelen@tue.nl*

**Abstract:** The design of complex real-time distributed hardware/software systems commonly involves evaluating the performance of several design alternatives. Early in the design process, it is therefore desirable that design methods support constructing abstract models for the purpose of analysis. Recent extensions to the Unified Modeling Language (UML) that enable specifying schedulability, performance and time provide a means to start developing such models directly after defining the concepts and requirements of a system. However, UML hampers the evaluation of performance properties because this requires constructing executable models with a modeling language that supports application of mathematical analysis techniques. In this paper, we present how the Software/Hardware Engineering (SHE) method can be used for the performance modeling of real-time distributed hardware/software systems. Starting from a UML specification, SHE enables constructing formal executable models based on the expressive modeling language POOSL (Parallel Object-Oriented Specification Language).

**Keywords:** Formal Semantics, Performance Modeling, Parallel Object-Oriented Specification Language (POOSL), System-Level Design, and Unified Modeling Language (UML).

## 1. INTRODUCTION

Designing complex real-time distributed hardware/software systems entails considering different options for realizing the demanded functionality. In the early phases of the design process, deciding for a certain design alternative may have a deep impact on the final performance of the system. To evaluate the performance of design alternatives before actually implement-

ing the system in hardware and software, system-level design methods can be applied. System-level design methods structure the early phases of the design process by assisting the designer in developing abstract models of the system for analyzing its properties [2]. To support performance modeling, system-level design methods provide heuristics for applying certain modeling languages and performance analysis techniques. Additionally, user-friendly computer support is often supplied in the form of tools that enable efficient application of the analysis techniques by executing the constructed models.

An example of a modeling language is the Unified Modeling Language (UML) [14], which provides a set of graphical notations for specifying the functionality of a system. UML is often used to initiate and stimulate discussions on the system's concepts and for the documentation of these concepts. In principle, UML specifications are not executable, which hampers analysis of especially large real-life industrial systems. UML tools, however, often allow supplementing UML diagrams with executable code using a host language such as C++. These tools provide additional support for handling important aspects like concurrency and time because the semantics of UML does not describe how to manage them. Recent extensions to UML provide a standardized way for denoting concurrency and time aspects of a system [16, 4]. Although this real-time version of UML additionally allows capturing performance requirements and QoS characteristics, application of mathematical analysis techniques remains complicated due to the difficulty of relating such formal techniques to the informal UML diagrams.

To assist the designer in analyzing properties of a system during the early stages of the design process, the Software/Hardware Engineering (SHE) [13, 3] method was introduced. SHE provides heuristics for developing UML models that can be transformed into executable models specified in the Parallel Object-Oriented Specification Language (POOSL) [13, 1]. POOSL is an expressive modeling language with a formal (i.e., mathematically defined) semantics, intended for analyzing complex real-time distributed hardware/software systems. Based on the formal semantics, POOSL allows mathematical reasoning about the performance of a system. It has proven to be successful for modeling and analyzing real-life industrial systems. In [18] for instance, POOSL was applied for evaluating the performance of design alternatives for an industrial Internet router. This router basically concerns an input/output buffered switch system that is protected by a flow control mechanism. In this paper, we elaborate on the SHE method and discuss the UML profile it defines for developing executable performance models with POOSL. The Internet router of [18] is used as an example to illustrate several aspects of applying the SHE method.

The remainder of this paper is structured as follows. The next section presents the system-level design flow on which performance modeling with SHE is based. In section 3, the main characteristics of the UML profile for SHE are discussed, whereas the evaluation of performance properties is examined in section 4. In section 5, an overview is given of the tools provided by SHE. Section 6 briefly compares the SHE method with other methods for UML-based performance modeling. Conclusions and directions for future work are summarized in section 7.

## 2. SYSTEM-LEVEL DESIGN FLOW

Using the SHE method for performance modeling is based on the (idealized) system-level design flow presented in figure 1. System-level design starts with discussions and brainstorm-sessions on *concepts* for realizing the demanded functionality. In addition, a set of performance *requirements* that are to be satisfied by the final implementation is defined. Design experience and design decisions taken in the past may affect these concepts and requirements. Since discussions and brainstorm-sessions have a rather unstructured and undocumented character, a stage of *formulation* is involved. SHE supports the formulation stage with Object-Oriented Analysis (OOA) techniques and utilizes UML diagrams to formulate the concepts in a *concept model*. The requirements are formulated as *questions*, which actually concerns determining the design issues that need to be assessed. The result of the formulation stage is a structured (but informal) description of the concepts and requirements using schematic pictures and plain texts, composing the deliverable at milestone A (see figure 1).

