Automated Generation of Queuing Network Model from UML-based Software Models with Performance Annotations

ZHONGFU **X**U

AXEL **L**EHMANN

Bericht Nr. 2002-06

September 2002

Universität der Bundeswehr München

Fakultät für

INFORMATIK

Werner-Heisenberg-Weg 39 • D-85577 Neubiberg • Germany

Automated Generation of Queuing Network Model from UML-based Software Models with Performance Annotations

Zhongfu Xu and Axel Lehmann

Technical Report No. 2002-06

Universität der Bundeswehr München
Institut für Technische Informatik
85577 Neubiberg, Germany
Email: {xu, lehmann@informatik.unibw-muenchen.de}

Abstract

In order to analyze and evaluate performance properties of software design plans before they are implemented, we propose a methodology for automatic generation of queuing network models from UML-based software models. The UML extensions defined in *OMG UML profile for schedulability, performance and time specification* are used to model the software workload, active computer devices and their activities. A UML activity diagram is used to reformulate the software execution in terms of activities performed by computer devices, and a UML deployment diagram shows the computer devices and their relationships. In the proposed approach, performance information is annotated to these two diagrams. An XMI-based algorithm was developed to reformulate the performance information into a format suitable for automatic generation of queuing network-based performance models. A case study is performed to demonstrate our proposals with a simple example modeling a hypothetical Information Retrieval System .

Keywords: software performance analysis, unified modeling language (UML), extensible markup language (XML), XML-based metadata interchange (XMI), queuing network model (QNM)

Contents

1 Introduction

Software performance specifies the external effectiveness and internal efficiency of the software execution. As advocated in [Smith 1990, 2002], [Lindemann 1998], [Schmietendorf 1999], etc., performance properties of software design plans should be analyzed and evaluated before they are implemented. Consequently, performance bottlenecks and deficiencies can be identified early in the software life cycle, and the time and implementation costs can be minimized. In order to achieve this objective, stochastic models, such as queuing network model (QNM), should be generated from the design model as directly as possible. They should be analyzable by quantitative methods (such as analytic approach, numerical method or simulation) in order to obtain performance measures of interest.

Since years UML is widely used as a standard modeling language for visualizing, specifying, constructing and documenting artifacts of software systems under development [Booch 1999]. UML defines a rich set of modeling elements and diagrams capable of describing both the static and dynamic aspects of a software system. In order to enable tailoring and extending UML for special software modeling projects and meanwhile retaining the understandability and interoperability of the resulted models, three extension mechanisms are built in UML, i.e., stereotypes, tagged values and constraints. A stereotype is a new kind of metamodel element defined by the modeler based on an existing metamodel element. It has the same structure as the existing one, but with additional properties and semantics. A tagged value is a tag-value string pair that can be used to attach arbitrary information to any model element. A constraint textually describes semantic restrictions associated with a modeling element in some designated constraint language. These extension mechanisms extend the UML vocabulary, properties and semantics of UML modeling elements, respectively [Booch 1999] [UML1.4].

In this work software performance analysis is performed on the basis of software design model constructed using UML. The computer devices actively participating in a software design plan are identified from the design model. UML activity- and deployment diagrams are formulated by using the commercial UML tool Rational Rose [Rational] to describe the dynamic behavior and structural properties of computer devices. UML extensions defined in [OMG 2002] (*OMG-UML-Profile for Schedulability, Performance, and Time Specification*) are used to annotate performance-related information to these two diagrams. These UML diagrams are automatically transformed into XML (eXtensible Markup Language) [XML 1.0] documents by using the Unisys XMI (XML-based Metadata Interchange) [XMI 1.2] support available in Rational Rose. We developed an algorithm to extract performance information from the XML document. The extracted information is reformulated in a format suitable for automated generation of QNM for calculation of performance measures, such as software execution time, throughput, and utilization of each active computer device.

This report is organized as follows: In the following Section 2 some related work is introduced and discussed. Section 3 proposes how performance information is annotated to UML-based software models. In Section 4 the algorithm we developed for automated generation of QNM is explained. An example in Section 5 illustrates the application of the proposed methodology. Section 6 concludes the report and discusses directions of future work.

2 Related Work on Analyzing Software Performance based on UML Models

Following work has been performed to map UML-based software models to performance modeling mechanisms. It is common that in these work the UML models are constructed in a functionality-oriented way and performance aspects are not deliberately considered. As a result, performance models have to be equipped with performance-related information after they are obtained through mapping the UML models. This enlarges the gap between design and performance models in that performance information is not directly obtained from design models. Furthermore, the knowledge necessary for the chosen performance modeling mechanism might be an impediment for the software designer to contribute to provide performance information.

- [Pooley 1999a] presents ideas for exploiting UML diagrams for software performance modeling and conceptualizes a mapping from UML deployment diagrams to QNM. Three building blocks are identified to build the skeleton of a QNM: actors supplying workload, components providing services, and links joining components.

- [Pooley 1999b] shows that the mapping from combined UML collaboration- and state-chart diagrams to a **stochastic process algebra model** is possible for a very simple example. The internal behavior of each object in the collaboration diagram is shown by a UML state-chart diagram. All the states, transitions, triggers and actions in these state-chart diagrams are translated to elements of a stochastic process algebra model. The subset of UML features used in the example is very restricted. As seen by the authors, the mapping for more realistic UML designs would be unlikely to be so straightforward, and considerable further work is needed to develop a robust method for direct performance analysis of UML models in this way.

- The work in [King 1999] shows the possibility of transforming UML design models, concentrating on combined UML collaboration and state-chart diagrams, into a **stochastic Petri net** for software performance estimates. State-chart diagrams are used to show the internal behavior of objects in the collaboration diagram. The state-chart diagram for each object is mapped to one or more Petri nets by identifying states in a state-chart diagram with places in the Petri net and state changes with transitions. As stated by the authors, it is difficult to determine which transitions should be shared with other components when these individual Petri nets are combined using the collaboration diagram as a guide.

- The work in [Smith 2002] employs use case scenarios as a bridge between object-oriented development and Software Performance Engineering [Smith 1990]. Use case scenarios are described in terms of software functional components and their interactions by using UML sequence diagrams with MSC (Message Sequence Chart) extensions. The sequence diagrams are manually translated to **execution graphs** [Smith 1990] which are processed by the software performance engineering tool SPE•ED [Smith 1997] for performance measures of interest.

- In [Cortellessa 2000] a translation algorithm is developed to transform UML sequence diagrams into an **execution graph**. After specifying the processing overhead of each node in the execution graph, an early software performance prediction is made on the effectiveness of design alternatives in terms of elapsed times.

