Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models[*]

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

[*]This work has been partially supported by MIUR Research Project FIRB "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools".

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.

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 multiclass Queuing Network (QN) performance model. Our approach can be used to validate non-functional, performance-related requirements: evaluation of the performance model computes the utilization and throughput of the resources (processors) available in the system, and the mean execution time of actions in UML Activity diagram. Hence, the software designer can check whether the expected value of these parameters are within given thresholds in order to validate the software design.

As performance has been recognized as a particularly important issue of software systems [13], many software performance evaluation approaches have been proposed in the literature [2]. Recent approaches consider UML [10] as the software system specification. UML is the de-facto standard for modeling software systems; moreover, it provides standard extension mechanisms based on additional constructs, which can be used to extend its semantic in a standard and consistent way. The UML Profile for Schedulability, Performance and Time Specification (in short, SPT Profile) [9] allows the specification of quantitative information directly in the UML model.

We propose an approach for software performance modeling based on UML as the software description notation, and multichain and multiclass QN [3] as the performance model. The 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 interpretation at the original software specification level of performance results obtained by the performance model solution. Note that our approach differs from similar ones (e.g., CLISSPE [8] because we do not use intermediate hand-coded representations, as the QN model is directly derived from annotated UML diagrams. We consider UML 1.x Use Case, Activity and Deployment diagrams as proposed in the *Activity-based approach* of [9].

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. The proposed approach leads to product-form QN models [3], which can be efficiently analyzed, if the software model satisfies some constraints (discussed later more precisely). 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 information on hardware platforms is not available, 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. Our proposal 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 [1] 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 [7] providing a simulation-based evaluation of UML-based software systems.

The paper is organized as follows. In Section 2 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 3. Finally, conclusions and open problems are discussed in Section 4.

2. FROM UML TO QN MODELS

In the following we propose an algorithm for automatic derivation of multiclass QN performance models from annotated UML specifications. QN proved to be an effective modeling tool for different classes of systems, including communication networks and computer systems [5].

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.

2.1 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 [9], as described in the following.

Use Case diagrams represent workloads applied to the system. Actors stereotyped as ≪PAopenLoad≫ represent an infinite stream of external requests. The interarrival time between two subsequent requests is specified using the `PAoccurrence` tagged value. We 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.

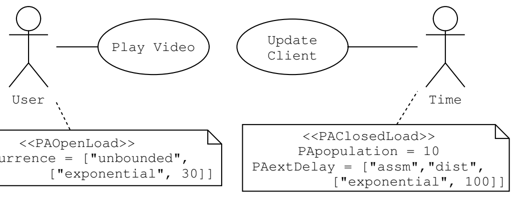
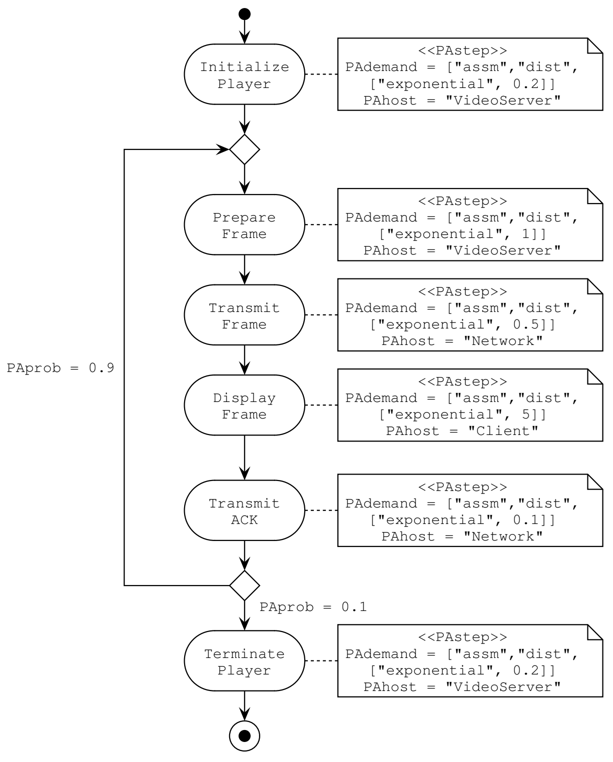

Play Video
Update Client
User
Time
<<PAOpenLoad>>
PAoccurrence = ["unbounded", ["exponential", 30]]
<<PAClosedLoad>>
PApopulation = 10
PAextDelay = ["assm","dist", ["exponential", 100]]

