UNIVERSITÀ CA' FOSCARI DI VENEZIA
Dipartimento di Informatica
Technical Report Series in Computer Science

Rapporto di Ricerca CS-2004-13

Dicembre 2004

S. Balsamo, M. Marzolla

Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models

Dipartimento di Informatica, Università Ca' Foscari di Venezia
Via Torino 155, 30172 Mestre–Venezia, Italy

Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models

Simonetta Balsamo   Moreno Marzolla
Dipartimento di Informatica
Universitá Ca' Foscari di Venezia
via Torino 155 30153 Mestre, ITALY
{balsamo,marzolla}@dsi.unive.it

ABSTRACT

Software performance based on performance models can be applied at early phases of the software development cycle to characterize the quantitative behavior of software systems. We propose an approach based on queueing networks models for performance prediction of software systems at the software architecture level, specified by UML. Starting from annotated UML Use Case, Activity and Deployment diagrams we derive a performance models based on multichain and multiclass Queueing Networks (QN). The UML model is annotated according to the UML Profile for Schedulability, Performance and Time Specification. The proposed algorithm translates the annotated UML specification into QN performance models, which can then be analyzed using standard solution techniques. Performance results are reported back at the software architecture level in the UML diagrams. As our approach can be fully automated and uses standard UML annotations, it can be integrated with other performance modeling approaches. Specifically, we discuss how this QN-based approach can be integrated with an existing simulation-based performance modeling tool.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling Methodologies*

General Terms

Performance, Design, Queueing Networks

Keywords

Software Performance, Queueing Networks, Performance Modeling

Submitted to WOSP'05

1. INTRODUCTION

Functional and non-functional validation of complex software systems is a crucial aspect of every software development process. In particular, non-functional requirements such as performance, safety, fault tolerance and reliability must often be carefully evaluated in order to guarantee that the system can be actually used. As developing large software systems is a difficult and costly process, it is very useful to be able to validate non-functional requirements as early as possible during the software development life cycle. Software performance based on performance models can be applied at early phases of the software development cycle to characterize the quantitative behavior of software systems.

In this paper we consider model-based performance evaluation and prediction of software architectures at the Software Architecture (SA) level, specified by Unified Modeling Language (UML). In particular, we propose an algorithm for translating annotated UML diagrams into a performance model based on multiclass Queuing Network (QN). Model-based software performance allows the software designer to receive feedback at the software architectural level. Such feedback can be used

1. to verify whether the proposed SA meets given non-functional performance requirements,
2. to compare different architectural implementations of the same system from a performance-oriented perspective.

The quality of the results from alternative (1) largely depends on the accuracy of the software specification with respect to the "real system", and on the quality of the parameters (estimated resource demands, workload intensities and so on) used to annotate the software model. Alternative (2) may be used to compare different systems, at a high level of abstraction, under the same environmental conditions (e.g., assuming the same workload intensities, or assuming the same set of available physical resources). In both cases the software modeler may perform "what if" experiments to check how performances are modifies by changing some model parameters.

As performance has been recognized as a particularly important issue of many software systems [19], many software performance evaluation approaches have been proposed in the literature, based on various types of software specification models and performance models [3]. Recent approaches consider UML [15] as the software system specification. UML is the de-facto standard for modeling software systems: it has a large user base and it is able to model static, dynamic and behavioral aspects of a system. Moreover, UML provides standard extension mechanisms based on additional constructs, which can be used to extend its semantic in a standard and consistent way. However, standard UML models cannot directly be used for quantitative evaluation purposes, as they lack quantitative information that are necessary for evaluating performance models. To fill this gap, the UML Profile for Schedulability, Performance and Time Specification (in short, SPT Profile) has recently been adopted as an OMG standard [16]. This profile allows the definition of requirements for performance and scheduling analysis, and the specification of quantitative information directly in the UML model.

In this paper we propose an approach for software performance modeling based on UML as the software description notation, and multichain and multiclass QN [5] as the performance model. The proposed approach takes advantage of the high level of abstraction of QN model, as we consider a simple and direct mapping between UML specifications and QN. This makes it easier the feedback interpretation of performance results obtained by the performance model solution at the original software specification model. Multiclass QN allows the representation of different classes of requests on the same system, and to model multiple actions requesting service from the same resource with different service demands. We consider software specifications described as UML Use Case, Deployment and Activity diagrams. Use Case diagrams describe the workloads applied to the system. Deployment diagrams describe the available hardware resources. Activity diagrams describe the computations the system performs in terms of service requests to the hardware resources.

Our approach provides some advantages. It makes use of the standard UML SPT profile for annotating the software model. In this way, the approach can be easily integrated into standard-compliant performance modeling tool, allowing the same software model to be analyzed with different performance models. This would be very useful as different performance models are best suited for special classes of architectural models; the advantages of multi-formalism software performance modeling has already been recognized [2]. The proposed approach leads to product-form QN models [5] if the software model satisfies some constraints which will be discussed later more precisely. Informally, such constraints are mostly on time distributions, allowed resource scheduling policies, lack of fork and join synchronization and simultaneous resource possession. In this case we can derive performance indices by efficient product-form solution algorithms for QN. Finally, we consider UML models describing both software and hardware components of the system to be analyzed. Taking into account information about hardware resources of the system allows more detailed performance models, and hence more precise results. If informations on hardware platforms are not available, it is still possible to use our approach to analyze the system at the SA level. To do so, it is sufficient to define enough identical "virtual" resources such that resource contention will not occur in the performance model. Performance results such as resources utilization and throughput will be meaningless in this case; other measures such as mean execution times of particular sequence of actions, or whole interactions described by an Activity diagram, can still be used.