After formulating the concepts and requirements, a stage of *formalization* is involved because the concept model is not executable. Formalizing the concept model concerns developing an *executable model* with the formal modeling language POOSL. The UML profile used in the formulation stage smoothens the application of POOSL for developing such models. *Validation* ensures that the executable model represents the (informal) concept model in a way that is adequate for answering the questions of interest. This is important considering the need for abstraction from implementation details that are either irrelevant for answering the questions or yet unknown in this early phase of the design. In parallel with formalizing the concept model, POOSL is used to formalize the questions into *formal properties*. These concern mathematical formulae specifying relevant performance metrics. As indicated in figure 1, the stage of formalizing the concept model and questions is completed with the deliverable at milestone B. It documents the validated executable model and formal properties.

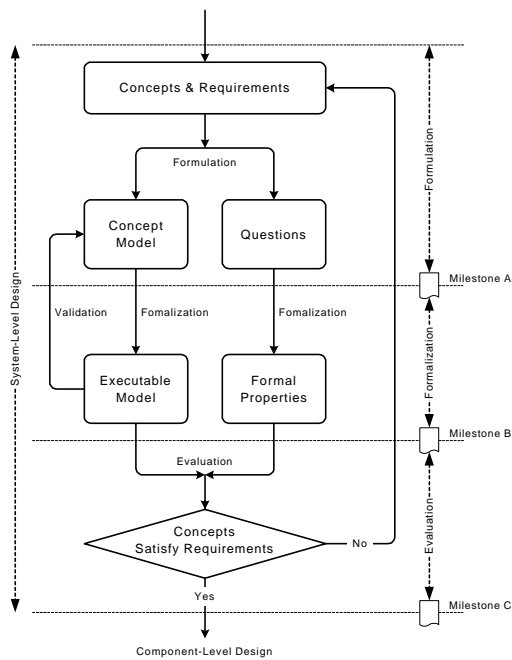


Figure 1. Formulation, Formalization and Evaluation in System-level Design.

In the final stage, the formal properties are *evaluated* against the executable model. Having a formal semantics, POOSL allows application of mathematical analysis techniques to evaluate performance metrics either by analytic computation or empiric simulation. Founded on the evaluation results, it can be concluded whether the concepts satisfy the requirements. If some requirement is not satisfied, a *design decision* can be taken based on the obtained evaluation results. Such design decisions may change the concepts for realizing the demanded functionality, the requirements or both. In these cases, the formulation, formalization and evaluation stages must be repeated\* and the deliverable at milestone C contains the evaluation results and conclusions to found the design decision. However, if all requirements are satisfied, the deliverable at milestone C documents the evaluation results and conclusions as initial requirements for the individual components of the system. As shown in figure 1, detailed component-level design can then be initiated.

\* When concurrently applying the three stages for several design alternatives, which are all expected to satisfy the requirements, an 'optimal' design alternative might be chosen based on the evaluation results. Sometimes, a parameterized model can be used to evaluate several design alternatives (See for example the Internet router of [18]). However, if none of the design alternatives satisfies the requirements, the three stages really have to be repeated to devise new concepts for realizing the demanded functionality.

### 3. UML PROFILE FOR SHE

To support developing executable models, SHE uses a UML profile in the formulation phase that expresses important characteristics of POOSL. In this section, we elaborate on the main aspects of this UML profile.

#### 3.1 Objects and Classes

Similar to the suggestion in [4], SHE distinguishes two types of resources for complex real-time distributed hardware/software systems. Active resources concern components of the system that may take the initiative to perform specific functions without involving any other resource.

Active resources can often be arranged according to a hierarchical structure. For instance, a system may incorporate several complex components, which all consist of some configuration of basic components. Passive resources, on the other hand, present information in a system that is generated, exchanged, interpreted, modified and consumed by active resources. Figure 2 depicts the three stereotype classes that SHE uses to specify different resource types.

SHE offers *data objects*, which are instances of *data classes*, to model passive resources of hardware/software systems. Their attributes are specified as *instance variables*. The sequential behavior of data objects is described with (*data*) *methods*. Data objects may receive messages, which leads to the atomic execution of equally named methods. Upon completion of executing a method, the result of a calculation or the data object itself is returned. We remark that data objects are comparable to objects in traditional object-oriented languages such as Smalltalk.

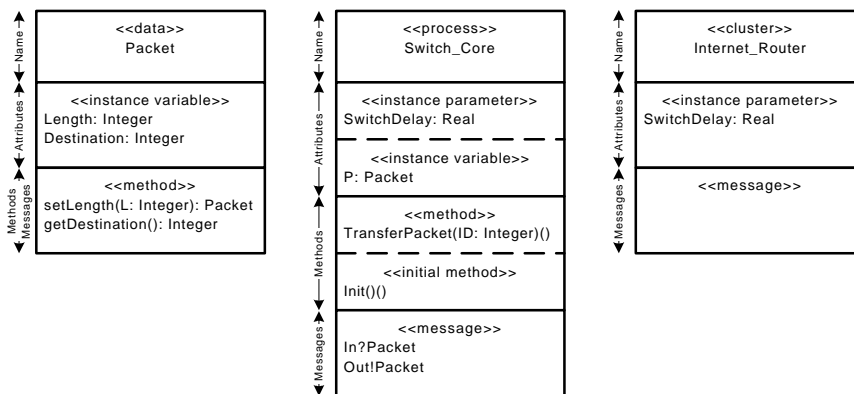


Figure 2. Data, process, and cluster classes.

