PERFORMANCE EVALUATION OF SOFTWARE ARCHITECTURES WITH QUEUING NETWORK MODELS

Simonetta Balsamo   Roberto Mamprin   Moreno Marzolla

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

KEYWORDS

Unified Modeling Language, Queuing Networks, Software Performance Evaluation, Distributed Systems

ABSTRACT

We propose an approach for software performance modeling based on UML software specifications and queuing network performance models. We consider the integration of performance and specification model to provide a tool for quantitative evaluation of a software architecture at the design phase in the software development cycle. The approach derives a performance model starting from an annotated Unified Modeling Language (UML) specification, based on a subset of the standard UML Profile for Schedulability, Performance and Time Specification. More specifically, we consider a set of UML diagrams, i.e., Use Case, Activity and Deployment diagrams, and we propose an algorithm for deriving a product-form queuing network performance model. Then the queuing network model is easily analyzed with product-form algorithms to obtain a set of performance indices that are used to provide feedback at the software architectural design level. The analysis cycle can be iterated to meet given performance goals or to compare different software alternatives. The approach has been implemented as a prototype tool written in Java.

1   INTRODUCTION

Large software systems play a vital role in many commercial and industrial infrastructures. Complex e-commerce sites, data acquisition networks and distributed computing environment are very expensive to develop and maintain. Such systems should provide an adequate level of performances in order to be used. Early performance analysis can help to identify and correct problems from the early stages of the software development life cycle, in order to compare design alternatives or to identify system bottlenecks. Early identification of performance problems is desirable as the cost of design change increases with the later phases in the software development cycle.

Performance evaluation should be integrated in the software development process (Balsamo et al., 2004b; Smith, 1990; Smith and Williams, 2002). The SPE approach by Smith (Smith, 1990) was the first integrated approach for development and performance evaluation of software systems. Several approaches have been proposed in the last years that can be classified by considering various characteristics, including the software and the performance model. A comprehensive review of the software performance methods can be found in (Balsamo et al., 2004b).

Many approaches consider UML (Object Management Group (OMG), 2001) as the software system specification. This choice is motivated by the fact that UML has a large users base, and is able to model many different aspects (static, dynamic, behavioral) of a system. Moreover, UML provides standard extension mechanisms based on additional constructs. Recently the UML Profile for Schedulability, Performance and Time Specification has been adopted as an OMG standard (Object Management Group (OMG), 2002a). It allows the definition of requirements for performance and scheduling analysis, and the specification of quantitative informations directly in the UML model. Some recent approaches for software performance analysis are based on the UML Profile (Xu et al., 2003). As the performance model is concerned, various methods consider formal and analytical models such as Queuing Network (QN), stochastic process algebras or stochastic Petri nets, and some are based on simulation. It has been observed that QN are the preferred model, taking advantage of the high level of abstraction of the QN formalism that allows easier mapping and feedback of software systems, especially in computer-based software development process (Balsamo et al., 2004b; Grassi and Mirandola, 2004).

In this paper we propose an approach for software performance modeling based on UML as the software description notation, and Queuing Networks (Kleinrock, 1975) as the performance model. The motivation of the choice of QN models is twofold. First, the high level of abstraction of the model makes it easy to define a direct correspondence between software components and performance model elements, so that we can easily derive the QN from UML diagram specification. This allows us to provide a direct feedback of performance results at the software design level. The second motivation is that we aim to identify a simple product-form QN (Kleinrock, 1975) in order to obtain performance measures by computationally efficient algorithms (Reiser and Lavenberg, 1980).

We consider the software performance evaluation cycle

described by the following steps:

1. Definition of the performance requirements of the software system.

2. UML specification of the software system.

3. Transformation of the software specification into a performance model based on QN, using a suitable transformation algorithm.

4. Analysis of the QN and derivation of performance indices.

5. Feedback of performance results on the software model elements.

6. Analysis of the software performance results and possible iteration from step 2 (e.g., if the performance requirements are not met or different architectural implementations have to be examined)

Starting from the UML software model annotated with performance-oriented annotations based on the *UML Performance Profile* (Object Management Group (OMG), 2002a), we consider UML Use Case, Activity and Deployment diagrams. We define an algorithm for translating the annotated software model into a QN based performance model. The performance model is solved using an appropriate solution technique and performance results provide feedback at the software specification level by the annotations. Hence the software performance analysis cycle can be iterated to meet given performance requirements or to compare different software design alternatives.

