Efficient Performance Models in Component-Based Software Engineering*

Simonetta Balsamo Moreno Marzolla†
Dip. di Informatica, Università Ca' Foscari di Venezia
via Torino 155, 30172 Mestre, Italy
{balsamo,marzolla}@dsi.unive.it

Raffaela Mirandola
Dip. di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
mirandola@elet.polimi.it

Abstract

Performance evaluation of Component-Based software systems should be performed as early as possible during the software development life cycle. Unfortunately, a detailed quantitative analysis is often not possible during such stages, as only the system outline is available, with very little quantitative knowledge. In this paper we propose an approach based on Queueing Network analysis for performance evaluation of component-based software systems at the software architectural level. Our approach provides performance bounds which can be efficiently computed. Starting from annotated UML diagrams we compute bounds on the system throughput and response time without explicitly deriving or solving the underlying multichain and multiclass Queueing Network model. We illustrate with an example how the technique can be applied to answer many performance-related questions which may arise during the software design phase.

1. Introduction

Component-Based Software Engineering (CBSE) is the emerging paradigm for the development of large complex software-intensive systems. The basic principle underlying CBSE is that the individual components are released once and for all with documented properties and that the properties then resulting for an assembled system can be obtained from these in compositional way. While this principle has been actively studied for the system functional properties both from industrial and academic communities, only in the last few years some attention has been paid also to include quantitative performance validation within component-based software development processes [2]. Since the ability to predict the performance characteristics of the assembled system should be applied as early as possible to reduce the costs of late problem fixing [10], it becomes crucial to determine from the design phase whether the component-based product will satisfy its requirements. In other words, the software designer should be able to operate "performance critical" choices since the abstract level of architectural specification.

In this paper we propose the use of simple and efficient methods for the performance analysis of Component-Based (CB) systems that are based on the well-known operational laws [7] of Queueing Network (QN) analysis to perform bottleneck analysis directly on the software system specification without requiring the derivation of performance models. Bound analysis is one of the most important steps of performance planning, and consists on identifying the resource(s) which constraints the system performance. We consider software specifications in term of annotated Unified Modeling Language (UML) Use Case, Activity and Deployment diagrams. We use the UML Schedulability, Performance and Time specification (SPT) profile for inserting quantitative, performance-oriented annotations into the UML model. Our approach computes performance bounds which are based on a multiclass and multichain QN model, although we do not derive nor evaluate explicitly the QN model from the software specification. This makes our approach well suited for efficiently answering, at the software architectural level, many performance-related questions without the need for providing too many details .

The advantage of the proposed approach is that perfor-

*This work has been partially supported by MIUR research project FIRB: "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools".

†Author's current affiliation: INFN Sezione di Padova, via Marzolo 8, 35131 Padova, Italy

mance bounds can be obtained with little computational effort, allowing the software developer to quickly answer a set of common performance-related questions arising during the components development cycle. If necessary, more accurate bounds can be derived by simply applying different techniques (e.g., the one described in [1]). Our approach can be applied: (1) at design time, to select components based on their expected performance, or to estimate the expected overall system performance; (2) at run time to reconfigure the system, e.g., after a component failed and is necessary to select a replacement one with performance guarantees. Moreover, the proposed approach can be useful for answering what-if questions regarding, for example, the resource representing the bottleneck if the platform configuration is changed, or the expected response time if a component is substituted with a different one and so on. Our short term goal is the integration of this method into the Component-Based Software Performance Engineering (CB-SPE) framework to enhance its potentialities with this simple and efficient method. In this way the software designer, according to the owned info, can choose the analysis method more suitable.

This 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. In Sections 3 trough 5 we describe our approach for bound analysis of software systems. Section 6 illustrates with an example how the proposed approach can actually be used. Finally, conclusions and future research directions are discussed in Section 7.