Figure 1: Annotated Use Case diagram

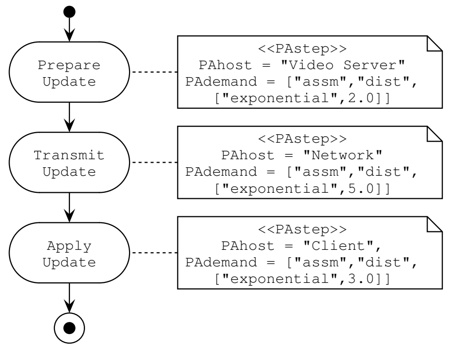
In the following, we use a simple case study to illustrate the annotations. We consider a network-based video server system, where a user requests video frames from a video server through a network connection. Figure 1 represents an annotated Use Case diagram. There are two actors, `User` and `Time` respectively. The first actor (`User`) represents an open population of requests with exponentially distributed interarrival time with mean of 8 time units. The second actor (`Time`) 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. Actor `User` represents the requests generated by users requesting videos from the video server. Actor `Time` 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, they describe the computation performed on the system. Each Use Cases must have exactly one associated Activity diagram. Each action state stereotyped as ≪PAstep≫ represents a service request from one active resource (processor) of the system. The resource name and service demand are indicated with the `PAhost` and `PAdemand` tags, respectively. In order to produce product-form QN models, service demands should be exponentially distributed. Transitions in the Activity diagram are annotated with a 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 and join synchronization can be represented into Extended QN models. However, the resulting network would not be in product-form, and could only be solved by approximate algorithms or by simulation [4, 5]. Fig. 2 shows the annotated Activity diagrams associated to the Use Cases of Fig. 1. The Activity diagram associated with the `Play Video` Use Case shows the computations performed in the system when a user requests a video. The Activity diagram associated with the `Update Client` Use Case shows the interactions executed when a software update is sent from the video server to the client machine through the network.

Deployment diagrams are used to describe the available



(a) Play Video



(b) Update Client

Figure 2: Annotated Activity diagram for the `Play Video` and `Update Client` Use Cases

hardware resources. 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. Fig. 3 shows an example annotated Deployment diagram with three resources: a video server, a client machine and the network.

2.2 QN model

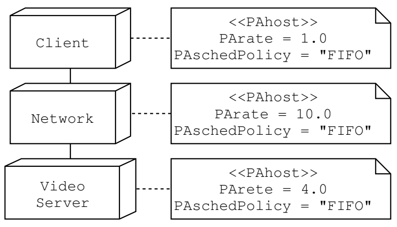


Figure 3: Annotated Deployment diagram

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. If there are multiple actors, representing multiple workloads, 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 corresponds to the path of job visits to the servers in the QN model. Table 1 summarizes the set of parameters that describe the QN model.

C	Number of customer classes
K	Number of service centers
$\mathbf{P} = P[i, r, j, s]$	Probability that a class r customer completing service at serv. center i enters serv. center j as a class s customer ($1 \leq i, j \leq K$, $1 \leq r, s \leq C$)
$\mu = \mu[i, r]$	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: Notation used in the algorithms

2.3 Transformation algorithm

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.

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. Let 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 we can 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 $R = \{r_1, r_2, \dots r_K\}$ of resources. We define an attribute for each action of the Activity diagram, $a \in A$, denoted by $res[a]$, 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 $[1 \dots K]$. We introduce the attribute $id[\cdot]$ for each resource $r_i \in R$ to denote such identifier, defined as $id[r_i] = i$. 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 open and a closed workloads, respectively, and then 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 the available resources.

It is possible to specify multiple actions requesting service from the same active resource with different service demands. Thus, we need 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 $count[r]$ that denotes the total number of actions requesting service from r. For each resource r, we label all the actions in the set $\{a \in A \mid res[a] = r\}$ with a unique number in the range $[1 \dots count[r]]$. Then for each action $a \in A$ we introduce the attribute $index[a]$ that denotes this unique number. Hence action a requesting service to resource $res[a]$ is translated into a job of class $index[a]$ requesting service to the service center representing $res[a]$. This attribute $index[a]$ is used to define the transition matrix $\mathbf{P}$ of 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.

Closed Workloads. Algorithm 2 describes the derivation of a closed QN from an actor W stereotyped as ≪PAclosedLoad≫. The approach is similar to the one used for open QN; however, note that the generated QN has $K + 1$ service centers, 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

Algorithm 1 Generation of an open QN

```
Require: Activity diagram $G = (A, T)$, Deployment diagram $R$, Actor $W$ stereotyped as ≪PAopenLoad≫
  Let $count[r] := 0$, for each $r \in R$
  for all $a \in A$ do
    $count[res[a]] := count[res[a]] + 1$
    $index[a] := count[res[a]]$
  $C := \max_{r \in R}\{count[r]\}$                {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
    $i := id[res[x]]$
    $r := index[x]$
    $j := id[res[y]]$
    $s := index[y]$
    $P[i, r, j, s] := p[t]$
  {Define Service Rates $\mu$}
  for all $a \in A - \{a_0\}$ do
    $i := id[res[x]]$
    $r := index[a]$
    $\mu[i, r] := rate[r]/demand[a]$
  {Define Arrival Rate $\lambda$}
  for all $t = (x, y)$ such that $x = a_0$ do
    $i := id[res[y]]$
    $r := index[y]$
    $\lambda[r] := arrivalrate[W]$
    $q[i, r] := p[t]$
```

"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.