The proposed approach can be integrated with other different methods for software performance analysis in a more general environment that can provide a set of tools for quantitative analysis of software systems. Specifically, as observed in (Balsamo et al., 2004a) software designers would take advantage of the combined use of different methodologies to evaluate the performance of software artifacts. To this aim, for example, the proposed approach can be easily integrated with the recently developed tool UML-Ψ (Balsamo et al., 2004a) providing a simulation-based evaluation of UML-based software systems.

The proposed method has been implemented as s Java program called UML Queuing Network Evaluator (UML-QNE). UML-QNE parses the annotated model produced by a UML CASE tool. In particular, we use the freely available ArgoUML tool (ArgoUML), that can export the software model in XML Metadata Interchange (XMI) format, which is a standard XML-based representation of UML models. UML-QNE parses the XMI file, builds the QN model and applies the MVA algorithm (Reiser and Lavenberg, 1980). Fig. 1 illustrates the steps described above and the structure of the proposed approach. In particular, steps 3 through 6 correspond to the UML-QNE Java tool.

The paper is organized as follows. In Section 2 we briefly discuss the proposed approach with respect to some recent model-based methods for software performance analysis. Section 3 presents the proposed approach by introducing the software model based on the UML Profile, the QN
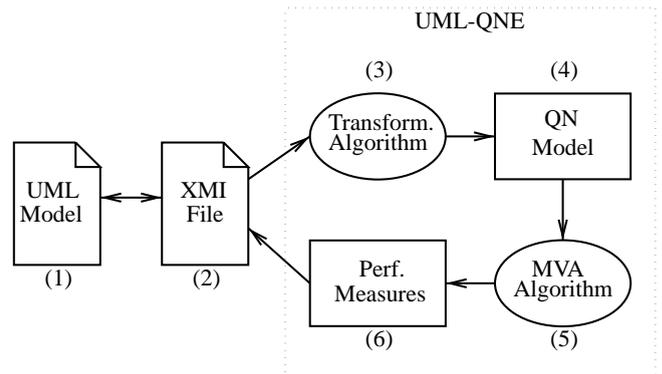

Figure 1: Software Performance Steps and UML-QNE Structure

performance model and the proposed translation algorithm. The method implementation is described in Section 4 and a simple case study are described in Section 5. Finally, Section 6 presents conclusions and open problems.

2 SOFTWARE PERFORMANCE EVALUATION

Different approaches dealing with performance evaluation of software systems have been proposed in the literature (Proceedings of WOSP 2000; Proceedings of WOSP 2002; Proceedings of WOSP 2004). In the following, we briefly recall some approaches based on UML as the software specification notation.

King and Pooley in (King and Pooley, 2000) derive performance models based on Generalized Stochastic Petri Nets from UML collaboration and statechart diagrams. In (Pooley and King, 1999), Pooley and King describe how the various kinds of UML diagrams can be used for performance evaluation purposes. The approach adds textual notations to UML diagrams to include useful information for performance evaluation (e.g., time labels in sequence diagrams). Such annotations are used to produce more complete models of software systems. Bernardi et al. (Bernardi et al., 2002) derive a SPN model from UML state and sequence diagrams. Cortellessa and Mirandola present in (Cortellessa and Mirandola, 2002) a methodology for translating UML sequence, use case and Deployment diagrams into performance model based on Extended Queuing Networks, using an intermediate transformation into Execution Graphs (Smith, 1990). Gomaa and Menascé (Gomaa and Menascé, 2001) derive a QN-based performance model from UML class and collaboration diagrams to represent the interconnection pattern of a distributed software architecture. Gu and Petriu (Gu and Petriu, 2002) and Petriu and Shen (Petriu and Shen, 2002) derive performance models based on Layered Queuing Network models from a description of Software Architecture (SA) based on annotated UML Activity diagrams. Lindemann et al. (Lindemann et al., 2002) develop an algorithm for deriving performance models based on Generalized Semi-Markov Processes from UML State and Activity

diagrams. Kähkipuro (Kähkipuro, 2001) proposes a framework on UML notation for describing performance models of component-based distributed systems. The performance model is based on Augmented Queuing Networks.