The proposed approach can be integrated with other different methods for software performance analysis in a more general framework. The framework can provide a set of tools for quantitative analysis of software systems. Specifically, as observed in [2] software designers would take advantage of the combined use of different approaches to evaluate the performance of software artifacts. The proposed approach can be easily integrated with the recently developed tool UML-Ψ [13] providing a simulation-based evaluation of UML-based software systems.

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 SPT Profile, the QN performance model and the translation algorithm. The method implementation and application to a simple case study are described in Section 4. Finally, conclusions and open problems are discussed in Section 5.

2. SOFTWARE PERFORMANCE MODELING WITH QN

In this paper we propose an algorithm for automatic derivation of multiclass QN performance models. QN proved to be an effective modeling tool for different classes of systems, including communication networks and computer systems [10]. QN models have been the first performance models used for software evaluation purposes. The SPE (Software Performance Engineering) approach by William and Smith [19, 21, 20] has been the first integrated performance analysis and software development approach. It uses QN as the system model (representing hardware and software components), and Execution Graphs as the software execution model.

Different approaches dealing with performance evaluation of software systems have been proposed in the literature; see [3] for a comprehensive review and comparison of model-based performance prediction at the software development time. In the following, we will briefly review only some recent approaches that consider UML as the software specification notation.

In [11] the authors derive multichain QN models from annotated UML 2.0 specifications. They consider UML models in term of Use Case, Sequence and Collaboration diagrams. Their approach is compositional: the approach defines a library of architectural patterns (Sequence diagram fragments) and relates them to QN patterns. This is facilitated by the new operators for the Sequence diagrams introduced by UML 2.0 that allows identifying patterns of interest. The approach works at a high level of abstraction, as it does not consider the hardware resources on which the system operates. While this assumption can be useful if hardware information is not available, it is likely to produce inaccurate results if more detailed information is indeed available.

In [7] Cortellessa and Mirandola use information from different UML diagrams to incrementally generate a performance model representing the specified system. The technique, called PRIMA-UML, considers SA described in term of annotated Use Case, Sequence and Deployment diagrams. Use Case diagrams are used to derive the operational profile of the system. Sequence diagrams are used to derive an Execution Graph [20], and Deployment diagrams are used to derive the system execution model. The performance model is based on Extended QN, and is derived from the Execution Graph and the Deployment diagram. In [9] the PRIMA-UML methodology has been extended in order to cope with the case of mobile SA by enhancing its UML description to model the mobility-based paradigms. The approach generates the corresponding software and system execution models allowing the designer to evaluate the convenience of introducing logical mobility with respect to communication and computation costs. The authors define extensions of EG and EQN to model the uncertainty about the possible adoption of code mobility.

Some approaches deriving Layered Queueing Network (LQN) models from software specifications have been proposed in the literature [18, 17, 8]. LQN models are particularly useful for modeling client-server communication patterns in distributed systems. A LQN model is an acyclic graph, with nodes representing software entities and hardware devices. Arcs represent service requests. Nodes with outgoing but no incoming arcs represent clients, intermediate nodes with incoming and outgoing arcs are usually software servers and nodes with no outgoing arcs are hardware servers. While LQN are particularly useful for representing client-server systems, QN have some advantages: they are a more general modeling technique, and special classes of QN models (product-form QN) can be analyzed exactly using efficient algorithms. Our approach is able to generate product-form QN if some constraints on the UML software model are satisfied. General QN models can be solved either by means of approximate numerical algorithms or by simulation [6].

The CLISSPE approach by Menascé and Gomaa [14] derives multiclass QN models from software specifications described as UML Use Case, Class and Collaboration diagrams. As the approach was developed before the UML SPT profile was developed, annotations have to be specified in the CLISSPE language which is then translated into the QN model. Our approach in contrast does not require the use of intermediate hand-coded representations, as the QN model is directly derived from annotated UML diagrams.

In this paper we propose a different approach for software performance modelling. We consider a SA described both at the software execution level and at the hardware level. In this way, the software engineer may be able to model the system by considering more information, hopefully getting better results. Our algorithm does not use any intermediate representation, thus it is able to translate UML software specifications directly into performance models. This leads to the advantage that the derived performance model is structurally very similar to the software model. Such similarity makes the process of interpreting performance results back at the SA design level very easy. In particular, each service center in the QN model represents a processing resources described as a node in the UML Deployment diagram. Thus, performance measures computed for a service center of the QN (e.g., utilization and throughput) directly corresponds to the same performance measure on the hardware resource it represents. Moreover, our approach is consistent with the Activity-based approach described in [16]. Thus, the proposed algorithm can be easily included in possible performance evaluation framework based on the UML SPT Profile, such as the UML-Ψ tool described in [13].

3. FROM UML MODELS TO QN PERFORMANCE MODELS

In this section we describe our approach for transforming annotated UML specifications into multiclass QN models. We consider a software performance modeling framework shown in Fig. 1. Starting from software performance requirements and an UML specification we consider an annotation step to enrich the UML model with performance information. Then we derive the performance model whose solution can provide a feedback at the level of the UML software specification model.