2. Related work

As already outlined in the introduction, the integration of quantitative evaluation into the software development processes is an important activity to meet non-functional, and in particular performance requirements. Balsamo et al. [2] presents a survey of different approaches for model-based performance evaluation. The proposed classification is based on the type of the performance model (Queueing Networks, Petri Nets, Process Algebras, Markov Processes), the applied evaluation method (analytical or simulative) and the presence of automated support for performance prediction. Some of these approaches have also been extended to deal with component-based systems. Becker et al. [3] presents a survey of the existing approaches for predictive performance analysis of CB systems based on the kind of analysis applied (quantitative or qualitative) and on the type of techniques used (measurement-based, model-based, combination of measurement-based and model-based).

Examples of **quantitative** approaches based on measurement can be found in [13, 5, 6]. The work in [13] is an introductive study and is simply based on the monitoring of single applications, while results presented in [5, 6] seem more promising to deal with component-based systems. Both of them take middleware details into account and consider J2EE application with EJB containers. The methods based on measurements show a good accuracy of obtained results, while their cost effectiveness is low, as they require already implemented systems. Moreover, they are often platform-specific and this limits their adaptability and scalability.

Several model-based approaches for the predictability of performance behavior of component-based systems have been proposed in the last years (see [2] for a comprehensive review). Each of these focuses on a particular type of source design-oriented model and a particular type of target analysis-oriented model, with the former spanning UML, Message Sequence Chart, Use Case Maps, formal language as Æmilia, ADL languages as Acme, and the latter spanning Petri nets, queueing networks, layered queueing network, stochastic process algebras, Markov processes (see [3] for a thorough overview of these proposals). The main weak point of model-based approaches is the lack of empirical studies for the validation of the predicted results.

Qualitative approaches evaluate the quality of component-based systems, either based on the affinity between software architecture and software components [12] or exploiting the principles of the model driven engineering ([11], [8]). The common characteristic of these approaches is to consider qualitative analyses that are derived from an attribute-based style or trough "screening questions" and are meant to be coarse-grained versions of the quantitative analysis that can be performed when a precise analytic model of a quality attribute is built.

3. Methodology Overview

We consider component-based applications, built up from software components glued together by means of some integration mechanism. In this context, the components provide the application-specific functionalities (and are mainly considered as black boxes), and the glue defines the workflow that integrates these functionalities to deliver the services required from the CB application.

In Fig. 1 we show an Activity Diagram with the steps of the proposed methodology. Swimlanes show who is responsible for executing each action.

Step 1:Components with performance annotations We assume that components are provided with interfaces that explicitly declare the component estimated performance properties. The estimate performance should be used by the Software Architect to identify those components which better fulfills the non-functional, performance-related requirements.

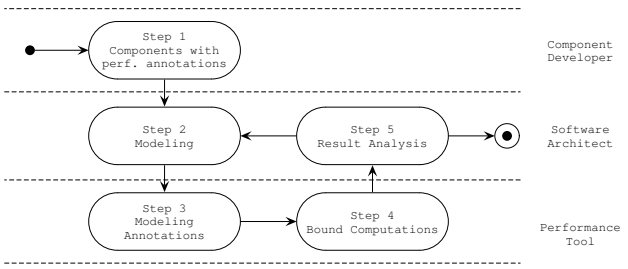

Figure 1. Activity diagram for the proposed methodology

Step 2: System Modelling The Software Architect should identify the different types of application users and the corresponding Use Cases. The different application workflows are described through Activity diagrams where the activities represent either the requests of execution of a component service or correspond to information/data exchanged between the components and the swimlanes describe the involved components. Finally, a Deployment diagram (DD) will model the available resources and their characteristics. In this case the nodes of the DD can be associated to classical resources (device, processor, database) and communication devices. The UML model can be built by combining partial UML specifications of the individual system components, whenever these are available.

Step 3: Model Annotations The three types of diagrams developed at step 2 have to be annotated with the proper values and parameters. Annotations are given according to the UML SPT profile [9]. The annotations are used to describe quantitative, performance-oriented information to be associated to single system elements (processing steps, workloads, resources); annotation values can either be inferred from vendor-supplied informations (e.g., datasheets), estimated, or measured on available, similar systems.