A few works also consider simulation-based performance models for software systems (Arief and Speirs, 2000; De Miguel et al., 2000; Hillston and Wang, 2003; Marzolla, 2004). In general, simulation-based approaches derive a simulation model from UML specifications; the simulation model is implemented as a simulation program which is then executed. Simulation results are computed as confidence intervals for some figures of merit (e.g., utilization and throughput of resources, average execution times).

Analytical performance models have some advantages over those based on simulation. Some classes of analytical models can be solved very efficiently either exactly or by approximation. Moreover, some analytical models allow parametric solution, with respect to some parameter of the performance mode. Hence, in this case we can easily carry on parametric performance analysis under different model assumptions (e.g., by varying parameters' value), avoiding the re-evaluation of the entire model. On the other side, generally speaking, system performance analysis with simulation-based models leads to a high computation time cost for model development, validation and execution. Moreover, simulation provides a solution in terms of estimated values, with confidence intervals. However, the main advantage of simulation models is their generality and wide applicability, i.e., any system which can be described can also be analyzed by simulation, while analytical models require specific assumptions, and efficient solution algorithms apply under special constraints (Lavenberg, 1983). Efficient solution algorithms are only available for analytical models satisfying certain constraints.

We propose an approach that derives a QN model from annotated UML specifications. The software specification model must satisfy relatively mild constraints (service demands can only be exponentially distributed). Under these assumptions, the QN model is in product-form and can be efficiently solved by using the MVA algorithm (Reiser and Lavenberg, 1980). Differently from other approaches, we consider the direct derivation of a simple product-form network from a UML software specification model based on a variation of the standard UML Performance Profile (Object Management Group (OMG), 2002a).

3 THE PROPOSED APPROACH

In this section we present the proposed approach by introducing the annotated UML software specification model, the QN performance model and the transformation algorithm (step 3 in Fig. 1).

3.1 The software model

The software system is described in term of the following UML diagrams: Use Case diagrams, representing workloads applied to the system; Deployment diagrams, describing the
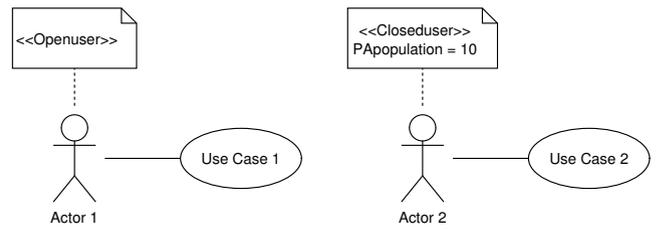

Figure 2: Example of annotated Use Case diagrams

available physical resources (processors) where computations take place; Activity diagrams, describing both the order in which resources are used, and the corresponding service demand.

Use case diagrams Each actor in a Use Case diagram may represent a stream of requests arriving at the system. There may be an unlimited sequence of requests (open workload), or a fixed population of users requiring service from the system (closed workload). Actors representing open workloads are stereotyped as ≪ OpenUser ≫, while actors representing closed workloads are stereotyped as ≪ ClosedUser ≫. ≪ OpenUser ≫ actors

For the latter class of actors it is necessary to specify the total number of requests circulating in the system; this is done with the PApopulation tagged value associated to the actor. Multiple actors may be present in the same system, meaning that there are multiple concurrent streams of requests. Each actor has an associated use case representing the computations triggered by each actor. The details of the computations are described by an Activity diagram which must be associated to each Use Case.

Fig. 2 shows an example of annotated Use Case diagram, where Actor 1 represents an open workload and Actor 2 represents a closed population of 10 requests circulating in the system.

Deployment diagrams Deployment diagrams are used to model the physical resources available in the system. Each resource is represented by a node in the Deployment diagrams. Each node, which must be stereotyped as ≪ node ≫, represents a processor with a given scheduling policy. The following tagged values can be associated to Deployment diagram nodes:

PAschedpolicy Defines the scheduling policy of that processor, which can be one of "FIFO" (First-In-First-Out), "LIFO" (Last-In-First-Out) and "PS" (Processor Sharing).

PArate defines the processing rate (speed) of the processor. This means that a user with a service request of S time units will complete in $S/PArate$ time units.

PAservers defines the number of processors concurrently executing the requests. This tag can be used to represent a multiprocessor resource having N equal proces-

sors executing in parallel. All the processors share the same queue of pending requests.