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, 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 according to the following steps:

```
  $Ch := \emptyset$                          {The set of chains}
  for all Workload $W$ do
    if $W$ is stereotyped as ≪OpenWorkload≫ then
      Let $Q$ be the QN computed according to Alg. 1
    else if $W$ is stereotyped as ≪ClosedWorkload≫ then
      Let $Q$ be the QN computed according to Alg. 2
    $Ch := Ch \cup Q$
  Merge all chains in $Ch$
```

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. k is defined as the maxi-

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 2 Generation of a closed QN

```
Require: Activity diagram $G = (A, T)$, Deployment diagram $R$, Actor $W$ stereotyped as $\ll$PAclosedLoad$\gg$
  Let $count[r] := 0$, for each $r \in R$
  for all $a \in A$ do
    $count[res[a]] := count[res[a]] + 1$
    $index[a] := count[res[a]]$
  $C := \max_{r \in R}\{count[r]\}$          {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
    $i := id[res[x]]$
    $r := index[x]$
    $j := id[res[y]]$
    $s := index[y]$
    $P[i, r, j, s] := p[t]$
  {Define Service Rates $\mu$}
  for all $a \in A - \{a_0\}$ do
    $i := id[res[x]]$
    $r := index[a]$
    $\mu[i, r] := rate[r]/demand[a]$
  Let service center 0 be a Delay Center
  $\mu[0, \cdot] := 1/extdelay[W]$ for all job classes
  $N := population[W]$
```

mum 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 [3]: (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 can not include fork and join nodes. If such constraints do not hold, a QN model can still be derived from the UML model. However, the QN model does not have a product-form solution, and its analysis requires approximate solution techniques or simulation. These limitations are related to the QN performance model only, as the UML SPT profile allows a general system model representation and annotation.

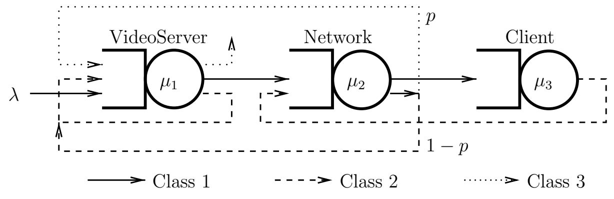


Figure 4: QN for the `Show Video` Activity diagram.

3. EXAMPLE

As an example, we consider the UML diagrams shown in Fig. 1–3. The `User` workload of Fig. 1 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. The transformation algorithm produces the QN in Fig. 4 which has three customer classes which 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 translated according to Algorithm 2 into the closed network of Fig. 5. The network has 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. The QN has a single job class. Service rates are $\mu_0 = 0.01$, $\mu_1 = 2$, $\mu_2 = 2$, $\mu_3 = 1/3$

Finally, the complete multiclass QN model corresponding to the whole system is shown in Fig. 6. 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.

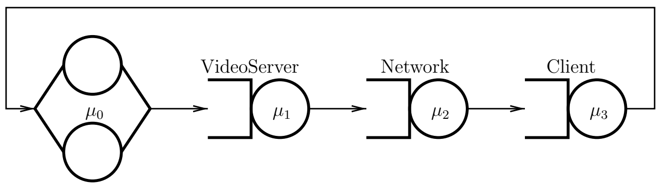


Figure 5: QN for the `Update Client` Activity diagram.

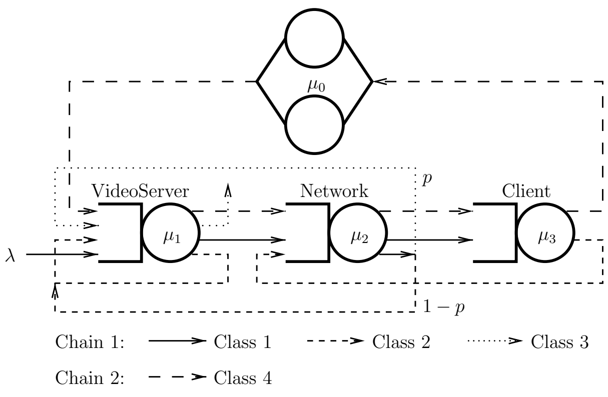


Figure 6: The full QN model

The QN model of Fig. 6 can 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.

4. 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. 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; performance results can be reported back into the software model as tagged values associated to the corresponding UML elements. It should be observed that 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 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 [14, 12, 11], or simulation [7, 6]).

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. Finally, we plan to extend our approach to component and composite structure diagrams of UML 2.0.

5. REFERENCES

[1] 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.

[2] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Trans. Soft. Eng*, 30(5), May 2004.

[3] 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.

[4] 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.

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

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

[7] M. Marzolla and S. Balsamo. UML-PSI: The UML Performance SImulator. In *Proc. QEST 2004*, pages 340–341, Enschede, The Netherlands, Sept. 27–30 2004. IEEE Computer Society.

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

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

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

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

[12] 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.

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

[14] C. M. Woodside, C. Hrischuk, B. Selic, and S. Brayarov. Automated performance modeling of software generated by a design environment. *Performance Evaluation*, 45:107–123, 2001.