To specify the real-time behavior of (non-composite) active resources, SHE provides process objects or processes. Processes are instances of process classes. Their attributes are specified either as instance variables or as instance parameters. Instance parameters allow parameterizing the behavior of a process at instantiation. We remark that such parameterization is especially useful for analyzing several design alternatives with a single model. Processes perform their behavior independently from each other, in an autonomous (asynchronous) concurrent way. The behavior of processes is described with sequentially or concurrently executed (process) methods. Methods may be called with parameters and may return results through return variables. Methods without return results can be called tail-recursively, which offers an intuitive way to model infinitely repeating behavior. The start behavior of a process is defined with a unique initial method.

Processes and clusters (see below) are statically interconnected by *channels*. Channels model any possibility to exchange information between components as for example specified in collaboration or sequence diagrams. Channels interconnect processes and clusters independent from communication protocols and may reflect any topology [13]. Processes can communicate (copies of) their encapsulated data objects by (synchronously) passing messages over channels through *ports*. Next to the use of standard UML message types, SHE defines some additional message stereotypes. As POOSL only incorporates a statement for synchronous message passing, a set of modeling patterns is provided for formalizing the other types of messages. Notice that messages exchanged between processes are not directly related to their methods (as is the case for data objects). Therefore, a separate compartment in process classes specifies the messages it can send and receive, see figure 2.

To model composite active resources, SHE provides *clusters*. Clusters are instances of *cluster classes* and group a set of processes and clusters (of other cluster classes). The need for using a cluster emerges from aggregation relationships between classes that represent active resources. Parameterized behavior of processes and clusters incorporated in a cluster can be initialized via the *instantiation parameters* of that cluster. The behavior of clusters is defined by the parallel composition of the incorporated processes (possibly interconnected by channels) and therefore does not extend the behavior of these processes. Consequently, cluster classes do not include a method compartment. Messages of incorporated processes (or clusters) that may pass the cluster's boundary through ports are specified in the message compartment.

Focusing on class diagrams, the identification of specialization/generalization relationships commonly results in using the inheritance and method overriding capabilities of POOSL. We remark that specialization/generalization relationships are only possible between passive resources

on one hand and between active resources on the other hand. Any relationship in class diagrams other than specialization/generalization and aggregation has to be expressed by the behavior of the involved objects. Another important aspect of using SHE is the need for modeling the environment as specified with actors in use case diagrams. Being active resources, actors are modeled with processes and clusters. The classes representing actors therefore emerge in class diagrams as well.

### 3.2 Architectural Structure

Processes, channels and clusters provide the means for specifying the hierarchical structure, topology or implementation boundaries of real-time distributed hardware/software systems [13]. These aspects are commonly reflected in deployment, component or collaboration diagrams. Although supporting these types of specifications, using SHE involves deriving so-called *instance structure diagrams*, which formally define the architecture of models. Instance structure diagrams graphically represent the static structure of processes and clusters in a way that is independent from scenarios [13]. Figure 3 depicts an instance structure diagram for the Internet router example of [18], where similar physical resources were modeled using a single process. Channels are introduced when processes or clusters communicate in some scenario that is reflected in for example collaboration or sequence diagrams. Such channels are labeled with a name and are connected to the ports of the involved processes and clusters, which are indicated with a small rectangle at their edges. In instance structure diagrams, no information on the order of messages nor on the type of message is represented. We remark that one channel may be used to indicate the possibility of communicating different message types between the same components.

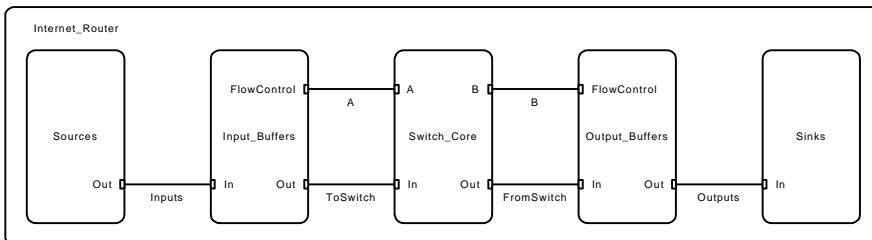


Figure 3. Instance structure diagram of the cluster class named Internet\_Router.

### 3.3 Behavior of Processes

To formalize the behavior of active resources, which is often specified using state or activity diagrams, POOSL offers the statements indicated in table 1.

Method abstraction and the statements for sequential and parallel composition provide ample means for formalizing the behavioral hierarchy that is sometimes expressed in state and activity diagrams. The statement for parallel composition denotes the purely interleaved (i.e., non-communicating) execution of statements  $S_1$  through  $S_n$ . With this statement, POOSL offers (asynchronous) concurrency *within* processes next to the usual concurrency *among* processes. The concurrent activities specified with  $S_1$  through  $S_n$  may share the data objects assigned to the instance parameters and instance variables of the involved process. Operations on data objects are atomic (i.e., indivisible), which solves mutual exclusion problems in a natural way.