Fig. 3 shows an example of annotated Deployment diagram. Each node represents a processor with the characteristics described with the tagged values.

Activity diagrams Having described the workloads and the physical resources available in the system, it is now necessary to specify how the resources are used, that is, which computations are performed in the system. Computations are described by associating an Activity diagram to Use Cases. In this way, the workload represented by an actor will trigger the computation represented by the Activity diagram associated to the corresponding Use Case.

Each action state of an Activity diagram, stereotyped as ≪ ServiceCenter ≫, represents a computation which requires service to one resource. The following tagged values can be specified to provide informations for building the performance model:

PAresource is the name of the resource (node in one of the Deployment diagrams) from which service is requested.

PAoccurrence represents the interarrival time of this service request. The tag can be specified only if the QN performance model is an open QN (described in the following).

Action states are linked each other with a predecessor-successor relationship. This relationship is used to model the sequence of computations which are executed by a request. In general, one action state may have multiple successors. In this case, each transition must be labeled with the probability that the transition is traversed. This models nondeterminism in the sequence of actions which are executed.

Fig. 4 represents an example of annotated Activity diagram. Transitions are annotated with the PAprob tag showing the probability of traversing the corresponding arc. Values of the PAresource tags refers to node names of the Deployment diagram in Fig. 3.

Note that our approach does not support synchronization bars in Activity diagrams. Synchronization bars are used to split the execution flow in multiple, concurrent threads or to join multiple threads into a single flow. While synchronization bars could be translated into fork and join nodes in the QN model, the resulting QN would not be in product form, thus requiring more complex solution algorithms, which may produce only approximate results instead of exact ones.

3.2 The Performance Model

We derive the performance model from the UML specification, as sketched in Section 1, by defining a multiclass QN (Lavenberg, 1983). A Queuing Network is a set of service centers (also called nodes or service stations). Each service center is formed by a queue and a set of identical servers.
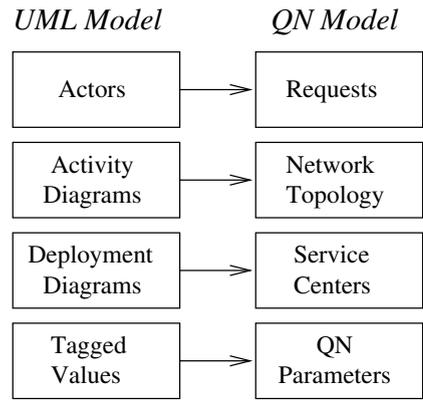

Figure 5: Mapping Between UML and Performance Model Elements

Requests join the queue and are serviced by the first server becoming idle, according to a specific queuing discipline.

UML model components are translated into the corresponding QN model elements as follows. Each node in the Deployment diagrams defines a service center. From Actors in Use Case diagrams we identify the type of QN model, e.g., open, closed or mixed QN. Finally, from Activity diagrams we derive the network topology, that is the behavior of the classes of users circulating through the system. The mapping between UML and performance model elements is illustrated in Fig. 5.

Open QN can be generated from Use Case diagrams in which the actor is stereotyped as ≪ OpenUser ≫. Note that no tagged values are defined for this kind of actor, as externally arriving requests can be defined by the PAoccurrence associated to each action state. This means that the action state can be executed also by an external stream of requests reaching the system at the given rate. Closed QN are made of a number of service centers in which a fixed population of requests circulates. Each request receives service from a service center, after which it is routed to a (possibly different) another service center. This kind of network is generated from an actor stereotyped as ≪ ClosedUser ≫, where the associated PApopulation tag is used to specify the number of requests. Mixed QN are an extension of the previously described networks: in mixed networks there are R different classes of circulating requests. Each class of requests moves through the network differently from requests of other classes. Some classes of requests move through the network with a closed topology, i.e., external arrivals and departures are not allowed, other classes of requests move according to an open topology. The tagged value PAserviceTime specifies the average amount of time that requests spend at a service center. Mixed QN are generated from Use Case diagrams with multiple actors, and where each actor represents one user class, depending on its stereotype.