Step 4: Bound Computation The annotated UML specifications are used to derive a multiclass QN model. Actors in Use Case diagrams correspond to workloads in the QN model; nodes in Deployment diagrams correspond to service centers; Activity diagrams describe the sequence of requests which customers perform on service centers. The QN model is used to compute performance bounds on throughput and response time. Such informations correspond to bounds on throughput and response times of the resources on the software system under evaluation. The performance model which we implicitly consider is a multiclass QN model, in which we associate each Actor in the UML specification to a customer class in the QN model. Bound analysis does not require the explicit derivation of the QN model. Instead, we need to compute the service demands only, which can easily be done by using the UML model annotations and by computing the visit ratios of the service centers, as will be described in the following section.

Step 5: Result Analysis The Software Architect uses the computed performance bounds to choose among the components available those that better fulfil the settled performance requirements; performance results can also be used to answer "what-if" questions about the system. Based on the analysis results provided by the tool, the Software Architect can reach a more informed decision about the system design. If the performance requirements are fulfilled, he/she can proceed with the acquisition of the pre-selected components and their assembly; otherwise he/she has to iterate the process by repeating the steps described, or lastly admit the infeasibility of the performance requirements with the acquisition of off-the-shelf components.

In the following we give a detailed description of steps 2–4; step 5 is illustrated through the example in section 6; the reader is referred to [4] for more information on step 1.

4. Modelling Software Systems (Steps 2 and 3)

As introduced in the previous section, we consider a CB system described as a set of annotated UML Use Case, Activity and Deployment diagrams. The diagrams are annotated according to a subset of the UML Profile for Schedulability, Performance and Time specification [9]. The UML SPT profile defines a set of stereotypes and tagged values which can be used to associate quantitative, performance-oriented informations to UML elements.

In the following, given a component X we denote by $Att[X]$ the value of attribute *Att* for component X. We consider a software system model $(\mathcal{U}, \mathcal{A}, \mathcal{R})$, where $\mathcal{U} = (U_1, \ldots U_K)$ is the set of Actors, $\mathcal{A} = (A_1, \ldots A_K)$ is the set of Activity diagrams, and $\mathcal{R} = (R_1, \ldots R_M)$ is the set of resources (Deployment diagram nodes). Each Activity diagram A_i describes the internal behavior of one Use case, which is associated to an Actor. Actors are used to represent workloads, i.e., an Actor represents a set of external requests arriving to the system. Internally-generated events, such as periodic activities triggered by timers, can be described in the same way, by representing the "time" as an Actor, and specifying on such actor the appropriate tagged values describing its timing behavior.

Actor U_i may denote an *open* workload if it is stereotyped as $\ll$PAopenLoad$\gg$. In this case U_i represents an infinite stream of requests arriving to the system with rate $arrivalrate[U_i]$. Actor U_i denotes a *closed* workload if it is stereotyped as $\ll$PAclosedLoad$\gg$. Then U_i represents

a fixed population of $population[U_i]$ requests. After completion of the service in the system, the request spends time $extdelay[U_i]$ outside the system before the next interaction. Requests generated by Actor U_i trigger the execution of Activity diagram A_i, which describes the Use Case associated with U_i. In order to simplify the notation, we assume that only one Use Case is associated with each Actor.

Each Activity diagram A_i is a set of n_i states $\{a_{i1}, a_{i2}, \ldots a_{in_i}\}$, $1 \leq i \leq K$. We consider seven different types of states: *Start nodes* (the point from which execution starts), *End nodes* (the point where execution stops), *Action nodes* (a request for service from some resource), *Fork nodes* (used to split the execution flow in parallel threads), *Join nodes* (used as a synchronization point to merge parallel threads back into a single execution flow), *Branch nodes* (denoting alternative execution flows) and *Merge nodes* (used to balance the corresponding branch nodes). Without loss of generality, assume that for each Activity diagram A_i the Start node is a_{i1} and the End node is a_{in_i}.