Some other work develops new notations and tools for performance modeling. The difficulty to understand and use these proprietary notations and tools makes them less attractive to general-purpose software performance analysis.

- In [Kabajunga 1998] a simulator is built around the UML sequence diagram. The names of objects in the sequence diagram and time durations of messages among objects are encoded in a driver file. Based on this driver file, a simple discrete event simulator generates an animated sequence diagram as a trace of events. For this work only fairly standard examples are tested, and its effectiveness on complex software systems needs to be examined.

- [Kähkipuro 2001] develops a textual notation, *Performance Modeling Language*, to normalize the UML-based software models for performance modeling by filtering out features that are not relevant to performance analysis.

- In [Arief 2000] a simulation framework called *Simulation Modeling Language* (SimML) is developed. This framework identifies components and actions that are common for simulation programs. A tool is constructed to enable the representation of structure and behavior of software system using UML. The performance-related information collected by the tool is stored in a set of component and action classes in the SimML framework and is used to generate a simulation program for performance calculation.

- The work in [Hoeben 1998] uses the script language of Rational Rose to automatically create new model elements and enter performance-related information into Rational Rose models. A prototype tool called *Model Extractor* is developed to transport all performance information to a database. Another tool called *Performance Analyser* is developed to compute the estimated software response times by using the Little´s Law.

In [Anciano 2001] some extensions to a subset of UML diagrams are proposed in order to include performance information in a system design. Although these extensions are not in accordance with those defined in [OMG 2002], this work establishes that "a fully user-centered approach should be based on extending UML diagrams with performance-oriented annotations that could be further translated to well-known solution methods in the performance-modeling domain".

Our work makes following main contributions with regard to the related work:

- the obtained QNM need not be extended with additional performance-related information, since the annotated UML models are intended to contain sufficient information.
- only standard UML building blocks (including extensions defined in [OMG 2002]) are used to construct and annotate the software model. No new notation and tool are needed. Based on the XMI support available in a variety of UML tools, the automated generation of queuing network-based performance models would be straightforward.

3 Annotating UML-based Software Model with Performance Information

In this work the software model contains UML activity- and deployment diagrams. It realizes the domain concepts (in Figure 1) that are identified in [OMG 2002] for modeling and analyzing real-time systems.

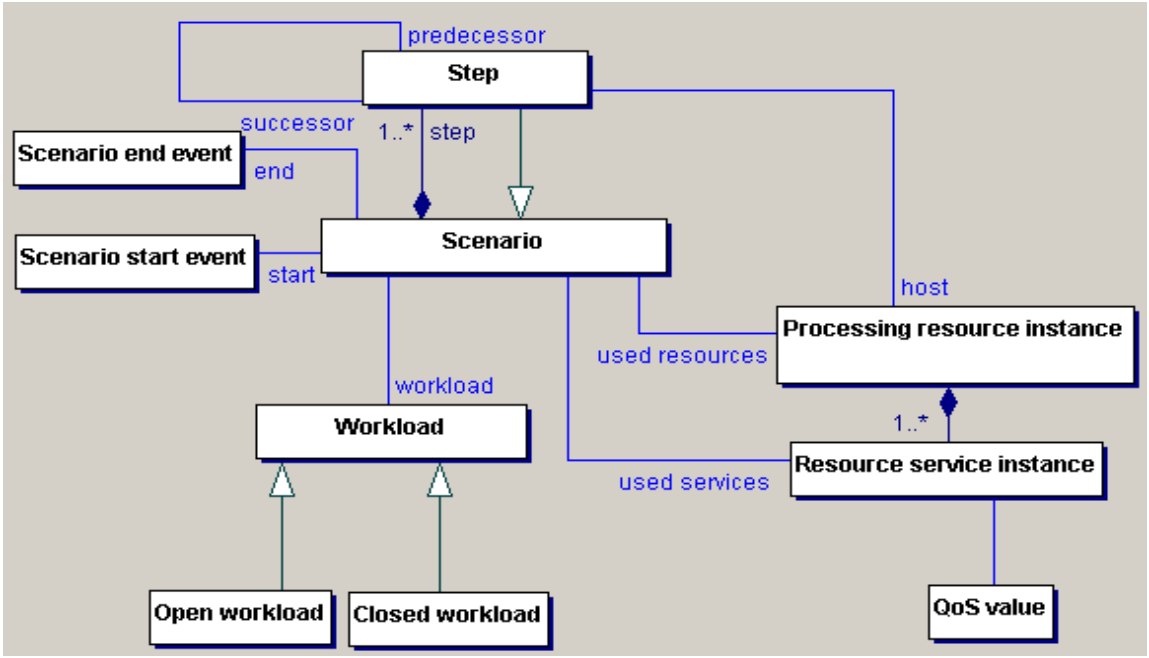

Figure 1: Domain concepts for modeling and analyzing real-time systems [OMG 2002].

Scenario: A scenario describes the dynamic usage of a set of resources and their services. It involves a set of action executions that are performed by processing resource instances.

Scenario start event: This event occurs when a scenario starts its execution.

Scenario end event: This event occurs when the scenario ends its execution.

Workload: A workload specifies the intensity of demand on the execution of a specific scenario as well as the required or estimated response time for the workload. It has two subtypes: open workload and closed workload. An **open workload** represents a stream of jobs that arrive at the system from an unbounded outside source. The job arrival process is characterized by a predetermined arrival pattern. Upon processing completion, the jobs leave the system. A **closed workload** is generated by a fixed number of users who continually interact with the system. The users enter requests, examine responses, and submit the next request after a specified delay time (i.e., think time). No jobs enter the system from and leave the system to outside. The population of jobs in the system is constant. The closed workload is characterized by the number of users and the average delay between the receipt of a response and submission of the next request.

Processing resource instance: It represents a run-time entity that offers services to perform a specific scenario.

Resource service instance: It is a specific instance of a resource service offered by a processing resource instance.

QoS value: It specifies the value of a QoS characteristic offered by a resource service instance. The value can range from a single number to a complex structured value such as a probability distribution.

Step: A step is an increment in the execution of a software scenario. It is performed by a processing resource instance and is related to other steps in predecessor/successor relationships.

In this work, the execution of a software scenario is represented by a UML activity diagram in terms of a set of ordered atomic activities. An atomic activity is performed merely by one computer device without

collaboration from other devices. It realizes the domain concept *step* in Figure 1. The processing resource instances that execute the software scenario are realized by computer devices that perform the atomic activities, such as processors, I/O devices. Generally, computer devices are not explicitly shown in software design models. Their behavior is represented by operations of design elements (i.e., design objects/subsystems). They are identified by decomposing the design element operations into atomic activities. Computer devices provide service instances by performing atomic activities. They will be mapped to queue or delay servers in the QNM to be built for performance calculation. A UML deployment diagram is used to represent the computer devices and their relationships.