3.3 The transformation algorithm

The proposed algorithm, named Algorithm UML-QNE, translates an annotated UML specification into the QN
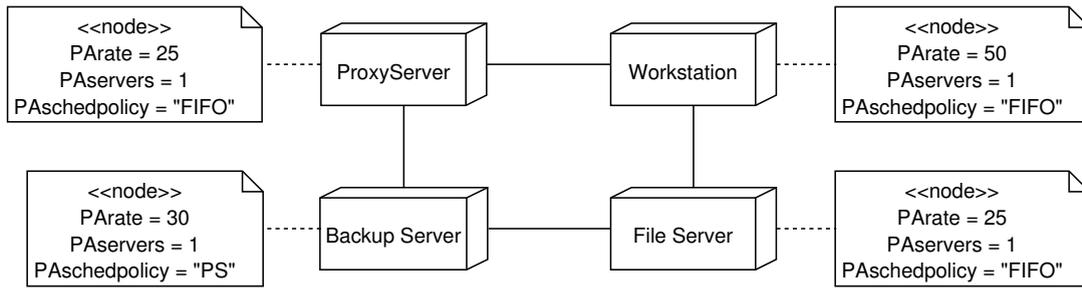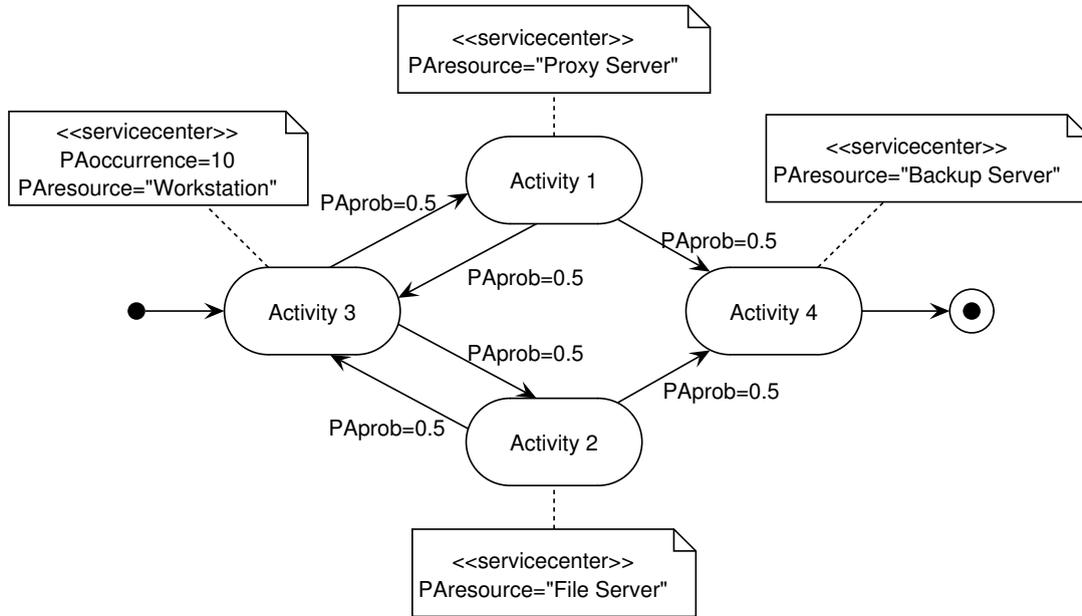
Figure 3: Example of annotated Deployment diagram

Figure 4: Example of Annotated Activity Diagram

model. We use the model notation shown in Table 1. In the algorithm we denote with *TagName*(X) the value of tag *TagName* associated to UML element X.

N	Number of nodes in the UML Deployment diagram
S_i	i-th service center
μ_i	Service rate of service center S_i
λ_i^C	Arrival rate of class C customers at service center S_i
NS_i	Number of servers in service center S_i
$\mathbf{P}^C$	$N \times N$ routing matrix for class C customers

Table 1: Notation used in Algorithm 1

The algorithm is illustrated in Fig. 1 and works according to the following steps:

1. For each Deployment diagram node we define a corresponding service center; if N is the number of nodes in the Deployment Diagrams, we define N service centers $S_1, S_2, \ldots S_N$.

2. For each Actor in Use Case diagrams we define a class of customers in the QN.

3. For each Activity diagram associated to an Actor we define the routing matrix for the current class of customers

according to the predecessor-successor relationship of Action states. Routing probabilities are obtained from the `PAprob` tag associated to UML transitions.