Table 1. Statements for specifying the behavior of processes.

Statement $S$	Description
$m(E_1, \dots, E_i)(v_1, \dots, v_j)$	Method Abstraction
<b>par</b> $S_1$ <b>and</b> $S_2$ <b>and</b> ... <b>and</b> $S_n$ <b>rap</b>	Parallel Composition
$S_1; \dots; S_n$	Sequential Composition
$E_p!m(E_1, \dots, E_i)\{E_{at}\}$	Message Send
$E_p?m(v_1, \dots, v_i E_{rc})\{E_{at}\}$	(Conditional) Message Receive
<b>sel</b> $S_1$ <b>or</b> $S_2$ <b>or</b> ... <b>or</b> $S_n$ <b>les</b>	Non-deterministic Selection
$[E]S$	Guarded Execution
<b>abort</b> $S_1$ <b>with</b> $S_2$	Abort
<b>Interrupt</b> $S_1$ <b>with</b> $S_2$	Interrupt
<b>if</b> $E$ <b>then</b> $S_1$ <b>else</b> $S_2$ <b>fi</b>	Choice
<b>while</b> $E$ <b>do</b> $S$ <b>od</b>	Loop
$E$	Data Expression
<b>skip</b>	Empty Behavior
<b>delay</b> $E$	Time Synchronization

The send and receive statements formalize state transitions or events related to the communication of information between components. As an extension to standard UML, message passing in SHE can be conditional. Communication only occurs when the message names match, the number of message parameters is equal and the (optional) *reception condition*  $E_{rc}$  (possibly depending on the received data) evaluates to **true**. Both the send and receive statement may be followed by an atomically executed expression  $E_{at}$  on data objects.

Utilizing the statement for non-deterministic selection results in executing one of its constituent statements, choosing for the (non-blocking) alternative that performs an execution step first. It is especially useful for formalizing transitions to alternative states in the case where the transitions that can

be taken are triggered by different events. Guarded execution allows blocking the execution of a statement  $S$  as long as the guarding expression  $E$  evaluates to **false** and provides the means for formalizing guard conditions in state diagrams.

Abnormal termination of behavior as specified in state diagrams can be formalized with the abort statement. Next to exceptions resulting in terminating behavior, SHE allows to specify the interruption of behavior. Such exceptions can be formalized by using the interrupt statement. The statements for formalizing a choice or loop have their usual interpretation. We remark that in case the involved methods do not involve return variables, iterating transitions are commonly formalized based on tail-recursive method calls. Data expressions enable formalizing complex functional operations or calculations on data objects. The **skip** statement provides a means for formalizing a transition without any effect.

The statement **delay**  $E$  models postponing activity for  $E$  units of time. It offers the only way to express quantitative timing behavior. This is sufficient because it can be combined with the interrupt and abort statements to formalize more intricate timing behavior like time-outs or watchdogs. We remark that the time domain of SHE can be dense (in which case  $E$  can be real-valued).

SHE provides only a limited number of heuristics for deriving appropriate POOSL code from state or activity diagrams. Using separate methods for representing individual (composite) states or activities is one of the possibilities to obtain such POOSL code. Based on the mathematical approval of combining statements without limitations, POOSL is expressive enough for formalizing behavior according to the designer's preferences.

#### 4. PERFORMANCE EVALUATION

To enable evaluation of performance properties with POOSL, probabilistic behavior and performance related information of the system has to be formalized. For specifying probabilistic behavior like state transition probabilities in state diagrams or probabilistic time delays, SHE provides library (data) classes, which enable instantiating random variables of various probability distributions. Performance related information may explicitly be specified in UML diagrams as QoS characteristics like proposed in [4] or may originate from the involved performance questions. To formalize performance related information, the SHE method involves extending a POOSL model with additional variables and behavior. Such variables enable collecting rewards for the performance metrics that need to be evaluated. To analyze for example the average occupancy of a buffer, we can introduce a vari-

able `Occupancy` that represents the current occupancy of the buffer. Each time an item is put into or removed from the buffer, added behavior updates `Occupancy` with a new value. The long-run average occupancy can now be evaluated based on the consecutive values of the variable `Occupancy` using appropriate mathematical analysis techniques.

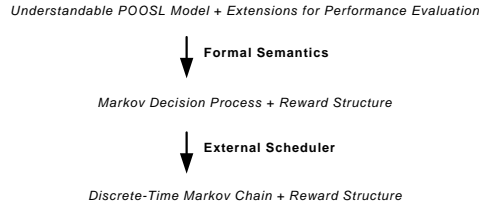


Figure 4. Mathematical framework for performance evaluation with POOSL.