A UML activity diagram shows all the atomic activities within each scenario execution and their chronological sequence. The flow of control from activity to activity in the activity diagram originates from a start state and terminates in an end state. The scenario start and end events in Figure 1 are represented by the start and end state in the activity diagram respectively.

The execution of a software scenario is invoked by the system users who interact with the system. The workload imposed by the users on the system is modeled by *PAopenLoad* stereotype or *PAclosedLoad* stereotype that are defined in [OMG 2002]. These two stereotypes are described in Table 1 and 2.

Table 1: Specification of *PAopenLoad* and *PAclosedLoad*.

stereotype	tags	domain concept
PAopenLoad	PArespTime PApriority PAoccurrence	open workload
PAclosedLoad	PArespTime PApriority PApopulation PAextDelay	closed workload

Table 2: Tag definitions for *PAopenLoad* and *PAclosedLoad*.

tag	stereotype	type	multiplicity	description
PArespTime	PAopenLoad PAclosedLoad	PAperfValue[1]	1	Assumed or expected delay between the starting and completing instants of the software scenario
PApriority	PAopenLoad PAclosedLoad	Integer	1	Priority of the workload
PAoccurrence	PAopenLoad	RTarrivalPattern[2]	1	Pattern of job arrival process
PApopulation	PAclosedLoad	Integer	1	Number of system users
PAextDelay	PAclosedLoad	PAperfValue	1	Average delay between the receipt of a response and the submission of the next request

[1] PAperfValue specifies a complex performance value and is an instance of the list type of TVL (Tag Value Language) (appendix A in [OMG 2002]). Its elements are typically mixtures of strings, numeric literals, TVL variable names, and TVL expressions. For example, an assumed performance value that conforms to exponential distribution with mean value of 1 millisecond is specified as ('assm', 'dist', 'exponential', 1, 'ms'). (see [OMG 2002] for its detailed format description in extended BNF).
[2] RTarrivalPattern specifies a concrete value of arrival pattern and is also an instance of the list type of TVL. For example, a Poisson process with an average arrival rate of 1 job per second is specified as ('unbounded', 'Poisson', 1, 's'). (see [OMG 2002] for its detailed format description in extended BNF).

Workload information is annotated to the start state in the activity diagram using a textual note. The note consists of the tags of *PAclosedLoad* or *PAopenLoad* and their values. Using the BNF notation, the syntax of the note is represented as follows:

<note for open workload> ::= "{" 'open' "," < response time> "," <priority> "," <arrival pattern> "}"
<response time> ::= 'PArespTime' = <PAperfValue>
<priority> ::= 'PApriority' = <Integer>
<arrival pattern> ::= 'PAoccurrence' = <RTarrivalPattern>

<note for closed workload> ::= "{" 'closed' "," < response time> "," <priority> "," <population> "," <think time> "}"

<response time> ::= 'PArespTime' = <PAperfValue>
<priority> ::= 'PApriority' = <Integer>
<population> ::= 'PApopulation' = <Integer>
<think time> ::= 'PAextDelay' = <PAperfValue>

The atomic activities in the activity diagram are modeled by the stereotype *PAstep* defined in [OMG 2002] for the execution of an action. The QoS values for each atomic activity are annotated by a textual note consisting of values of tags of *PAstep* stereotype. Table 3 lists some of the predefined *PAstep* tags used in this work. It contains also three new tags we add to *PAstep* specific to our need, i.e., *PAname*, *PAdevice*, and *PAcallNext*.

Table 3: Tag definitions for *PAstep*.

tag	type	multiplicity	description
PAdemand	PAperfValue	1	Total execution demand of the atomic activity on its host device
PAdelay	PAperfValue	[0..1]	Value of inserted delay (wait or pause) within the atomic activity
PArep	Integer	[0..1]	Number of times the atomic activity is repeated (more than once)
PAinterval	PAperfValue	[0..1]	Time interval between successive repetitions of an atomic activity, when it repeated
PAname	String	1	Name of the atomic activity
PAdevice	String	1	Name of the device that performs the atomic activity
PAcallNext	PAperfValue	1	Duration of the execution of an atomic activity before it invokes another atomic activity that follows

Using the BNF notation, the syntax of the note attached to each atomic activity is represented as follows:

<note for atomic activity> ::= "{" <name> "," <device> "," <demand> "," [<delay>]","
<call next> "," [<repetition>]"," [<repetition interval>] "}"
<name> ::= 'PAname' = <String>
<device> ::= 'PAdevice' = <String>
<demand> ::= 'PAdemand' = <PAperfValue>
<delay> ::= 'PAdelay' = <PAperfValue>
<call next> ::= 'PAcallNext' = <PAperfValue>
<repetition> ::= 'PArep' = <Integer>
<repetition interval> ::= 'PAinterval' = <PAperfValue>

The active computer devices in the deployment diagram are modeled by the *PAhost* stereotype defined in [OMG 2002] for processing resources. The QoS values for each device are annotated by a textual note consisting of values of tags of *PAhost* stereotype. Table 4 lists some of the predefined *PAhost* tags used in this work. It contains also two new tags we add to *PAhost* specific to our need, i.e., *PAmultiplicity* and *PAcapacity*.

Table 4: Tag definitions for *PAhost*.

tag	stereotype	type	multiplicity	description
PAutilization[1]	PAhost	Real	1	The percentage of the time the device is busy providing service
PAthroughput[1]	PAhost	Real	1	The rate at which the device performs activities
PAschdPolicy	PAhost	Enumeration: {'FIFO', 'PR', 'PS', 'PPS', 'LIFO', 'infinite server'}[2]	1	Policy that controls access to the device
PAmultiplicity	PAhost	Integer	1	Number of the devices of the same type
PAcapacity[3]	PAhost	Integer or 'infinite'	[0..1]	Maximum number of jobs permitted to wait for service from the device

[1] Values are specified after performance calculation
[2] FIFO: first-in-first-out; PR: preempt-resume; PS: processor-sharing; PPS: priority-processor-sharing; LIFO: last-in-first-out.
[3] Not be specified for devices whose PAschdPolicy is 'infinite server'.