Transitions of Activity diagrams are labeled with probabilities, where $p[a_{ij}, a_{ik}]$ is the probability that action a_{ik} will be executed immediately after completion of action a_{ij}. Transition probabilities $p[\cdot]$ can be described as tagged values associated to each transition. If a_{ik} is not one of the successors of a_{ij}, then $p[a_{ij}, a_{ik}] = 0$. If a_{ik} is the only successor of a_{ij}, then $p[a_{ij}, a_{ik}] = 1$. We require that for each $1 \leq i \leq K$, $1 \leq j \leq n_i$, $\sum_{k=1}^{n_i} p[a_{ij}, a_{ik}] = 1$. Each action a_{ij} is executed on resource $res[a_{ij}] \in \mathcal{R}$. The service demand is denoted as $demand[a_{ij}]$, and the number of repetitions of each action is denoted as $rep[a_{ij}]$.

5. Bound Analysis of Software Systems (Step 4)

The annotated UML model can be used to derive a multiclass QN model made of exactly M service centers and K different customer classes. Each service center in the QN corresponds to one of the resources $(R_1, \ldots R_M)$, and there is one customer class for each workload. The bound analysis of a QN simply requires the knowledge of the overall service demand of the customer to each node. Hence we derive this demand starting from the component annotations introduced at step 3. The overall service demand can be defined as the product of the visit ratio and the service demand for each visit. We shall now describe the derivation of these two quantities.

5.1. Computation of the visit ratios

Let us consider an Activity diagram A_i driven by the workload represented by Actor U_i. We denote with V_{ij} the *visit ratio* of state a_{ij}, that is, the ratio between the number of visits to state a_{ij} and the number of completions for the whole activity diagram A_i. Visit ratios can be computed according to the following equations:

$$V_{i1} = \begin{cases} arrivalrate[U_i] & \text{if } U_i \text{ is an open workload;} \\ 1 & \text{otherwise.} \end{cases} \quad (1)$$

$$V_{ij} = \sum_{k=1}^{n_i} p[a_{ik}, a_{ij}] V_{ik} \quad \text{if } a_{ij} \text{ is not a Fork nor Join node} \quad (2)$$

Note that Eq. 2 also holds, under certain assumptions, if a_{ij} is a fork or join node. Specifically, the equation can be applied if fork and join nodes are balanced, which is often the case in actual Activity diagrams.

5.2. Computation of the service demand

We shall now derive d_{ij} for each action state a_{ij}. The value of d_{ij} depends on the service demand of each execution of action a_{ij} ($demand[a_{ij}]$), on the number of times the action must be repeated ($rep[a_{ij}]$), on the processing rate of the resource on which a_{ij} is executed ($rate[res[a_{ij}]]$), and on the visit ratio V_{ij}, as follows:

$$d_{ij} = \frac{demand[a_{ij}]}{rate[res[a_{ij}]]} \times rep[a_{ij}] \times V_{ij} \quad (3)$$

The overall service demand D_i^j of resource $R_i \in \mathcal{R}$ for workload $U_j \in \mathcal{U}$ is the sum of the individual resource demands over all actions of workload U_j (associated to Activity diagram A_j) requesting service from resource R_i and is defined as follows:

$$D_i^j = \sum_{k \mid a_{jk} \in A_j \wedge res[a_{jk}] = R_i} d_{jk} \quad (4)$$

Hence, the overall service demand D_i for resource R_i, and the service demand D^j for workload U_j, can be defined respectively as follows:

$$D_i = \sum_{j=1}^{K} D_i^j \quad (5) \qquad D^j = \sum_{i=1}^{M} D_i^j \quad (6)$$

5.3. Bounds for Multiclass QN

We shall now derive the performance bounds for the software model. We recall the definition of some simple performance bounds for multiclass QN models, which extend those for single class models [7]. The proposed methodology is independent of the specific algorithm used to compute bounds. Hence one can apply the proposed

approach by using more accurate, although usually more compute-intensive, algorithms if better bound estimates are required [1].

We consider a mixed multiclass QN model with M service centers and K customer classes. We denote with $\mathbf{M} = \{1, \dots M\}$ the set of service center indexes, and with $\mathbf{K} = \{1, \dots K\}$ the set of customer class indexes. Each customer class can be either open or closed, depending whether its customer population is unlimited or fixed. We denote with $\mathbf{C} \subseteq \mathbf{K}$ the set of indices of closed classes; $\mathbf{O} = \mathbf{K} - \mathbf{C}$ is the set of indices of open classes.