Figure 4 gives a brief overview of the mathematical framework for performance evaluation with POOSL. Based on the formal semantics, each POOSL model defines a unique timed probabilistic labeled transition system [1]. The transition system concerns a Markov decision process, which can be transformed into a *discrete-time Markov chain*. As discussed in [21], the transformation involves resolving any non-determinism in the model by an external scheduler (i.e., execution engine) to ensure that each performance metric gives rise to a single performance figure. The schedulers of the tools presented in section 5 resolve non-determinism in a (uniform) probabilistic manner.

Adding variables to represent rewards for evaluating performance properties results in defining a *reward structure*. This reward structure enables evaluating performance metrics by analyzing the long-run average behavior of the underlying Markov chain. Standard techniques like equilibrium analysis [20] provide the means for *computing* performance results analytically. However, real-life systems commonly entail many concurrent activities resulting in huge Markov chains that are mathematically intractable. Performance evaluation is therefore often based on simulations, which *estimate* the actual performance results. A problem is however that it is unclear how long a simulation should run before the performance results are sufficiently accurate. Estimated performance results therefore only have a proper meaning if their accuracy is known, see for example [10]. Based on the formal semantics, POOSL enables integral accuracy analysis using *confidence intervals*. In [19], we introduced library (data) classes to support the accuracy analysis of different types of long-run averages and variances. These classes additionally enable automatic termination of a simulation when the results become sufficiently accurate.

## 5. TOOL SUPPORT

SHE supports an efficient application of POOSL for performance modeling with two different tools. In the formalization stage, the SHESim tool [3] enables incremental construction of POOSL models, that can be validated by interactive simulations. The recently developed Rotalumis tool [1] is intended for high-speed execution of (very large\*) POOSL models in the evaluation stage. We briefly discuss these tools below.

In the formulation stage, SHE currently relies on commercial tools for drawing (stereotyped) UML diagrams. Constructing a POOSL model from these UML diagrams is performed by hand. Future research includes an investigation on the possibility of deriving POOSL models more automatically from a consistent set of (stereotyped) UML diagrams specified with a tool that integrates computer support for all three stages of system-level design.

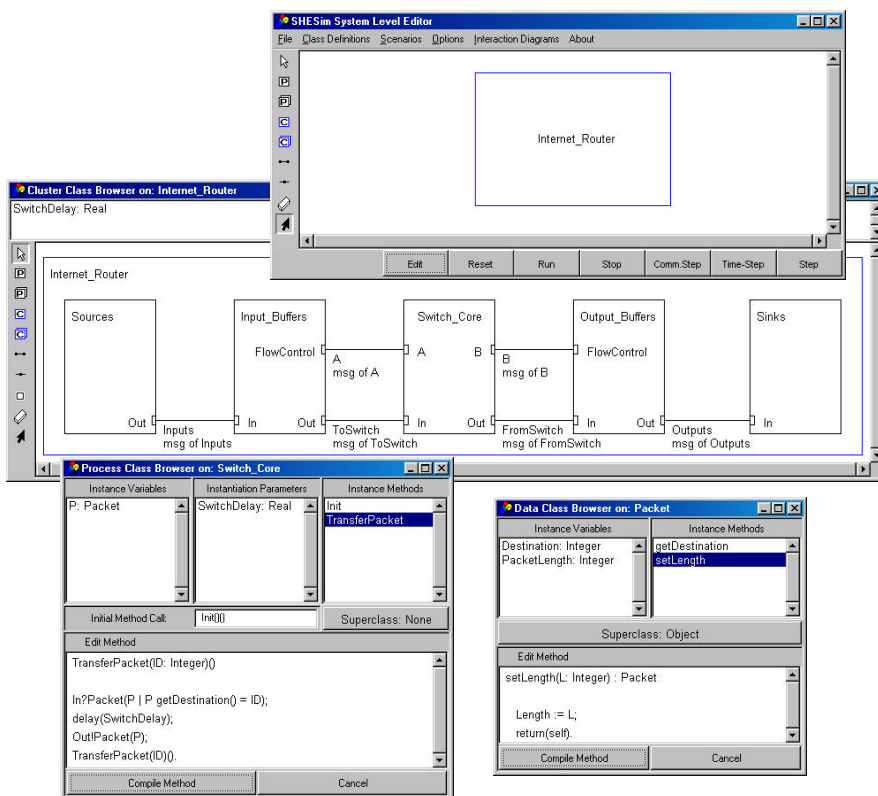


Figure 5. Editing data, process and cluster classes with SHESim.

\* We have been able to analyze models with over a million concurrent activities.

## 5.1 Formalization with SHESim

The SHESim tool is used for specifying data and process classes with appropriate POOSL code. Cluster classes are defined by drawing instance structure diagrams. A snapshot of the SHESim tool regarding the editing of data, process and cluster classes is given in figure 5. Notice the resemblance between the instance structure diagram of figure 3 and the cluster class browser window.