Using the BNF notation, the syntax of the note for each device is represented as follows:

<note for device>::="{" 'utilization' "," 'throughput' ","<scheduling>"," <multiplicity> "," [<capacity>] "}"
<utilization> ::= 'PAutilization= '
<throughput> ::= 'PAthroughput= '
<scheduling> ::= 'PAschdPolicy' = 'FIFO' | 'PR' | 'PS' | 'PPS' | 'LIFO' | 'infinite server'
<multiplicity> ::= 'PAmultiplicity' = <Integer>
<capacity> ::= 'PAcapacity' = <Integer> | 'infinite'

TVL variables can be used to specify the QoS of devices and their atomic activities. For example: A TVL scalar variable ($HL_Instruction) is defined in the note for a CPU to specify the time that the CPU needs to execute a high-level language instruction ($HL_Instruction=(2, 'us'). This variable can then be used in the note for a CPU activity to specify the execution demand (PAdemand='assm', 'mean', 10*$HL_Instruction).

During formulating the activity diagram, following software execution behavior needs to be handled elaborately:

- Forking and joining of control

Forking of control generates two or more concurrent flows of control that join together upon completion. In UML activity diagram the control forking and joining are represented by synchronization bars in pairs. The synchronization bar for control forking has one incoming transition and two or more outgoing transitions. The synchronization bar for control joining has two or more incoming transitions and one outgoing transition. In order to handle multiple occurrences of forking and joining of control, we propose to name the synchronization bar pairs and number the names. For example, when forking and joining of control occurs twice, we name the two synchronization bar pairs with (*fork1*, *join1*) and (*fork2*, *join2*).

- Splitting of control

Splitting of control generates two or more concurrent flows of control. These flows of control leave the system upon completion without joining together. In UML activity diagram the control splitting is represented by a synchronization bar which has one incoming transition and two or more outgoing transitions. The concurrent flows of control join together at another synchronization bar with an outgoing transition pointing directly to the end state of the activity diagram. Likewise, we propose to name the synchronization bar pair and number the name, for example, as *splitter1* and *splitterJoin1*.

- Conditional branching

In UML activity diagram the conditional branching is represented by a decision icon that has one incoming transition and two or more outgoing transitions. Each outgoing transition reaches an activity and initiates a possible flow of control under a Boolean guard condition associated with this transition. The sum of all the probabilities associated with the outgoing transitions must be equal to one. All the possible flows of control join together at another decision icon that has two or more incoming transitions and one outgoing transition. Again,

we propose to name the decision icon pair and number the name. For example, in the case of two occurrences of conditional branching we name the two decision icon pairs as (*decider1*, *merger1*) and (*decider2*, *merger2*).

- Repetition of a sequence of atomic activities

In some cases, a sequence of atomic activities will be repeated more than once. In each repetition the activities in the sequence are performed one after another. In order to specify the start and end points of the activity sequence, we propose to introduce a pair of *dummy* activities to the activity diagram. One dummy activity occurs before the first activity in the sequence and indicates the start point and the other one occurs after the last activity in the sequence and represents the end point. The repetition number of the sequence is indicated by a textual note attached to the first dummy activity. The possible multiple dummy activity pairs are distinguished from each other by having different names, such as (*loop1*, *loopEnd1*), (*loop2*, *loopEnd2*) and so forth.

The following activity diagram (Figure 2) is an example for modeling the software execution behavior stated above. The dummy activities for the repeated sequence of activities have names and a textual note (indicating the repetition number of the sequence) attached to the first dummy activity. The synchronization bars (for parallel execution) and decision icons (for conditional branching) have only names.

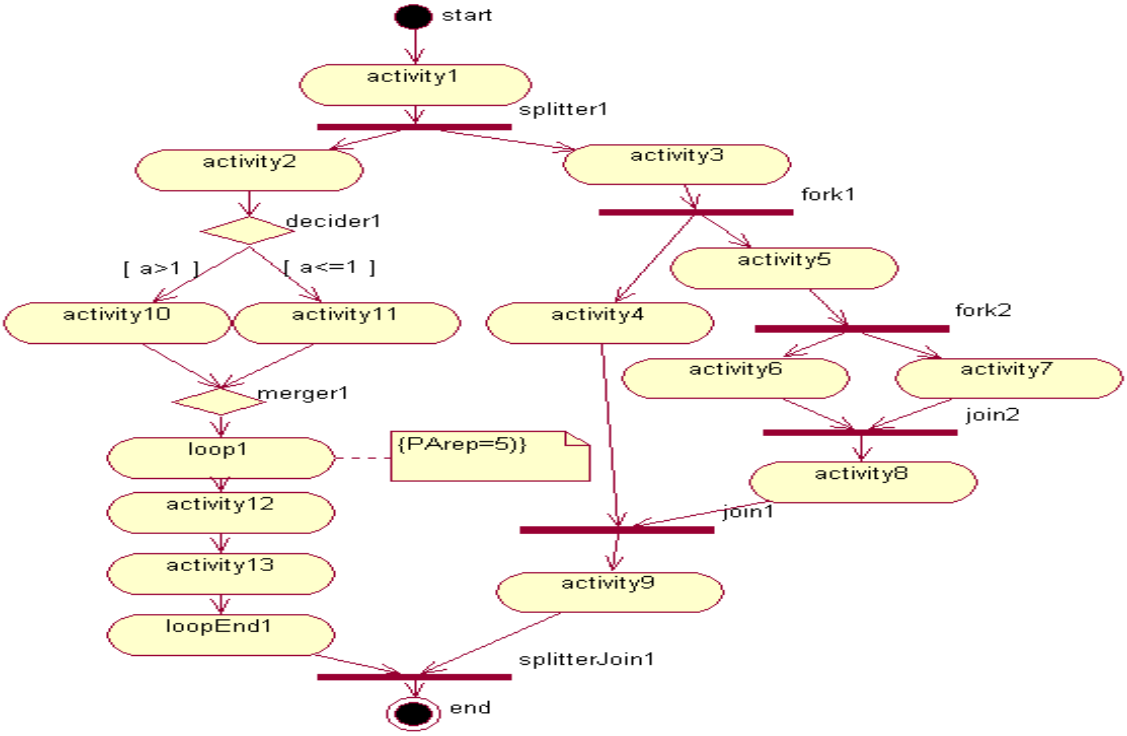

Figure 2: An example for modeling parallel execution, conditional branching and repetition of an activity sequence.

4 Automated Generation of Queuing Network Model for Performance Evaluation

The deployment- and activity diagrams are transformed into an XML document using the XMI support which is available in a variety of UML tools, such as Rational Rose (see [Jeckle 2002] for a list of UML tools with XMI support). The syntax of the XML document is defined by normative UML DTDs (see [XMI 1.2] for an example) corresponding to UML Interchange Metamodels [UML 1.4]. The elements of deployment- and activity diagrams are represented by a set of XML elements in the resulted XML document. Table 5 contains some examples.

Table 5: Examples for mapping UML model elements to XML elements.