For each open class $c \in \mathbf{O}$, parameter λ^c denotes the arrival rate of class c customers. For each closed class $c \in \mathbf{C}$, parameters (N^c, Z^c) denote the total (fixed) number of customers and their external "think time", respectively. The workload intensity vector for the entire system is thus $\mathbf{T} = (\mathbf{T}_1, \mathbf{T}_2, \dots \mathbf{T}_K)$ where, for each $i \in \mathbf{K}$, $\mathbf{T}_i = \lambda_i$ if i is an open class, $\mathbf{T}_i = (N^i, Z^i)$ if i is a closed class. We denote with D_i^c the service demand of class c customers at service center S_i, $c \in \mathbf{K}$, $i \in \mathbf{M}$. The total demand D_i on service center S_i is $D_i = \sum_{c \in \mathbf{K}} D_i^c$. Device S_{max} with the greatest demand $D_{max} = \max_i D_i$ is the *bottleneck device*. We denote with $D_{max}^c = \max_i D_i^c$ the bottleneck device for class c customers. We also denote with $D^c = \sum_{i \in \mathbf{M}} D_i^c$ the total service demand for class c customers.

We define $X(\mathbf{T})$ and $R(\mathbf{T})$ as the system throughput and response time with given workload intensity vector $\mathbf{T}$. Moreover, we define $X^c(\mathbf{T})$ and $R^c(\mathbf{T})$ as class c throughput and response time, respectively.

We now derive some simple bounds for server i throughput and response time, and for system throughput and response time, based on an extension of the well known results for single class bounds [7].

Bound on Throughput The throughput for a class $i \in \mathbf{K}$ customers cannot exceed $1/D_{max}^i$, due to class i bottleneck device. Thus we can write:

$$X^i(\mathbf{T}) \leq \frac{1}{D_{max}^i}, \qquad i \in \mathbf{K} \tag{7}$$

and similarly, the whole system throughput $X(\mathbf{T})$ can be defined as $X(\mathbf{T}) \leq 1/D_{max}$

For closed workloads we also note that the maximum throughput can be reached when jobs do not interfere each other, so that no queueing occurs. For the generic closed class $j \in \mathbf{C}$, each customer would spend time D^j being served, and time Z^c outside the system. In this case, taking also in consideration the generic upper bound from Eq. 7, we have:

$$X^j(\mathbf{T}) \leq \min\left(\frac{1}{D_{max}^j}, \frac{N^j}{D^j + Z^j}\right), \quad j \in \mathbf{C} \tag{8}$$

If $\mathbf{O} = \emptyset$, we can provide also a lower bound on the throughput for a closed classes. The minimum throughput for class $j \in \mathbf{C}$ customers happens when each customer is queued behind all other system customers before being served. Thus, each class j customer would spend $(N^j - 1)D^j + \sum_{i \neq j} N^i D^i$ time units being queued, D^j time in service and Z^j time waiting. Thus, the overall class j minimum throughput is $N^j/(Z^j + \sum_{i \in \mathbf{C}} N^i D^i)$

Thus, in the case $\mathbf{O} = \emptyset$, the bounds for closed class $j \in \mathbf{C}$ throughput $X^j(\mathbf{T})$ are

$$\frac{N^j}{Z^j + \sum_{i \in \mathbf{C}} N^i D^i} \leq X^j(\mathbf{T}) \leq \min\left(\frac{1}{D_{max}^j}, \frac{N^j}{D^j + Z^j}\right) \tag{9}$$

Combining Eq. 8 and 9 we can obtain an overall bound on the system throughput $X(\mathbf{T})$ as follows:

$$\begin{aligned} X(\mathbf{T}) &= \sum_{i \in \mathbf{C}} X^i(\mathbf{T}) + \sum_{j \in \mathbf{O}} X^j(\mathbf{T}) \\ &\leq \sum_{i \in \mathbf{O}} \frac{1}{D_{max}^i} + \sum_{j \in \mathbf{C}} \min\left(\frac{1}{D_{max}^j}, \frac{N^j}{D^j + Z^j}\right) \end{aligned}$$

In general, if $\mathbf{O} \neq \emptyset$ there are open workload, i.e., it is not possible to provide a lower bound on the system throughput.