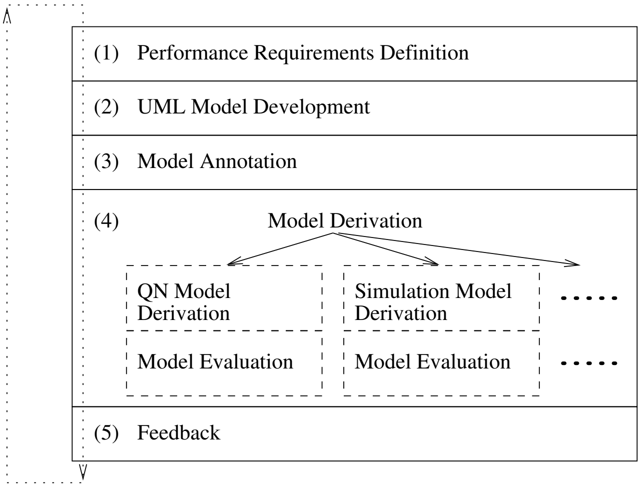


Figure 1: Software Performance Modeling Framework

More specifically, the four steps work as follows:

1. The software engineer defines the performance requirements of the software system.

2. A UML specification of the software system is developed. We consider software specifications in term of Use Case, Activity and Deployment diagrams. The software specification includes a description of both the hardware resources (Deployment diagram), workloads (Use Case diagram) and software execution (Activity diagram).

3. The UML model is annotated with quantitative, performance-oriented information. We use a subset of

the annotations defined in the UML Profile for Schedulability, Performance and Time Specification (in short, SPT Profile).

4. The annotated software model is translated into a suitable performance model, which is then evaluated. It is possible to derive multiple performance models from the same UML specification. We currently consider the proposed QN-based performance model, described in this paper and the simulation-based performance model described in [4].

5. Performance evaluation results provide feedback that should be interpreted at the SA design level. The software development process may be iterated from step (1) if the performance results show that the requirements are not satisfied.

The part of this framework dealing with simulation model derivation and execution have been implemented in the UML-Ψ tool [13]. In this paper we consider step (3) above and we define an algorithm for translating the annotated software model into a multiclass QN based performance model. The performance model can be solved using an appropriate solution technique and performance results provide feedback at the software specification level. In particular, feedback is provided by inserting performance results as annotations to the software model elements they refer to. Hence the software performance analysis cycle can be iterated to meet given performance requirements or to compare different software design alternatives.

We first show how the software model can be described and annotated using UML Use Case, Activity and Deployment diagrams. Then we introduce the multiclass QN performance model. Finally we give the transformation algorithm for translating the software model into the performance model. We illustrate with an example how the UML model annotation and the performance model derived.

3.1 UML model annotations

The software architecture is described in term of UML Use Case, Activity and Deployment diagrams. The diagrams are annotated according to the UML SPT Profile [16], as described below.

Use Case diagrams represent workloads applied to the system: each Actor models either an open or closed population of requests arriving to the system. Actors stereotyped as ≪PAopenLoad≫ represent an infinite stream of external requests. The interarrival time between two subsequent requests can be specified by using the `PAoccurrence` tagged value. We usually assume exponential interarrival times distribution in order to derive a product-form QN model. Actors stereotyped as ≪PAclosedLoad≫ represent a fixed population of requests that repeatedly interact with the system. The number of requests is given by the `PApopulation` tag, and the time spent by each completed request before the next interaction with the system is given by the `PAextDelay` tag.

In the following, we consider a simple case study to illustrate the annotations. We consider a network-based video server system, where a user requests video frames from a video server through a network connection.

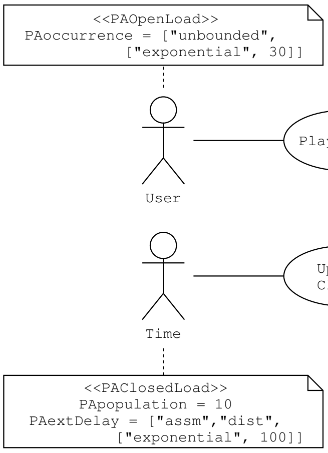


Figure 2: Example of Annotated Use Case diagram

Figure 2 represents an annotated Use Case diagram. We have two actors, `User` and `Time` respectively. The first actor represents an open population of requests with exponentially distributed interarrival time with mean 8 time units. The second actor represents a fixed population of 10 requests which continuously circulate through the system; each request spends an exponentially distributed amount of time outside the system, with mean 10 time units, before interacting again. In our example, actor `User` represents the requests generated by users requesting videos from the video server. The `Time` actor represents periodic updates which are sent by the video server to the player software on the client workstation.

Activity diagrams are used to describe in more detail the content of each Use Case. In particular, Activity diagrams describe the computation performed on the system. Thus each Use Cases must have exactly one associated Activity diagram. Each action state stereotyped as ≪PAstep≫ in the Activity diagrams represents a service request from one active resource (processor) of the system. The resource name is indicated with the `PAhost` tag, and the service demand is given with the `PAdemand` tag. In order to produce product-form QN models, service demands should be exponentially distributed. Transitions in the Activity diagram can be annotated with an associated probability `PAprob`, which represents the probability of taking one specific transition. This is only meaningful if there are multiple outgoing transitions from the same action state; in this case, the sum of probabilities of all outgoing transitions must be 1. Note that UML Activity diagrams may contain fork and join synchronization nodes. Fork nodes allow the execution flow to split into multiple concurrent threads; join nodes allow multiple independent execution threads to synchronize. Fork and join synchronization can be represented into the QN model. However, the resulting network would not be in product-form, and could only be solved by approximate algorithms or by simulation [6].