Next to the construction of POOSL models, SHESim supports their execution. Execution of POOSL models boils down to traversing one of the paths through the underlying Markov chain. With the various buttons at the bottom of the system-level editor window in figure 5, the model can be executed in several modes. As discussed in [3], SHESim allows executing POOSL models on a per scenario basis, which provides a means to validate whether for example different use cases are adequately formalized. When executing a POOSL model with the SHESim tool, messages passed between processes and clusters are shown on the involved channels. This is illustrated in the lower window of figure 6 and allows validation of the model by comparing the occurring sequences of messages with the specification in collaboration diagrams. In addition, *interaction diagrams* can be generated, which provide ample means to validate the model against sequence diagrams. Figure 6 shows an example interaction diagram.

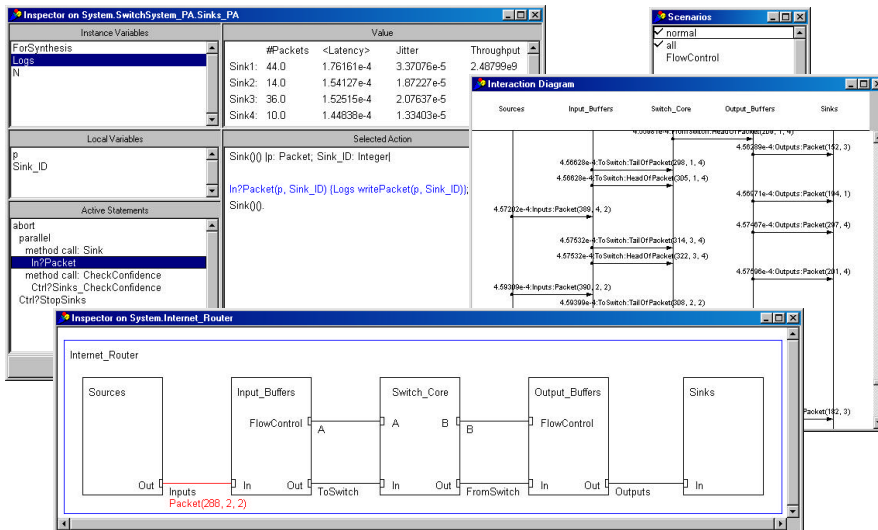


Figure 6. Simulation with SHESim.

As indicated in [3], it is possible to open inspectors on each part of an executing model (data objects, processes, clusters and channels). Such inspectors show for example the data object assigned to a variable. As such, they can reflect the (intermediate) results for a performance metric based on using the POOSL library classes of [19], see figure 6 for an example. It is also possible to inspect the current state of a process. As shown in figure 6, the statements that can potentially be executed are highlighted. Inspection of processes therefore allows validating whether state and activity diagrams are formalized in an adequate way.

## 5.2 Evaluation with Rotalumis

Rotalumis is especially developed for high-speed execution of very large POOSL models in the evaluation stage. Whereas SHESim executes a POOSL model in an interpretive way, Rotalumis compiles the POOSL model into intermediate byte code that is executed on a virtual machine implemented in C++. Compared to SHESim, this improves the execution speed by a factor of about 100. Figure 7 depicts a snapshot of the screen output that Rotalumis produces during execution of a POOSL model. It displays only a restricted amount of information about the progress of a simulation. For instance, it shows the run time (RT) of the execution and the simulated or model time (ST). Simulation results for evaluated performance metrics are logged to files. For this purpose, the POOSL library classes for accuracy analysis of performance results introduced in [19] implement logging facilities.

```
*****
****  Rotalumis high-speed execution engine
***   Programmed by L.J. van Bokhoven. (c) 2001
**    For more information visit: http://www.ics.ele.tue.nl/~lvbokhov
*

Garbage collector: incremental Baker's treadmill
Simulation time representation: 64-bit floating point.

Loading & compiling POOSL specification...

Running simulation...

RT: 0.22 Steps: 32769 ST: 0.0011098 GCCC: 0 VMC: 1.12746e+006
RT: 0.45 Steps: 65538 ST: 0.00194425 GCCC: 0 VMC: 2.21533e+006
RT: 0.66 Steps: 98307 ST: 0.00283991 GCCC: 0 VMC: 3.33673e+006
RT: 0.98 Steps: 131076 ST: 0.00377648 GCCC: 0 VMC: 4.4428e+006
RT: 1.37 Steps: 163845 ST: 0.00457058 GCCC: 0 VMC: 5.52231e+006
```

Figure 7. High-speed execution with Rotalumis.