The computational complexity of Algorithm 1 is $O(N + T + A)$, where N is the number of nodes in the Deployment Diagrams, T and A are the number of transitions and action states in all the Activity diagrams, respectively.

4   IMPLEMENTATION

A prototype Java tool called UML Queuing Network Evaluator has been implemented in order to demonstrate the approach described in the previous section. The tool parses an XMI representation of an annotated UML model to derive the QN model. The performance model is then solved using the MVA algorithm (Reiser and Lavenberg, 1980).

The internal structure of UML-QNE is shown in Fig. 6. UML-QNE builds an internal representation of the whole performance model. The QN model is made of resources and users (requests). Each user is associated to a `Topology` object, representing the routing matrix which applies to that user class. There are two different kind of users, which are `OpenUser` and `ClosedUser`; they describe open and
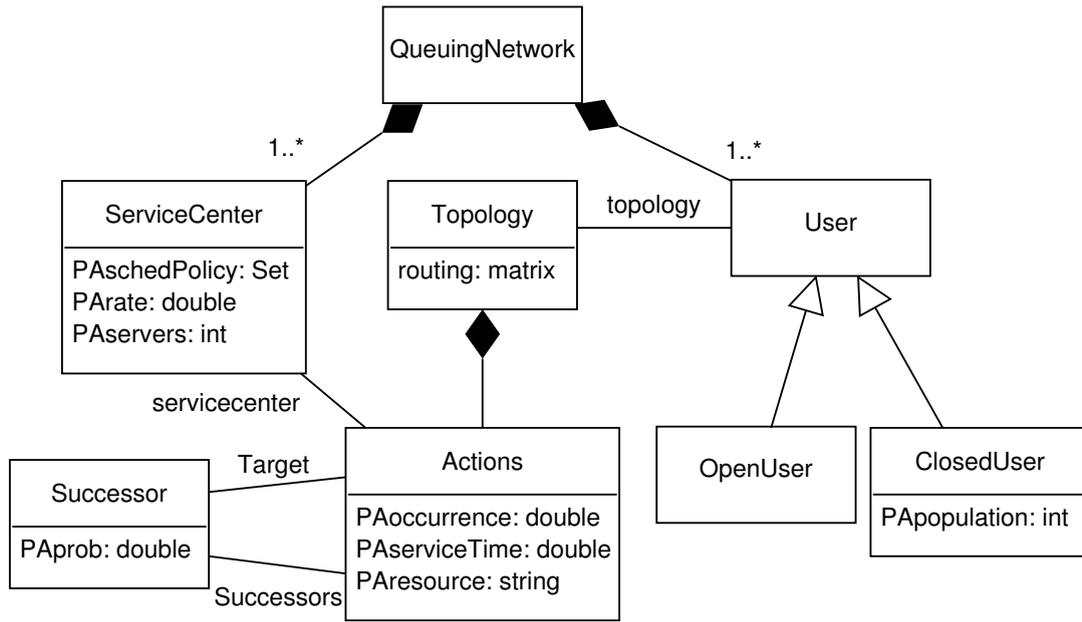
Figure 6: Class diagram of the UML-QNE Java tool

Algorithm 1 QN Model Generation

  for all Deployment diagram node R_i, $i = 1 \ldots N$ **do**
    $S_i :=$ New Service Center
    $\mu_i := 1/PAserviceTime(R_i)$
    $NS_i := PAservers(R_i)$
  end for
  Let $C \leftarrow 0$
  for all Actor A **do**
    Initialize routing matrix $\mathbf{P}^C$ for class C to zero
    $AD :=$ Activity diagram associated to A
    for all Transition t from action state a_i to a_j of Activity diagram AD **do**
      $R_k := PAhost(a_i)$
      $R_l := PAhost(a_j)$
      $P_{k,l}^C := PAprob(t)$
    end for
    if A is a ClosedUser **then**
      Set class C as a closed chain with $PApopulation(A)$ requests
    else
      Set class C as an open chain
      for all Action state a of Activity diagram AD **do**
        if $PAoccurrence(a)$ is defined **then**
          $R_i := PAhost(a)$
          $\lambda_i^C := PAoccurrence(a)$
        end if
      end for
    end if
    Let $C \leftarrow C + 1$ {New Customer Class}
  end for

---

closed user classes, respectively. Each user can move through the service centers according with its routing matrix, which is described by an Activity diagram. A `Topology` object is made of a number of `Action` objects. Each `Action` is associated to a `ServiceCenter` node, representing the fact that the given action requests service to its associated service center. Actions are linked in a predecessor-successor relationship, each transition having an associated probability.