Fig. 3 shows the annotated Activity diagrams associated to the Use Cases of Fig. 2. The Activity diagram associated with the `Play Video` Use Case shows the computations performed in the system when a user requests a video. First, the video server is initialized; then, a sequence of actions is iterated several times to transmit the frames. Each frame is encoded by the server, transmitted through the network, displayed on the client machine and finally an acknowledge is sent back through the network. When the loop is finished, the server process is terminated.

The activity diagram associated with the `Update Client` Use Case shows the interactions performed in the system when a software update is sent from the video server to the client machine through the network.

Deployment diagrams are used to describe the hardware resources available on the system. Each node stereotyped as ≪PAhost≫ represents an active resource (processor). The scheduling policy can be specified with the `PAschedPolicy` tag: we consider the tag values for "FIFO", "LIFO" and "PS" (Processor Sharing) scheduling policies. The `PArate` tag allows the modeler to specify the relative speed of the processor (for example if `PArate`=2 the processor will spend 0.5 time units to satisfy a service request for 1 time unit).

Note that the UML SPT Profile allows the representation of active resources as well as passive ones. However, QN with simultaneous, passive resource possession do not allow a product-form solution, so in the following we will consider active resources (processors) only.

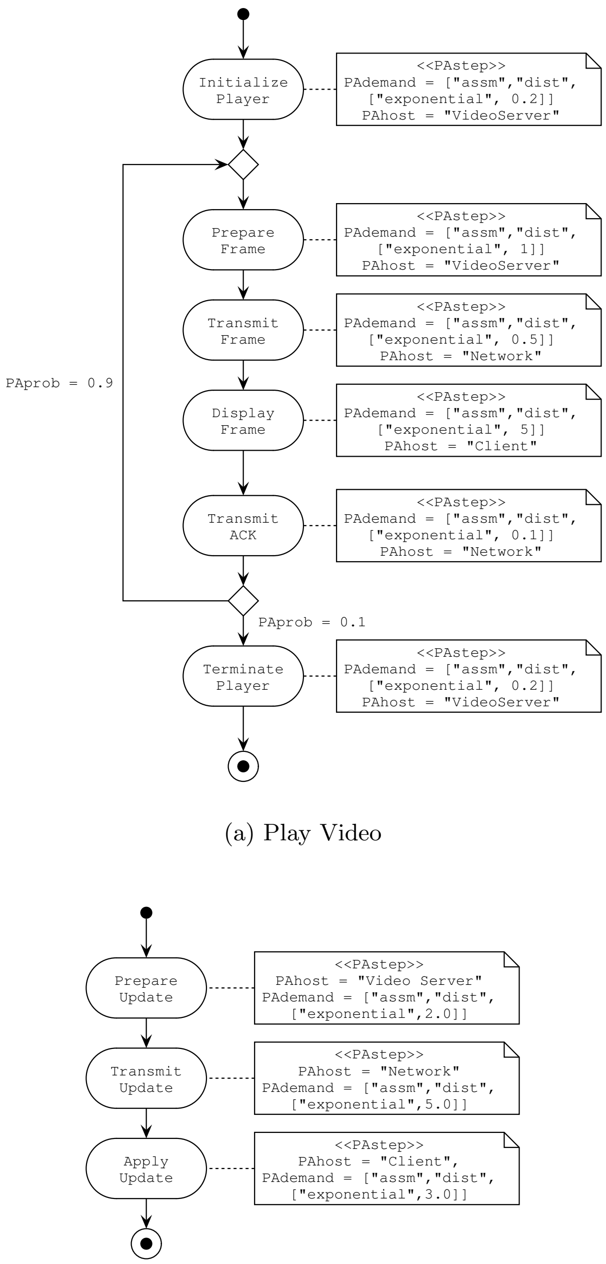
Fig. 4 shows an example annotated Deployment diagram. We see three resources available in the system: the video server, the client machine and the network. All of them represent processors with FIFO scheduling policy, and different service rates shown in the annotations.

3.2 QN model

We consider QN models with K service centers and C different classes of customers. The service centers represent the resources of the UML model. If there are closed workloads applied on the system, then we define one additional delay center (infinite server) for each closed workload. Performance measures can be directly interpreted at the SA design level: utilization, throughput and mean queue length of the service centers denote the utilization, throughput and mean number of waiting requests respectively on the physical resources on which the SA executes.

Multiple actors represent multiple workloads. In this case, the QN model can be decomposed into multiple disjoint chains. Each chain represents the requests from the same workload. Requests belonging to the same chain (workload) can be of different classes: job classes are used to model the path of requests to the various resources in the UML model that correspond to the path of job visits to the servers in the QN model.

The QN model is described by a set of parameters, which are summarized in Table 1. The QN consists of K servers and C different classes of requests. The routing probability matrix is $\mathbf{P} = [P[i, r, j, s]]$, $1 \leq i, j \leq K$, $1 \leq r, s \leq C$, where $P[i, r, j, s]$ denotes the probability that a class r job finishing its service at service center i immediately join service center j as a class s job. Let $\mu[i, r]$ denote the service rate for class r customers at service center i. For an open chain, $\lambda[r]$ denotes the external arrival rate of class r customers and $q[i, r]$ is the probability of an arrival of class r requests at service center i . For a closed chain, N denotes the constant number of requests.