XML element	attributes	UML element
UML:Diagram	xmi.id, name, toolName, diagramType, style, etc.	Activity diagram, deployment diagram
UML:Node	xmi.id, name, etc.	Devices in deployment diagram
UML:PseudoState	xmi.id, name, kind, etc.	States in activity diagram (initial, branch, fork, join)
UML:FinalState	xmi.id, name, etc.	End state in activity diagram
UML:ActionState	xmi.id, name, etc.	Activities in activity diagram
UML:Transition	xmi.id, name, etc.	Transitions between states and activities in activity diagram
UML:Transition.source		State or activity from which the transition originates
UML:Transition.target		State or activity in which the transition terminates
UML:Transition.guard		Guard condition associated with this transition
UML:StateVertex.incoming		Transitions which terminate in the state or activity
UML:StateVertex.outgoing		Transitions which originate from the state or activity
UML:Stereotype	xmi.id, name, baseClass, etc.	Stereotype contained in UML diagrams
UML:Stereotype.extendedElement		UML model element with this stereotype
UML:Comment	xmi.id, name, etc.	Note attached to UML model element
UML:Comment.annotatedElement		UML model element with this note

The XML elements in Table 5 hold all the information about the structural features and quantitative behavior of the devices involved in software execution. In order to reformulate the information in a format suitable for building a queuing network-based performance model, we developed an algorithm whose functional structure is shown in Figure 3.

The algorithm produces the **software execution trace** and **device profile**. The software execution trace begins with the name of the start state, note for workload, ends with the name of the end state, and contains the notes for the activities (i.e., activity name, name of device performing the activity, service demand on device, inserted delay, execution duration before calling the activity that follows, number of repetitions, and interval between repetitions). It might also contain names of dummy activities and repetition number for repeated activity sequence, names of pseudo states for splitting, forking and joining, conditional branching and guard conditions associated with condition branching. The algorithm also produces a profile for all devices involved in software execution (i.e., device name, control policy for access to the device, number of devices of the same type, buffer capacity). The results of the algorithm are used to build the queuing network-based performance model for performance evaluation:

QNM service centers (i.e., queue servers and delay servers): The devices included in the device profile are mapped to service centers of QNM.

Queue-scheduling discipline and multiplicity of service centers: They are respectively specified by values of *PAschdPolicy* tag and *PAmultiplicity* tag of the corresponding device.

Queue capacity of queue server: It is specified by the value of *PAcapacity* tag of the corresponding device.

Distribution of workload among QNM service centers: The host device of each activity in the execution trace is specified by the value of *PAdevice* tag. All the activities performed by one device constitute the workload on this device.

Processing of workload on QNM service centers: Each activity performed by a device is described in the execution trace by the value of *PArep* (representing number of repetitions), value of *PAdemand* (representing service demand on host device), value of *PAcallNext* (representing execution duration before calling the following activity), value of *PAdelay* (representing inserted delay), and value of *PAinterval* (representing the interval between successive repetitions).

Path of execution locus within QNM: The execution path follows the predecessor/successor relationships of activities specified in the execution trace. For each activity its host device has been specified. When dummy activities for sequence repetition, pseudo states for execution forking and splitting, or pseudo sates for conditional execution are included in the execution trace, they contribute to understanding the movement of execution locus within QNM. The path of execution locus implies the interconnections of QNM service centers.

Workload of QNM: It is specified in the head of the execution trace, by values of *PArespTime* and *PApriority*, value of *PAoccurence* for open workloads and values of *PApopulation* and *PAextDelay* for closed workloads.

In some cases, the obtained QNM may need to be extended to model some special software execution behavior. (see Case Study for an example)

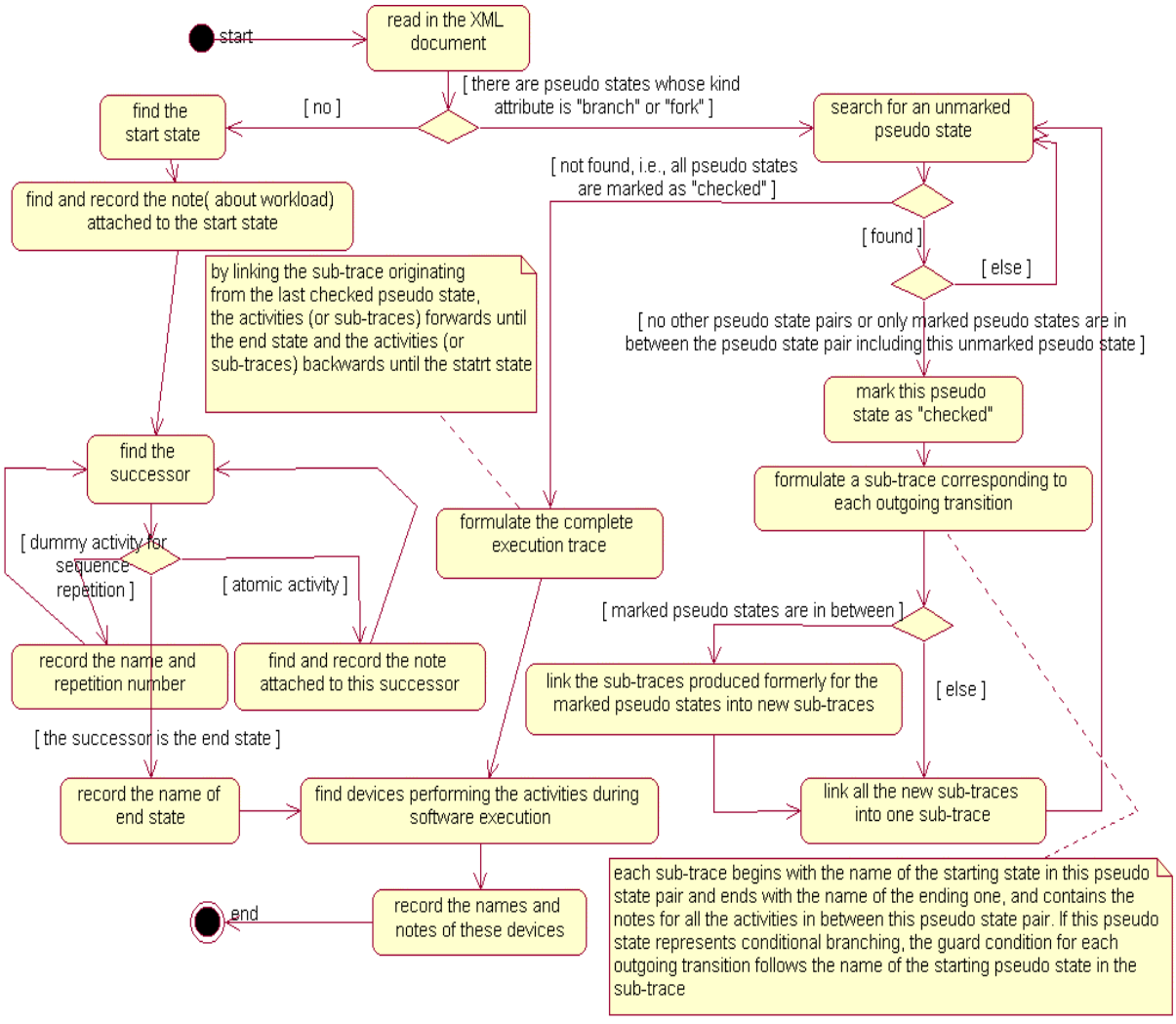

Figure 3: Algorithm for extracting and reformulating performance information.