UML-QNE currently understands the UML models produced by the ArgoUML CASE tool (ArgoUML). ArgoUML model encoding is based on XMI (Object Management Group (OMG), 2002b), which is an XML-based notation for UML model representation. UML-QNE parses the XMI file and builds an internal representation of the UML model. This internal representation, whose structure is illustrated in Fig. 6, is then used to build the QN model, which is finally solved by using the MVA algorithm. Performance results include, for each service center S_i, the mean number of customers (mean queue length N_i), the utilization U_i, throughput X_i, and the mean waiting time R_i. Given that each service center in the QN model represents one resource in the UML model, performance results can be directly interpreted at the software architectural level.

5 AN EXAMPLE

The QN model of Fig. 7 is an example of a network derived from the Deployment and Activity diagrams of Fig. 3 and 4, respectively, assuming that the Activity diagram is associated to an actor stereotyped as $\ll$ OpenUser $\gg$.

In this example, there is only one job class, which corresponds to the only actor in the UML model. The QN is made of four service centers $S_1, \ldots S_4$ which correspond to the four resources represented in the Deployment diagram of Fig. 3. S_1 corresponds to the "Proxy Server" node, S_2 corresponds to the "File Server" node, S_3 corresponds to the "Workstation" node and S_4 corresponds to the "Backup Server" node. The topology of the QN is derived from the Activity diagram of Fig. 4: each UML transition from activity i to activity j is mapped into an edge from server S_i to
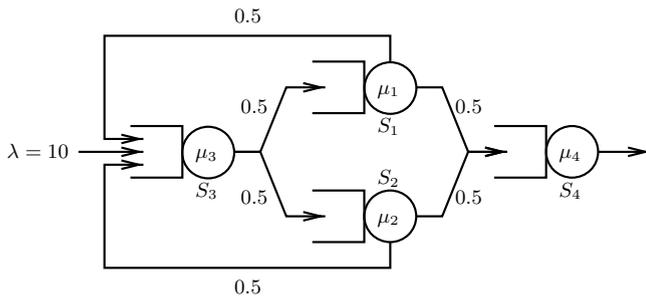
Figure 7: Open QN. Service rates are $\mu_1 = \mu_2 = 25$, $\mu_3 = 50$, $\mu_4 = 30$

server S_j. The service rates $\mu_1, \ldots \mu_4$ are derived from the `PArate` tags of the Deployment diagram.

The analysis of the product-form QN of Fig. 7 provides a set of average performance indices, that include mean number of requests, component utilization and throughput and average response time. Table 2 shows some numerical results for the specified set of parameters value, for each system component. The performance analysis can provide indication for possible system modification and further software performance evaluation can be iterated by choosing a different set of parameters value, to be inserted in the UML annotation.

Resource	N_i	U_i	X_i	R_i
Workstation	0.67	0.4	20.0	0.03
File server	0.67	0.4	10.0	0.069
Proxy Server	0.67	0.4	10.0	0.069
Backup Server	0.6	0.33	10.0	0.05

Table 2: Performance Results for the QN of Fig. 7: Mean number of Customers (N_i), Utilization (U_i), Throughput (X_i) and Mean Response Time (R_i)

6 CONCLUSIONS

In this paper we proposed an approach for performance modeling of UML software architectures. The approach considers annotations based on the UML Performance Profile, and derives a performance model based on Queuing Network. Performance evaluation of the QN by efficient algorithms provides a set of steady-state performance indices that characterize the software system behavior. Such results are immediately reported back in the UML software specification model as tagged values in the diagrams. The approach has been illustrated by implementing a prototype Java tool which automates the performance model derivation and solution phases.

Future works include the integration of UML-QNE in a more general framework for quantitative analysis of Software Architectures based on different techniques such as including simulation (Marzolla, 2004).

REFERENCES

ArgoUML. ArgoUML – Object-oriented design tool with cognitive support. http://www.argouml.org/.

L. B. Arief and N. A. Speirs. A UML tool for an automatic generation of simulation programs. In Proceedings of WOSP 2000 , pages 71–76.