(a) Play Video

(b) Update Client

Figure 3: Annotated Activity diagram for the `Play Video` and `Update Client` Use Cases

3.3 Transformation algorithm

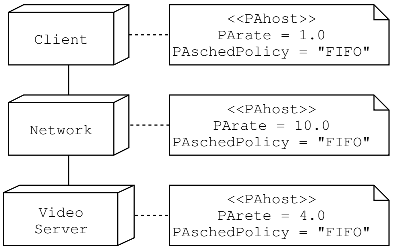


Figure 4: Example of Annotated Deployment diagram

C	Number of customer classes
K	Number of service centers
$\mathbf{P} = P[i, r, j, s]$	Probability that a customer of class r completing its service at service center i by enters service center j as a class s customer
$\mu = \mu[i, r]$	Service rate of class r customers at service center i
$\lambda = \lambda[r]$	(Open QN) Arrival rate of class r customers
$\mathbf{q} = q[i, r]$	(Open QN) Probability that an arriving class r customer enters the network at service center i
N	(Closed QN) Number of customers in the system

Table 1: Description of symbols used in the algorithms

We shall now present the transformation algorithm from the annotated UML software model into the QN performance model.

We consider a SA described as a set of annotated UML Use Case, Activity and Deployment diagrams. Use Case diagrams are translated into workloads of the QN model, Activity diagrams define the QN model routing matrix, and Deployment diagrams describe the service centers.

Let us introduce the notation used for defining the transformation algorithm. In the following, given an object x we denote by $a[x]$ the value of attribute a for object x. We consider an UML Activity diagram as a directed graph $G = (A, T)$ with nonnegative edge weights, where $A = \{a_0, a_1, a_2, \ldots\}$ is the finite set of action states of the Activity diagram and $T = \{t_1, t_2, \ldots\}$ the finite set of transitions. We denote with a_0 the entry and exit point of the Activity diagram. A generic transition t is a pair of activities (a_i, a_j), $1 \leq i, j \leq |A|$. We define the attribute $p[t] \in [0, 1]$ for each transition t which denotes the probability of traversing t. A transition t from action a to action b means that action b is executed after a is completed. To model different possible execution paths, it is possible to specify multiple successors of an action in the Activity diagrams. Each transition in the UML Activity diagram is annotated with the `PAprob` tagged value which specifies the transition probability.

The UML model contains a set $R = \{r_1, r_2, \ldots r_K\}$ of resources. We define an attribute for each action of the Activity diagram, $a \in A$, denoted by $\mathit{res}[a]$, that represents the resource on which the action a is executed. The association between actions and resources derives from the `PAhost` tag of UML action a. We assume that all resources in R get a unique identifier in the range $[1 \ldots K]$. We introduce the attribute $\mathit{id}[\cdot]$ for each resource $r_i \in R$ to denote such identifier, defined as $\mathit{id}[r_i] = i$.

Table 2 summarizes the introduced attributes used in the algorithm and the corresponding UML tag name used derive the attribute values.

We describe now the transformation algorithm from the annotated UML model to the QN, first for an open and a closed workloads, respectively, and finally for the entire model.

Open Workloads

Algorithm 1 describes the transformation for generating open QN models. The inputs of the algorithm are an actor W stereotyped as ≪PAopenLoad≫, the activity diagram A describing the associated Use Case, and the Deployment diagram R describing the set of all the resources available in the system.

In general the software modeler may specify multiple actions requesting service from the same active resource with different service demands. Thus, it is necessary to ensure that every execution path in the Activity diagram corresponds to the correct sequence of visits to the service centers in the QN model. To this aim we proceed as follows. We consider the set of actions that require service to each resource in the UML model in order to define the set of classes that belong to the corresponding service center in the QN that represent the resource. For each resource $r \in R$ we define the attribute $\mathit{count}[r]$ that denotes the total number of actions requesting service from r. For each resource r, we label all the actions in the set $\{a \in A \mid \mathit{res}[a] = r\}$ with a unique number in the range $[1 \ldots \mathit{count}[r]]$. Then for each action $a \in A$ we introduce the attribute $\mathit{index}[a]$ that denotes this unique number. Hence action a requesting service to resource $\mathit{res}[a]$ is translated into a job of class $\mathit{index}[a]$ requesting service to the service center representing $\mathit{res}[a]$. This attribute $\mathit{index}[a]$ is used to define the transition matrix $\mathbf{P}$ of the QN.

To prove the correctness of Algorithm 1 we observe that the routing matrix of the QN model, defined in lines (16)–(22), is such that if there is a transition $t = (x, y)$ from action x to action y with probability $p[t]$ in the UML model, then there is a QN transition from service center $\mathit{id}[\mathit{res}[x]]$ to $\mathit{id}[\mathit{res}[y]]$ with the same probability. The normalizing assumption on the probabilities `PAprob` associated to the multiple outgoing transitions from the same action state in the Activity diagrams given in section 3.1 guarantees the normalizing probability condition of matrix $\mathbf{P}$ in the QN.

External arrivals are defined on the service center associated to the initial action of the Activity diagram. The arrival rate is set equal to the `PAoccurrence` tag associated to the Actor executing the Activity diagram. Job classes are used to route requests according to the action order defined in the UML Activity diagram. Transition probabilities are derived from the `PAprob` tags associated to UML transitions.