## 6. RELATED RESEARCH

In recent years, much research is performed on methods for UML-based performance modeling of real-time distributed hardware/software systems. An important issue is how accompanying tools handle probabilism, concurrency and time. In POOSL, these aspects are inherent to a model and the formal semantics defines how the SHESim and Rotalumis tools manage them [1]. Tools of other UML-based methods often do not separate simulated or model time from run time. For example, deriving SDL [22] specifications from UML diagrams or approaches based on ROOM [17] relying on tools like ObjectTime or RoseRT suffer from the problem that hard real-time behavior cannot be modeled adequately [7]. This is because time is not an inherent aspect of SDL and ROOM models. As a consequence, these approaches are less suited for adequately analyzing for example dedicated hardware systems. Probabilistic behavior can be specified in SDL and ROOM using a proper random number generator. However, ROOM lacks a mathematical basis to enable integral application of performance analysis techniques as is possible when using POOSL, which does incorporate probabilistic features in its formal semantics [1].

The need for a rigorous mathematical basis to support performance modeling was also recognized in [12]. An important conclusion of [12] is the necessity to express probabilistic information in UML diagrams. Both [12] and [15] suggest to develop an integrated framework for explicitly representing performance related information in UML diagrams (as is now standardized in [4]) and the transformation of these UML diagrams into a mathematically analyzable performance model. An example of such an integrated framework is discussed in [5], where the transformation to a queuing network is based on an intermediate textual representation of the performance related information specified in UML diagrams. The SHE method is comparable to [5] in that it also uses an intermediate modeling language to formalize a UML specification. Opposed to POOSL, the language used in [5] lacks however proper support for specifying concurrent activities.

In [12], several options are assessed for automatically deriving a queuing network, Petri net or Markov chain from UML diagrams. References [9, 6] and [11] respectively discuss these options in more detail. In [9], the structure of queuing networks is obtained from use case and deployment diagrams, whereas the behavior of servers in these queuing networks is determined by combining collaboration and state diagrams. However, since the use of exponential distributions in queuing networks only allows for very abstract performance modeling, their practical usability for performance modeling of real-life industrial systems might be limited. In [6], the difficulty of deriving a single Petri net when combining the individual Petri nets

of communicating active objects was discussed. The approach in [11] for generating a Markov chain suffered from a similar problem, which was solved using a process algebra. The formal semantics of POOSL combines ideas of traditional imperative object-oriented programming languages with a probabilistic real-time version of the process algebra CCS [8]. Instead of directly deriving a Markov chain as proposed in [11], SHE involves using an understandable modeling language to implicitly define a Markov chain. Only in case of calculating performance results analytically it is really necessary to generate this Markov chain explicitly from the POOSL model. Evaluating performance metrics by simulation merely requires to traverse one of the paths through the Markov chain [19]. This approach is especially efficient for large real-life industrial systems.

## 7. CONCLUSIONS AND FUTURE WORK

This paper discussed UML-based performance modeling according to the SHE method for system-level design of complex real-time distributed hardware/software systems. SHE structures performance modeling in the stages of formulation, formalization and evaluation. It uses UML diagrams in the formulation stage for documenting the system's concepts and accompanying requirements. These diagrams comply with an intuitively applicable UML profile, which distinguishes data, process and cluster classes to specify the passive and active resources of hardware/software systems. Basic active resources, of which the behavior is commonly expressed in collaboration, sequence, state or activity diagrams, are modeled with processes. To define the architectural structure of composite active resources, which is often expressed in deployment, component or collaboration diagrams, SHE involves deriving scenario or use case independent instance structure diagrams to define cluster classes.

The UML profile for SHE smoothens utilization of the modeling language POOSL in the formalization stage, where the set of UML diagrams is unified into an executable model. Formalizing the behavior of processes is based on a small set of powerful statements. The formal semantics of POOSL allows one to combine these statements without limitations, providing ample means for formalizing the behavior expressed in collaboration, sequence, state or activity diagrams. One of the statements allows specifying concurrency within processes next to the usual concurrency among processes, which offers full support for modeling concurrent activities.

To enable evaluation of performance properties, SHE entails extending the POOSL model with reward variables based on the performance related information in the UML diagrams or the involved performance questions. In

the evaluation stage, the performance properties are evaluated against the executable model to be able to judge whether the system's concepts satisfy the requirements. Proper application of mathematical analysis techniques is supported by the formal semantics of POOSL. Evaluation of performance properties can be performed either by analytic computation or by empiric simulation. In the latter case, designers do not need to be experts of the applied analysis techniques as the underlying Markov chain is implicitly defined by the POOSL model and the accuracy of different types of long-run averages and variances is analyzed based on using library classes.

The SHE method is accompanied with two different tools. SHESim is a tool for constructing POOSL models that can be validated against the UML diagrams by interactive simulations. The Rotalumis tool enables high-speed execution of very large POOSL models in the evaluation stage. The formal semantics of POOSL prescribes how these tools manage aspects like probabilism, concurrency and time, which are inherent to a POOSL model.

Deriving POOSL models from UML diagrams is currently performed by hand. Future research includes an investigation on the possibility of a more automatic generation of POOSL models using a tool that integrates computer support for all three stages of system-level design.

This research is supported by PROGRESS, the Program for Research on Embedded Systems and Software of the Dutch Organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW.