S. Balsamo, A. D. Marco, P. Inverardi, and M. Marzolla. Experimenting different software architectures performance techniques: A case study. In Proceedings of WOSP 2004 , pages 115–119.

S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering*, 30(5), May 2004b.

S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable Petri net models. In Proceedings of WOSP 2002 .

V. Cortellessa and R. Mirandola. PRIMA–UML: a performance validation incremental methodology on early UML diagrams. In Proceedings of WOSP 2002 .

M. De Miguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, and S. Piekarec. UML extensions for the specifications and evaluation of latency constraints in architectural models. In Proceedings of WOSP 2000 , pages 83–88.

H. Gomaa and D. A. Menascé. Performance engineering of component-based distributed software systems. In *Performance Engineering – State of the Art and Current Trends*, volume 2047 of *Lecture Notes in Computer Science*, pages 40–55. Springer, 2001.

V. Grassi and R. Mirandola. Towards automatic compositional performance analysis of component-based systems. In Proceedings of WOSP 2004 , pages 59–63.

G. Gu and D. C. Petriu. XSLT transformation from UML models to LQN performance models. In Proceedings of WOSP 2002 .

J. Hillston and Y. Wang. Performance evaluation of UML models via automatically generated simulation models. In S. A. Jarvis, editor, *UK Performance Engineering Workshop*, pages 64–78, July 2003.

P. Kähkipuro. UML-based performance modeling framework for component-based distributed systems. In R. R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, editors, *Performance Engineering*, volume 2047 of *Lecture Notes in Computer Science*. Springer, 2001. ISBN 3-540-42145-9.

P. J. B. King and R. J. Pooley. Derivation of Petri net performance models from UML specifications of communications software. In B. R. Haverkort, H. C. Bohnenkamp,

and C. U. Smith, editors, *Computer Performance Evaluation / TOOLS*, volume 1786 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2000. ISBN 3-540-67260-5.

L. Kleinrock. *Queueing Systems*, volume 1. J. Wiley, 1975.

S. S. Lavenberg. *Computer Performance Modeling Handbook*. Academic Press, New York, USA, 1983.

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

M. Marzolla. *Simulation-Based Performance Modeling of UML Software Architectures*. PhD Thesis TD-2004-1, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy, Feb. 2004.

Object Management Group (OMG). Unified modeling language (UML), version 1.4, Sept. 2001.

Object Management Group (OMG). UML profile for schedulability, performance and time specification. Final Adopted Specification ptc/02-03-02, OMG, Mar. 2002a.

Object Management Group (OMG). XML Metadata Interchange (XMI) specification, version 1.2, Jan. 2002b.

D. C. Petriu and H. Shen. Applying the UML performance profile: graph grammar-based derivation of LQN models from UML specifications. In T. Field, P. G. Harrison, J. Bradley, and U. Harder, editors, *Computer Performance Evaluation / TOOLS*, volume 2324 of *Lecture Notes in Computer Science*. Springer, 2002. ISBN 3-540-43539-5.

R. J. Pooley and P. J. B. King. The Unified Modeling Language and performance engineering. In *IEE Proceedings – Software*, volume 146, pages 2–10, February 1999.

Proceedings of WOSP 2000. *Proc. of the Second International Workshop on Software and Performance (WOSP 2000)*, Ottawa, Canada, Sept. 2000. ACM Press.

Proceedings of WOSP 2002. *Proc. of the Third International Workshop on Software and Performance (WOSP 2002)*, Rome, Italy, July 24–26 2002. ACM Press.

Proceedings of WOSP 2004. *Proc. of the Fourth International Workshop on Software and Performance (WOSP 2004*, Redwood Shores, California, Jan. 14–16 2004. ACM Press.

M. Reiser and S. S. Lavenberg. Mean value analysis of closed multichain queueing networks. *Journal of the ACM*, 27: 313–322, 1980.

C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, 1990.

C. U. Smith and L. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

J. Xu, C. M. Woodside, and D. C. Petriu. Performance analysis of a software design using the UML profile for schedulability, performance and time. In P. Kemper and W. Sanders, editors, *Proc. of 13th International Conference on Modelling Techniques and Tools for Computer Performace Evaluation - Performance TOOLS'2003*, volume 2794 of *Lecture Notes in Computer Science*, pages 291–307. Springer, Sept. 2003.