Attribute	Description	UML Tagged Value
$res[a]$	Resource on which action a executes	`PAhost`
$demand[a]$	Service demand of action a	`PAdemand`
$p[t]$	Probability of traversing transition t	`PAprob`
$rate[r]$	Service rate of resource r	`PArate`
$arrivalrate[W]$	Arrival rate of external requests (open workload)	`PAoccurrence`
$population[W]$	Number of requests (closed workload)	`PApopulation`
$extdelay[W]$	External delay of requests (closed workload)	`PAextDelay`

Table 2: Object attributes used in the algorithms and corresponding UML tag name defining their value

Algorithm 1 Generation of an open QN

```
Require: Activity diagram $G = (A, T)$, Deployment diagram $R$, Actor $W$ stereotyped as ≪PAopenLoad≫
{Initializations}
for all $r \in R$ do
    $count[r] := 0$
end for
for all $a \in A$ do
    $count[res[a]] := count[res[a]] + 1$
    $index[a] := count[res[a]]$
end for
$C := \max_{r \in R}\{count[r]\}$                {Number of Classes}
$K := |R|$                        {Number of Service Centers}
$\mathbf{P} = P[i, r, j, s] :=$ new vector $[K \times C \times K \times C]$
$\lambda = \lambda[r] :=$ new vector $[C]$
$\mathbf{q} = q[i, r] :=$ new vector $[K \times C]$
$\mu = \mu[i, r] :=$ new vector $[K \times C]$
{Define the routing matrix $\mathbf{P}$}
for all $t = (x, y) \in T$ such that $x \neq a_0$ and $y \neq a_0$ do
    $i := id[res[x]]$
    $r := index[x]$
    $j := id[res[y]]$
    $s := index[y]$
    $P[i, r, j, s] := p[t]$
end for
{Define Service Rates $\mu$}
for all $a \in A - \{a_0\}$ do
    $i := id[res[x]]$
    $r := index[a]$
    $\mu[i, r] := rate[r]/demand[a]$
end for
{Define Arrival Rate $\lambda$}
for all $t = (x, y)$ such that $x = a_0$ do
    $i := id[res[y]]$
    $r := index[y]$
    $\lambda[r] := arrivalrate[W]$
    $q[i, r] := p[t]$
end for
```

Closed Workloads

Algorithm 2 describes the derivation of a closed QN from an actor W stereotyped as ≪PAclosedLoad≫. The approach is very similar to the one used for open QN generation. Note however that in this case the number of service centers is $K+1$, where $K = |R|$ is the number of nodes in the Deployment diagram. The reason is that the constant number of requests may spend some time between the end of one interaction with the system and the beginning of the next one. Such time, also called "think time", can be defined by the software modeler by using the `PAextDelay` tag. The think time is modeled as a service center with infinite number of servers. The service rate of this service center is equal to $1/extdelay[W]$ for all job classes, W being the Actor representing the closed workload. The population N of requests is derived from the `PApopulation` tag associated to the Actor.

Algorithm 2 Generation of a closed QN

```
Require: Activity diagram $G = (A, T)$, Deployment diagram $R$, Actor $W$ stereotyped as ≪PAclosedLoad≫
{Initializations}
for all $r \in R$ do
    $count[r] := 0$
end for
for all $a \in A$ do
    $count[res[a]] := count[res[a]] + 1$
    $index[a] := count[res[a]]$
end for
$C := \max_{r \in R}\{count[r]\}$                {Number of Classes}
$K := |R| + 1$                    {Number of Service Centers}
$\mathbf{P} = P[i, r, j, s] :=$ new vector $[K \times C \times K \times C]$
$\lambda = \lambda[i, r] :=$ new vector $[K \times C]$
$\mu = \mu[i, r] :=$ new vector $[K \times C]$
{Define the routing matrix $\mathbf{P}$}
for all $t = (x, y) \in T$ do
    $i := id[res[x]]$
    $r := index[x]$
    $j := id[res[y]]$
    $s := index[y]$
    $P[i, r, j, s] := p[t]$
end for
{Define Service Rates $\mu$}
for all $a \in A - \{a_0\}$ do
    $i := id[res[x]]$
    $r := index[a]$
    $\mu[i, r] := rate[r]/demand[a]$
end for
Let service center 0 be a Delay Center with $\mu[0, \cdot] := 1/extdelay[W]$ for all job classes
$N := population[W]$
```

The Complete Algorithm

Algorithms 1 and 2 produce the QN for single workloads. Each QN consists of a single chain, which is an open or closed depending on the workload it represents. Since it possible to define multiple workloads to be applied to the software system, the complete QN model can be obtained by merging all the individual networks. The complete performance model will thus be made of multiple chains, each one with multiple job classes. Algorithm 3 summarizes the complete UML-to-QN translation. Let *Ch* denote the set of chains of the QN.

Algorithm 3 UML to QN model transformation

```
Let $Ch := \emptyset$    {The set of chains}
for all Workload $W$ do
    if $W$ is stereotyped as $\ll$OpenWorkload$\gg$ then
        Let $Q$ be the QN computed according to Algorithm 1
        $Ch := Ch \cup Q$
    else if $W$ is stereotyped as $\ll$ClosedWorkload$\gg$ then
        Let $Q$ be the QN computed according to Algorithm 2
        $Ch := Ch \cup Q$
    end if
end for
Let $Q$ be the QN obtained by merging all chains in $Ch$
Return $Q$
```

If the UML model is made of K_O actors representing open workloads, K_C actors representing closed workloads, N_R resources, then the QN model for the whole system has $K_O + K_C$ chains, including a total of k job classes, and $N_R + K_C$ service centers. Note that k is defined as the maximum number of actions requesting service from one of the resources, considering all the Activity diagrams.

The computational time complexity of the transformation algorithm is $O(|A| + |T|)$, where $|A|$ and $|T|$ are the total number of actions and transitions in all the Activity diagrams. The space required is dominated by that required for storing the routing matrix, which is $O((N_R + K_C)^2 k^2)$.