Bound on Response Time Response time $R^i(\mathbf{T})$ for class $i \in \mathbf{K}$ cannot be less than the total service demand D^i: $R^i(\mathbf{T}) \geq D^i$. No upper bound on response time can be given for open classes. For closed classes, it is possible to apply Little's law to transform bounds on throughput in bounds on response time. For the generic closed class $j \in \mathbf{C}$ we have: $X^j(\mathbf{T}) = N^j/(R^j(\mathbf{T}) + Z^j)$. Thus, from Eq. 9 we obtain:

$$\max\left(N^j D_{max}^j - Z^j, D^j\right) \leq R^j \leq \sum_{i \in \mathbf{C}} N^i D^i \tag{10}$$

Table 1 summarizes the performance bounds for open and closed customer classes, respectively. In mixed networks we may have different customer classes; performance bounds apply to individual classes of mixed networks.

5.4. The Bound Analysis Algorithm

We now summarize in Algorithm 1 the steps for the bound analysis algorithm for CB systems, that is step 4 of the proposed methodology. The algorithm can be used to automatically compute performance bounds on annotated UML specifications $(\mathcal{U}, \mathcal{A}, \mathcal{R})$.

Open Networks	Throughput	$X^i(\mathbf{T}) \leq \frac{1}{D^i_{max}}$
	Response Time	$R^i(\mathbf{T}) \geq D^i$
Closed Networks	Throughput	$\frac{N^j}{\sum_{i \in \mathbf{C}} N^i D^i + Z^j} \leq X^j(\mathbf{T}) \leq \min\left(\frac{1}{D^j_{max}}, \frac{N^j}{D^j + Z^j}\right), \quad \text{if } \mathbf{O} = \emptyset$ $X^j(\mathbf{T}) \leq \min\left(\frac{1}{D^j_{max}}, \frac{N^j}{D^j + Z^j}\right), \quad \text{if } \mathbf{O} \neq \emptyset$
	Response Time	$\max\left(N^j D^j_{max} - Z^j, D^j\right) \leq R^j(\mathbf{T}) \leq \sum_{i \in \mathbf{C}} N^i D^i, \quad \text{if } \mathbf{O} = \emptyset$ $R^j(\mathbf{T}) \geq \max\left(N^j D^j_{max} - Z^j, D^j\right), \quad \text{if } \mathbf{O} \neq \emptyset$

Table 1. Summary of Performance Bounds

Algorithm 1 Bound Analysis

$\mathbf{C} := \emptyset$ {Set of closed classes}
$\mathbf{O} := \emptyset$ {Set of open classes}
$\mathbf{K} := \{1, 2, \ldots K\}$ {QN customer classes}
for all $U_j \in \mathcal{U}$ **do**
  if U_j is an *open* workload **then**
    $\mathbf{O} := \mathbf{O} \cup \{j\}$
    $N^j := population[U_j]$
    $Z^j := extdelay[U_j]$
  else
    $\mathbf{C} := \mathbf{C} \cup \{j\}$
    $\lambda^j := arrivalrate[U_j]$
for all Activity diagram $A_i \in \mathcal{A}$ **do**
  for all Action state $a_{ij} \in A_i$ **do**
    Compute the visit ratios V_{ij} using 1–2
    Compute the service demand d_{ij} using 3
for all Resource $R_i \in \mathcal{R}$ **do**
  Compute the service demand D_i using 5
for all Workload $U_j \in \mathcal{U}$ **do**
  Compute the service demand D^j using 6
Use Table 1 to compute bounds

6. An Example

In this section we illustrate with an example a simple application of the proposed approach. We consider a simple E-Commerce application where users interact with a Web Server component to navigate an electronic catalog of products, and to purchase items. The Web Server component interacts with a DB server component which resides on a different machine connected through a LAN. There is also a fixed number N of processes which perform periodic maintenance operations on the DB server. We shall now apply the steps of the proposed methodology to derive performance bounds.