## REFERENCES

- [1] L.J. van Bokhoven. *Constructive Tool Design for Formal Languages: From Semantics to Executing Models*. PhD thesis, Eindhoven University of Technology, Eindhoven (The Netherlands), 2002.
- [2] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice-Hall, Englewood Cliffs, New Jersey (U.S.A.), 1994.
- [3] M.C.W. Geilen, J.P.M. Voeten, P.H.A. van der Putten, L.J. van Bokhoven, and M.P.J. Stevens. Modeling and Specification Using SHE. *Journal of Computer Languages*, 27 (3): pp. 19-38, December 2001.
- [4] Object Management Group. *UML Profile for Schedulability, Performance and Time Specification*. OMG Adopted Specification ptc/02-03-02, Object Management Group, March 2002.
- [5] P. Kähkipuro. UML-Based Performance Modeling Framework for Component-Based Distributed Systems. In: R. Dumke (Ed.), *Proceeding of the 2nd International Conference on the Unified Modeling Language (UML'99)*, pp. 167-184. Springer (LNCS vol. 2047), 1999.
- [6] P.J.B. King and R.J. Pooley. Using UML to Derive Stochastic Petri Net Models. In: N. Davies and J. Bradley (Eds.), *Proceedings of the 15th UK Performance Engineering Workshop (UKPEW'99)* (Bristol, United Kingdom, July 22-23), pp. 45--56. University of Bristol, Bristol (United Kingdom), 1999.

- [7] S. Leue. Specifying Real-Time Requirements for SDL Specifications - A Temporal Logic-based Approach. In: P. Dembinski and M. Sredniawa (Eds.), *Protocol Specification, Testing and Verification XV*, pp. 19-34. Chapman and Hall, London (United Kingdom), 1997.
- [8] R. Milner. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, New Jersey (U.S.A.), 1989.
- [9] R. Mirandola and V. Cortelessa. UML Based Performance Modeling of Distributed Systems. In: A. Evans, S. Kent, and B. Selic (Eds.), *Proceedings of the 3rd Conference on the Unified Modeling Language (UML'00)*, pp. 178-193. Springer (LNCS vol. 1939), 2000.
- [10] K. Pawlikowski, H.D.J. Jeong, and J.S.R. Lee. On Credibility of Simulation Studies of Telecommunication Networks. *IEEE Communications Magazine*, 40 (1): pp. 132-139, 2002.
- [11] R.J. Pooley. Using UML to Derive Stochastic Process Algebra Models. In: N. Davies and J. Bradley (Eds.), *Proceedings of the 15th UK Performance Engineering Workshop (UKPEW'99)* (Bristol, United Kingdom, July 22-23), pp. 23-34. University of Bristol, Bristol (United Kingdom), 1999.
- [12] R.J. Pooley and P.J.B. King. The Unified Modeling Language and Performance Engineering. *IEE Proceedings - Software*, 146 (1): pp. 2-10, February 1999.
- [13] P.H.A. van der Putten and J.P.M. Voeten. *Specification of Reactive Hardware/Software Systems*. PhD thesis, Eindhoven University of Technology, Eindhoven (The Netherlands), 1997.
- [14] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Amsterdam (The Netherlands), 1999.
- [15] Schmietendorf and E. Dimitrov. Possibilities of Performance Modeling with UML. In: R. Dumke (Ed.), *Proceedings of the 4th Conference on the Unified Modeling Language (UML'01)*, pp. 78-95. Springer (LNCS vol. 2047), 2001.
- [16] B. Selic. The Real-Time UML Standard: Definition and Application. In: B. Werner (Ed.), *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*, pp. 770-772. IEEE Computer Society, Los Alamitos (U.S.A.), 2002.
- [17] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modeling*. Wiley and Sons, New York (U.S.A.), 1994.
- [18] B.D. Theelen, J.P.M. Voeten, L.J. van Bokhoven, P.H.A. van der Putten, A.M.M. Niemegeers, and G.G. Jong. Performance Modeling in the Large: A Case Study. In: N. Giambiasi and C. Frydman (Eds.), *Proceedings of the 13th European Simulation Symposium (ESS'01)* (Marseille, France, October 18-21), pp. 174-181. SCS-Europe, Ghent (Belgium), 2001.
- [19] B.D. Theelen, J.P.M. Voeten, and Y. Pribadi. Accuracy Analysis of Long-run Average Performance Metrics. In: F. Karelse (Ed.), *Proceedings of PROGRESS'01* (Veldhoven, The Netherlands, October 18), pp. 261-269. STW Technology Foundation, Utrecht (The Netherlands), 2001.
- [20] H.C. Tijms. *Stochastic Models; An Algorithmic Approach*. John Wiley & Sons, Chichester (England), 1994.
- [21] J.P.M. Voeten. Performance Evaluation with Temporal Rewards. *Journal of Performance Evaluation*, 50 (2/3): pp. 189-218, 2002.
- [22] ITU-T Recommendation Z.100. *Specification and Description Language (SDL)*, November 1999.