It should be observed that in order to produce a product-form QN model the UML model must satisfy some constraints. Specifically, the following conditions must hold [5]:

1. Scheduling policies at the active resources must be one of FCFS (First-Come-First-Served), PS (Processor Sharing), LCFS-PR (Last-Come-First-Served with Preemptive Resume) and IS (Infinite Server).

2. Service demands from FCFS servers must have the same distribution. Service demand for PS, LCFS-PR and IS service centers can be class-dependent (i.e., multiple actions can have different service demands).

3. Arrival processes in open workloads must be Poisson. Multiple independent streams of requests may be present, with (possibly different) arrival rates λ_i.

4. No passive resources may be present in the system.

5. Activity diagrams may not include fork and join nodes.

Note that if the previous conditions are not satisfied, a QN model can still be derived from the UML model. However, the QN model does not admit a product-form solution, and its analysis requires approximate solution techniques or simulation.

4. EXAMPLE

As an example, we consider the UML diagrams shown in Fig. 2–4. From the Use Case diagram of Fig. 2 we see two actors, `User` and `Time`, representing an open and closed workload, respectively.

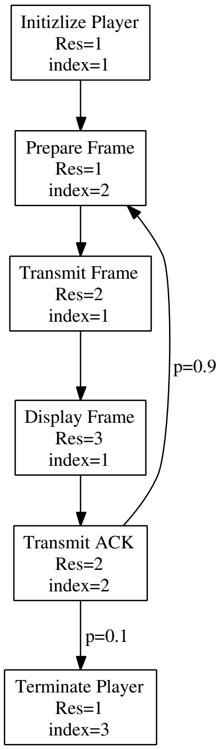


Figure 5: Annotated graph produced by applying Algorithm 1 to the "Play Video" Activity diagram

The `User` workload and its associated Activity diagram is processed according to Algorithm 1. We define three service centers that correspond to the three nodes in the Deployment diagram. The Activity diagram associated to the `User` workload has three actions (namely, `Initialize Player`, `Prepare Frame`, `Terminate Player`) that are executed on the same physical resource, the Video Server.

Let us suppose that the numbering of resources is as follows: resource 1 corresponds to the `Video Server` node, resource 2 corresponds to the `Network` node, resource 3 corresponds to the `Client Workstation` node. Algorithm 1 annotates the Activity diagram as in Fig. 5. Actions requesting service from the same processor get increasing values of their `index` attribute.

Given the intermediate annotations, the transformation algorithm produces the QN in Fig. 6 which has three customer classes. The reason is that there are three actions requesting service from the `Video Server` resource. Customer classes are shown in the figure using different line dashes. Class switching is denoted by lines entering a service center and leaving it with a different pattern. The service rates are defined according to the `PArate` tagged values on the UML diagrams. Service rates are $\mu_1 = (20, 4, 20)$, $\mu_2 = (20, 100, 1)$, $\mu_3 = (0.2, 1, 1)$. Arrival rate $\lambda = 1/30$ at service center 1, $p = 0.1$

The `Time` workload and its associated Activity diagram is

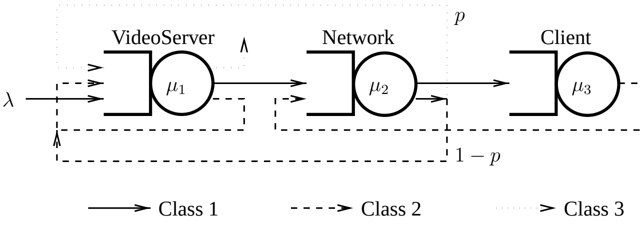


Figure 6: QN model for the `Show Video` Activitydiagram.

translated according to Algorithm 2 into the closed network shown in Fig. 7. In this case we have the three service centers representing the physical resources in the Deployment diagram, plus an additional delay center representing the external delay (think time) spent by requests outside the system. In this case the QN has just one job class, as each resource is used by at most one action. Service rates are $\mu_0 = 0.01$, $\mu_1 = 2$, $\mu_2 = 2$, $\mu_3 = 1/3$

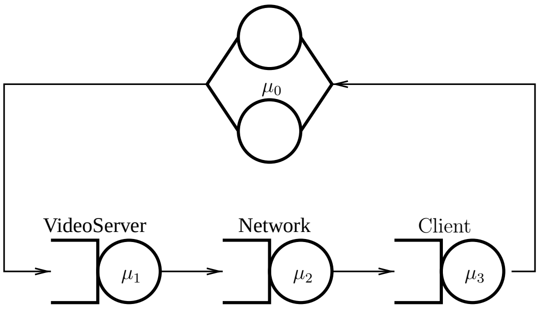


Figure 7: QN model for the `Update Client` Activity diagram.

Finally, the complete multiclass QN model corresponding to the whole system is shown in Fig. 8. The QN has four job classes, three service centers and one delay center. The four job classes belong to two different chains. The first chain corresponds to the interactions described by the open workload; the second chain corresponds to the interactions described by the closed workload.

The QN model of Fig. 8 can now be analyzed with an appropriate algorithm, and performance results can be computed. These performance measures include server utilization and throughput, which can be directly reported and interpreted at the SA design level: they correspond to utilization and throughput of the resources represented in the Deployment diagram. The software modeler may use these values to identify potential bottlenecks, or compare different architectural implementations of the same system from a performance-oriented viewpoint. The SA model can eventually be modified and the performance modeling approach iterated many times until the non-functional performance requirements are satisfied.