Depending on the specific types of behavior and parameters specified for the computer devices involved in software execution, suitable methods and tools can be used to solve the QNM for performance measures of interest. The obtained performance measures are used to check whether the predefined performance requirements for software under consideration are met. When they are met, the software development evolves. Otherwise, performance bottleneck analysis is carried out and design alternatives for performance improvements have to be identified. Performance analysis processes then are redone to evaluate the design alternatives. In some cases it may also be necessary to define more realistic performance requirements.

5 Case Study

As an example we consider a hypothetical Information Retrieval System. For this simple example, one use case is identified: the user uses the system to get the requested information. The following design model (Figure 4) is built to show the realization of this use case.

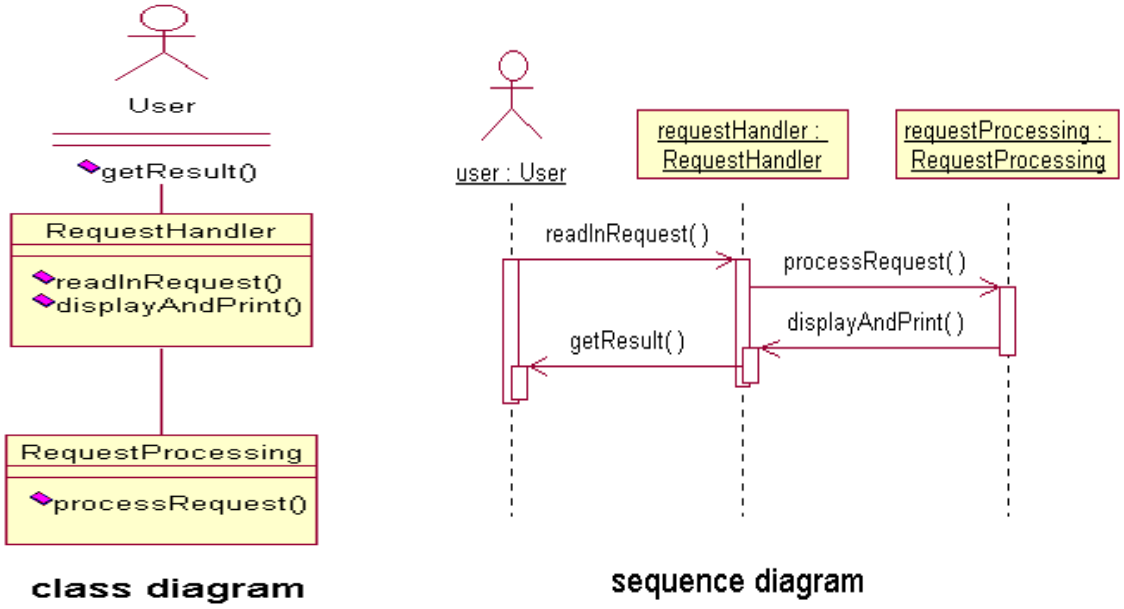

Figure 4: Design model for Information Retrieval System.

As shown in Figure 4, instances of three design classes (i.e., *user*, *requestHandler*, *requestProcessing*) interact to execute the use case. The operations performed by *requestHandler* (i.e., *readInRequest()* and *displayAndPrint()*) are considered to be atomic activities that are performed by a computer device *console*. The operation performed by *requestProcessing* (i.e., *processRequest()*) are decomposed into three atomic activities: *getRequest()* and *getInformation()* that are performed by a computer device *CPU*, and *retrieve()* by *storage device*. The three computer devices are modeled by *PAhost* and shown in a UML deployment diagram (Figure 5). Performance-related information about the computer devices is represented by the textual notes. The atomic activities are modeled by *PAstep*. Their chronological sequence is shown in a UML activity diagram (Figure 6). Performance information is annotated to atomic activities. The workload in open. The arrival process of user requests is Poisson process with an average arrival rate of 1/(30 seconds).

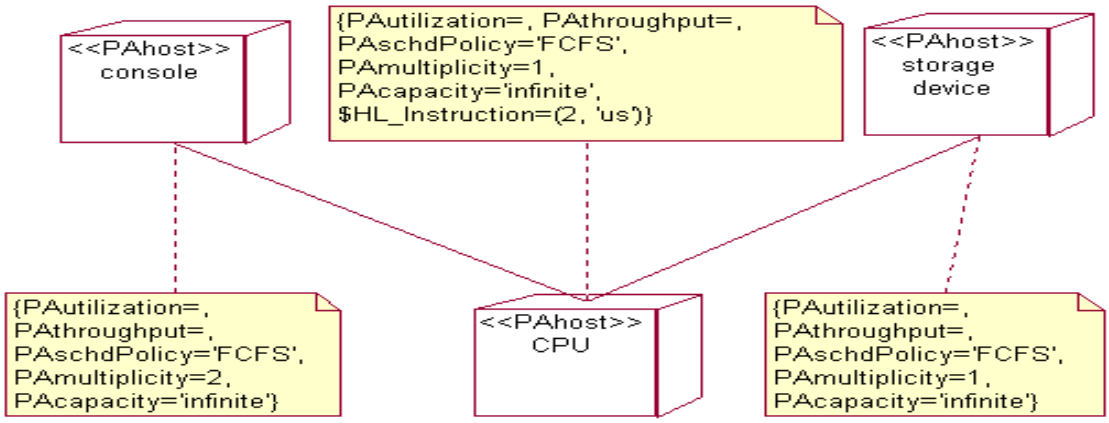

Figure 5: Computer devices and associated performance information.

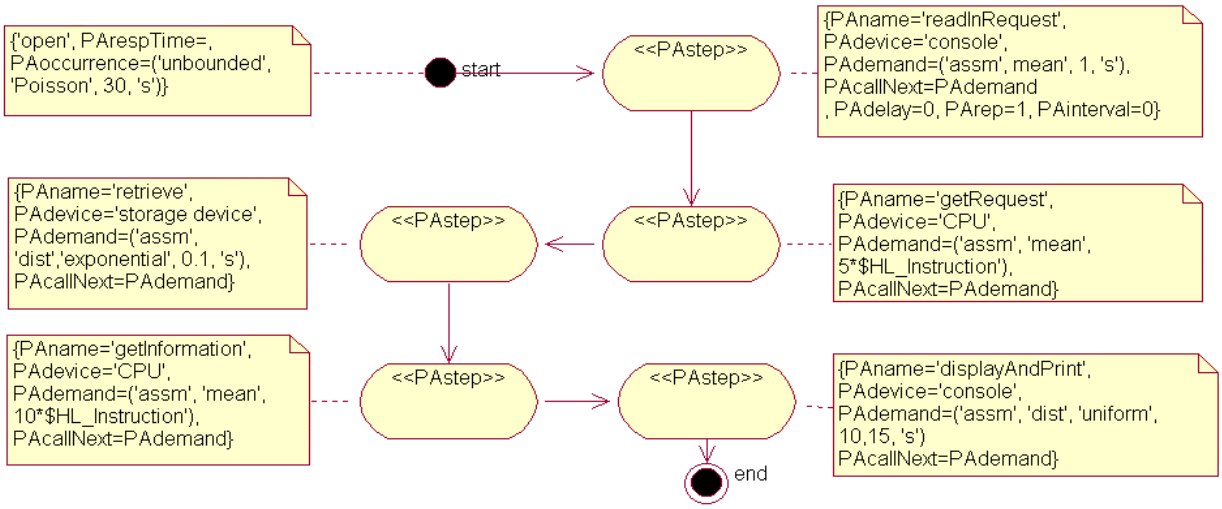

Figure 6: Atomic activities and associated performance information.

The activity- and deployment diagrams are transformed into an XML document using the Unisys XMI support in Rational Rose. Using this XML document as input, the algorithm introduced in Section 4 produces following results:

Software execution trace: {start; open, PAoccurrence=('unbounded', 'Poisson', 1/30, 's'); readInRequest(), console, PAdemand=('assm', 'mean', 1, 's'), PAcallNext=('assm', 'mean', 1, 's'), PAdelay=0, PArep=1, PAinterval=0; getRequest(), …; retrieve(), …; getInformation(),…; displayAndPrint(), …; end}

Device profile: {console, FCFS, 2, infinite; CPU, FCFS, 1, infinite, $HL_Instruction=(2, 'us'); storage device, FCFS, 1, infinite}

Based on these results, a QNM is constructed. The three devices in the device profile are mapped to queue servers in the QNM: console, CPU and storage device. Since an individual user is not interrupted by other users when he or she is interacting with the system, the QNM is extended with two artificial queue servers: *allocator* and *releaser*. Figure 7 shows the topology of the extended QNM. The parameters for this extended QNM is summarized in Table 6.

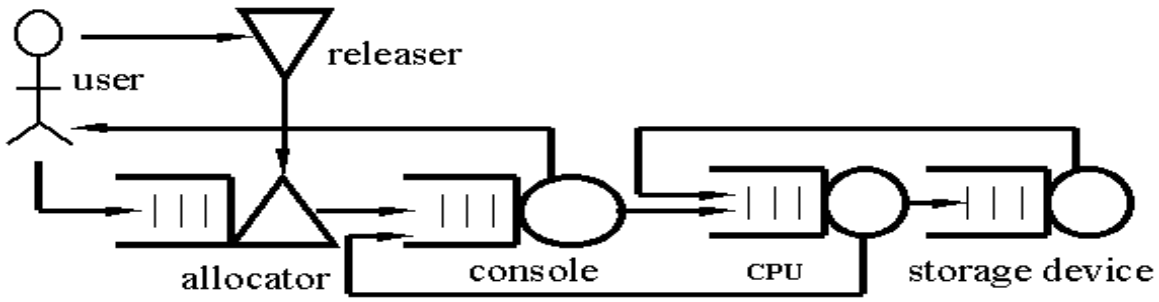

Figure 7: Extended QNM.

Table 6: Model parameters for QNM.

service center	multiplicity	activities	service demands
console	2	readInRequest	Assumed, mean value=1 sec
		displayAndPrint	Uniform, within [10sec, 15sec]
CPU	1	getRequest	Assumed, mean value =5*$HL_Instruction
		getInformation	Assumed, mean value =10*$HL_Instruction
			$HL_Instruction=(2, 'us')
storage device	1	retrieve	Assumed, exponential distribution, mean value=0.1 sec
--- all the service centers have infinite capacity --- queue-scheduling discipline for all the service centers is FCFS --- all the atomic activities are executed only once and have no inserted delay --- for all the atomic activities, PAcallNext is the same as PAdemand			

The user requests come from *user*. The *allocator* is blocked and new user requests wait at the queue of *allocator* if a user request is being in service at present. After the service is finished (i.e., information required by the user has been displayed and printed), *user* informs *releaser* to unblock *allocator* to let another user request pass through to *console*. The path of execution locus within QNM is as follows:
(readInRequest, console) → (getRequest, CPU) → (retrieve, storage device) → (getInformation, CPU) → (displayAndPrint, console) → (getResult, user).

The extended QNM is solved by the commercial simulation tool SIMUL8 [simul8] to obtain performance measures. In the simulation experiment the model was run 10 times. Each run has a results collection period of 600000 simulated seconds and a confidence level of 99%. The average number of user requests completed per simulation run is 19983.7. The average response time of jobs is 17.55±0.2% seconds. The minimum response time is 12.5 seconds, and the maximum is 51.33±6.5% seconds. It was shown that 98.95% of the user requests have response times less then 30 seconds. Table 7 shows the utilization and throughput of the computer devices.

Table 7: Simulation results for computer devices.

	console1	console2	CPU	storage device
utilization	21.9%	23.07%	0.005%	0.33%
throughput	0.032	0.034	0.067	0.033

It is also shown that the maximum queue length of users before one console is 2.4±22%, the average queuing time of users is 1.3±3% seconds, the average queuing time of users that actually have to queue (i.e. average of non-zero queuing times) is 6.12±2% seconds, the expected maximum queuing time during the results collection period is 32.17±14.3% seconds, and 87.88% of the queuing times are less then 5 seconds. The maximum queue length of users before another console is 2.2±19.5%, the average queuing time of users is 1.72±3.2% seconds, the average queuing time of users that actually have to queue (i.e. average of non-zero queuing times) is 7.09±1.4% seconds, the expected maximum queuing time during the results collection period is 33.59±12% seconds, and 84.48% of the queuing times are less then 5 seconds. These results can be used to check whether the predefined performance requirements are met. When not met, design alternatives should be recognized and reevaluated.