5. CONCLUSIONS

In this paper we described an algorithm for automatic translation of annotated UML specifications into multiclass QN performance models. We consider UML specifications in term of Use Case, Activity and Deployment diagrams. Use Case diagrams are used to describe external arrivals of requests. Activity diagrams specify the system execution model, i.e., which actions are performed and where those actions execute. Finally, Deployment diagrams describe the hardware resources available in the system. All diagrams must be annotated with quantitative, performance-oriented annotations according to the UML SPT Profile. The resulting multiclass QN model can be analyzed with standard techniques. The proposed approach can be fully automated and uses standard UML annotations; for this reason, it can be integrated with other software performance modeling approaches based on the UML SPT Profile.

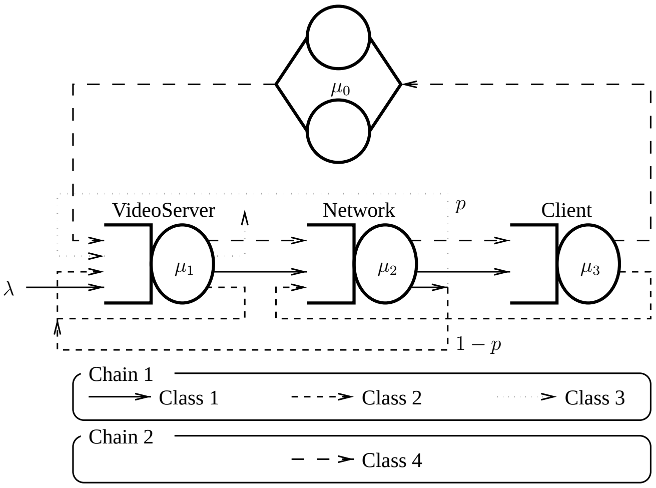


Figure 8: The full QN model

It should be observed that, while the approach we described can produce QN performance models from general UML specifications, the software model must be constrained in order for the performance model to be solved analytically. In particular, Activity diagrams with fork and join nodes, or with simultaneous resource possession (actions requesting and releasing passive resources) would result in Extended QN performance models which can be analyzed with approximate numerical techniques or by simulation. In the latter case, other performance modeling approaches may be applied (e.g., those based on LQN [1, 18, 17], or simulation-based models [13, 12]). The choice of the performance model which is best suited for a particular type of system and for the kind of analysis to be performed is outside the scope of this paper, and is discussed in detail in [2]

Future works include the integration of this approach, and other software performance modeling approaches based on the UML SPT profile, on a general framework. The aim is allowing the software designer to choose the performance evaluation tool best suited for the kind of analysis to be performed. Also, with the development of additional UML profiles, other non-functional aspects of SA could similarly be evaluated at the design level.

6. REFERENCES

[1] C. M. W. anb C. Hrischuk, B. Selic, and S. Brayarov. Automated performance modeling of software generated by a design environment. *Performance*

Evaluation, 45:107–123, 2001.

[2] S. Balsamo, A. D. Marco, P. Inverardi, and M. Marzolla. Experimenting different software architectures performance techniques: A case study. In *Proc. WOSP 2004*, pages 115–119, Redwood Shores, California, Jan. 14–16 2004. ACM Press.

[3] 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 2004.

[4] S. Balsamo and M. Marzolla. Simulation modeling of UML software architectures. In D. Al-Dabass, editor, *Proc. of ESM'03, the 17th European Simulation Multiconference*, pages 562–567, Nottingham, UK, June 9–1 2003. SCS–European Publishing House.

[5] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, Apr. 1975.

[6] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation With Computer Science Applications*. Wiley–Interscience, 1998.

[7] V. Cortellessa and R. Mirandola. Prima-uml: a performance validation incremental methodology on early uml diagrams. *Sci. Comput. Program.*, 44(1):101–129, 2002.

[8] D. G. Gu and C. Petriu. XSLT transformation from UML models to LQN performance models. In *Proc. WOSP 2002*, pages 227–234, Rome, Italy, July 2002. ACM Press.

[9] V. Grassi and R. Mirandola. Primamob-uml: a methodology for performance analysis of mobile software architectures. In *Proceedings of the third international workshop on Software and performance*, pages 262–274. ACM Press, 2002.

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

[11] A. D. Marco and P. Inverardi. Compositional generation of software architecture performance QN models. In *Proc. Fourth Working IEEE/IFIP Conf. on Software Architecture (WICSA04)*, Oslo, Norway, June 2004.

[12] 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.

[13] M. Marzolla and S. Balsamo. UML-PSI: The UML Performance SImulator. In *Proc. of the First International Conference on the Quantitative Evaluation of Systems (QEST 2004)*, pages 340–341, Enschede, The Netherlands, Sept. 27–30 2004. IEEE Computer Society.

[14] D. A. Menascé and H. Gomaa. A method for design and performance modeling of client/server systems. *IEEE Trans. Software Engineering*, 26(11):1066–1085, 2000.

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

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

[17] D. C. Petriu and H. Shen. Applying UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In *Proceedings of the 7th International Conference on Modelling Techniques and Tools for Performance Evaluation*, pages 159–177. Springer LNCS 794, 2002.

[18] D. C. Petriu and X. Wang. From UML descriptions of high-level software architectures to LQN performance models. In *Proceedings of AGTIVE 99*, pages 47–62. Springer Verlag LNCS 1779, 1999.

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

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

[21] L. G. Williams and C. U. Smith. Performance evaluation of software architectures. In *Proc. of the First International Workshop on Software and Performance (WOSP'98)*, pages 164–177, Santa Fe, New Mexico, 1998. ACM Press.