6 Conclusion and Future Work

In order to analyze and evaluate performance properties of software design plans before their implementation, we propose in this work a methodology for automated generation of queuing network-based performance model from UML-based software models with performance annotations: Computer devices that actively participate in software execution are modeled by *PAhost* stereotypes and represented in a UML deployment diagram. Their activities are represented by *PAstep* stereotypes and shown in a UML activity diagram. Workload on the software is modeled by *PAclosedLoad* or *PAopenLoad* stereotype. Performance information associated with devices, their activities and software workload is described by a set of tagged values in form of textual notes. An XMI-based algorithm is developed to reformulate performance information in a format suitable for automated generation of queuing network model. Solution of queuing network model provides insights into software responsiveness and usage of computer devices. A case study demonstrates the proposed methodology using a simple example.

One topic of the future work would be the definition of a QNM schema that can capture both the structure and behavior of a QNM. According to the schema a QNM will be automatically generated using the outcomes of the algorithm. The obtained QNM will be described in an XML document so that it can be understandable and processable for various QNM analysis tools.

In the current work the activity- and deployment diagrams are constructed manually and used as the basis for the generation of performance model. Our work will continue to automate the entire process. The functional-oriented design model (without performance information) will be transformed into an XML document. (It is assumed that the design model contains a UML sequence diagram to represent the software execution in terms of design objects and their interactions.) An XML parser will be developed to process the XML document. The operations performed by design objects in the sequence diagram will be identified. The parser will provide two GUIs for entering performance-related information. One GUI is used to gather performance-related information for workload, for each operation of design object and for the atomic activities within the operation. Another GUI is used to enter performance information for the computer devices that were identified during decomposing the operations of design objects into atomic activities. Using these information a QNM will be generated according to the predefined schema and exported into an XML document. This way, some benefits can be achieved: Entering performance information can be facilitated, e.g., by prompting keywords (i.e., tags) and their default values; Performance information are recorded separately from the design model and will not clutter it; This process applies also to the design models constructed by using other UML tools, e.g., Together Enterprise [TogetherSoft]. Optionally, the performance information can also be used to formulate the activity- and deployment diagrams in XMI syntax, which can be visualized in the UML modeling tools through reverse engineering.

References

[Anciano 2001] J.-L. Anciano and R. Puigjaner. "Annotating UML Diagrams to Carry Out Performance Analysis", in *Proceedings Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'01)*, pp.215-222, 2001.

[Arief 2000] L.B. Arief and N.A. Speirs. "A UML Tool for an Automatic Generation of Simulation Programs", in *Second Int. Workshop on Software and Performance (WOSP2000)*, Ottawa, Canada, September 18-20, 2000.

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

[Cortellessa 2000] V. Cortellessa and R. Mirandola. "Deriving a Queuing Network based Performance Model from UML Diagrams", in *Second Int. Workshop on Software and Performance(WOSP2000)*, Ottawa, Canada, September 18-20, 2000.

[Hoeben 1998] F. Hoeben and M. Sterk, "Performance in Distributed Object Systems", Software Engineering Research Center, Utrecht, The Netherlands, 1998.

[Jeckle 2002] M. Jeckle. "Unified Modeling Language (UML) tools", 2002. (http://www.jeckle.de/umltools.html)

[Kabajunga 1998] C. Kabajunga and R. Pooley. "Simulating UML Sequence Diagrams*", in R. Pooley and N. Thomas, eds., UK PEW´98-Proceedings of 14th UK Performance Engineering Workshop*, Edinburgh, July 1998.

[Kähkipuro 2001] P. Kähkipuro. "UML-Based Performance Modeling Framework for Component-Based Distributed Systems", in R. Dumke et al., eds., *Performance Engineering - State of the Art and Current Trends, LNCS 2047*, pp.167-184, Springer-Verlag, Berlin, 2001.

[King 1999] P. King and R. Pooley. "Using UML to Derive Stochastic Petri Net Models", in *N. Davies and J, Bradley, eds., UK PEW´99-Proceedings of the 15th UK Performance Engineering Workshop*, Department of Computer Science, the University of Bristol, July 1999.

[Lindemann 1998] C. Lindemann. *Performance Modeling with Deterministic and Stochastic Petri Nets*, John Wiley&Sons, 1998.

[OMG 2002] OMG: *UML Profile for Schedulability, Performance, and Time Specification, OMG Adopted Specification: ptc/02-03-02*, March 2002. (http://www.omg.org/cgi-bin/doc?ptc/2002-03-02)

[Pooley 1999a] R. Pooley and P. King. "The Unified Modeling Language and Performance Engineering", *IEE Proceedings-Software*, Vol. 146, No. 1, pp.2-10, February 1999.

[Pooley 1999b] R. Pooley. "Using UML to Derive Stochastic Process Algebra Models", in *N. Davies and J, Bradley, editors, UK PEW´99-Proceedings of the 15th UK Performance Engineering Workshop*, Department of Computer Science, the University of Bristol, July 1999.

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

[Schmietendorf 1999] A. Schmietendorf. "Anwendung von Modell-Lösungen für ein Performance-Engineering" in MMB-Mitteilungen, Nr36, Herbst 1999.

[Simul8] Simul8 Corporation, Simul8 Manual and Simulation Guide. (http://www.simul8.com)

[Smith 1990] Connie U. Smith. *Performance Engineering of Software Systems*, Addison-Wesley, Reading, Massachusetts, 1990.

[Smith 1997] C. U. Smith and L. G. Williams. "Performance Engineering Evaluation of Object-oriented Systems with SPE•ED". In *Computer Performance Evaluation: Modeling Techniques and Tools*, No. 1245, (R. Marie, et al. eds.), Springer-Verlag, Berlin, 1997.

[Smith 2002] C. U. Smith and L. G. Williams. "Performance and Scalability of Distributed Software Architectures: An SPE Approach". *Parallel and Distributed Computing Practices*, 2002.

[TogetherSoft] TogetherSoft Corporation. (http://www.togethersoft.com/)

[UML 1.4] OMG: *Unified Modeling Language Specification, version 1.4*, September 2001. (http://www.omg.org/technology/documents/formal/uml.htm)

[XMI 1.2] OMG: *XML Metadata Interchange (XMI) Specification, version 1.2*, January 2002. (http://www.omg.org/technology/documents/formal/xmi.htm)

[XML 1.0] W3C Recommendation: *Extensible Markup Language (XML) 1.0 (Second Edition)*, October 2000. (http://www.w3.org/TR/2000/REC-xml-20001006)