Step 1: Components with performance annotations The system has two components, a Web Server and a Database Server component. Due to lack of space, we do not show how performance properties of these components are declared within the component interfaces; the reader is referred to [4] for details.

Step 2 and 3: System Modeling and Annotations The system is deployed over the physical resources described in the annotated UML diagram of Fig. 2. Three resources are shown: R_1 (labelled "Web Server"), R_2 ("DB Server") and R_3 ("Network"); the processing rates are r_1, r_2 and r_3, respectively. There is a *Web Server* component which is hosted on resource R_1, and a *DB Server* component hosted on resource R_2.

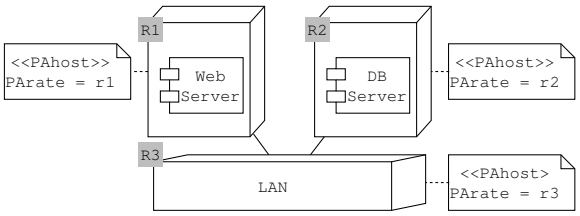

Figure 2. Deployment diagram

The system is subject to the external interactions described by the annotated Use Case diagram shown in Fig. 3. The system processes two kind of requests. *Workload* U_1 is a closed population of N processes which execute periodic maintenance operations on the DB server. After execution, each process waits for an average time Z before interacting again with the system. This workload is represented by the "Time" Actor. *Workload* U_2 is an open population of customers purchasing items from the online catalog. Customers arrival rate is λ. This workload is represented by the "User" Actor.

For the sake of simplicity, we omit class indices for parameters N, Z and λ as in this case there can be no confusion among classes.

The actions executed by the maintenance processes and by the customers are shown in the Activity diagram of Fig. 4 and 5, respectively. Each action in the Activity diagrams is labelled as a_{ij}.

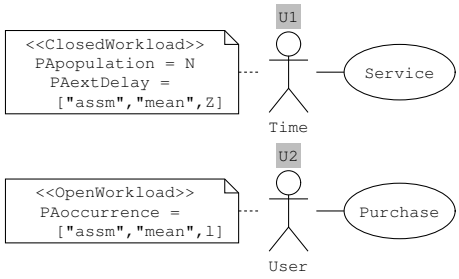

Figure 3. Use Case diagram

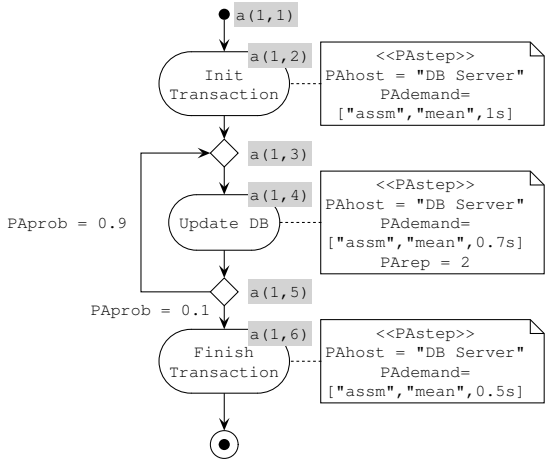

Figure 4. Activity diagram A_1 for the "Service" Use Case

Step 4: Bound computation Using the annotations in the diagrams, we can compute the visit ratios V_{ij} according to Eq. 1–2, and the service demands d_{ij} for each action a_{ij} according to Eq. 4. The results are reported in Table 2.

The service demands D_i^j for resource R_i from workload U_j can be computed using the informations summarized in Table 2, and are: $D_1^1 = 0, D_2^1 = d_{1,2} + d_{1,4} + d_{1,6} = 4.1/r_2, D_3^1 = 0, D_1^2 = d_{2,2}+d_{2,8}+d_{2,10} = 4.3\lambda/r_1, D_2^2 = d_{2,6} = \lambda/r_2, D_3^2 = d_{2,5} + d_{2,7} = 0.5\lambda/r_3$.

The workload demands D^j, and the resource demands D_i are: $D^1 = \frac{4.1}{r_2}$, $D^2 = \frac{4.3\lambda}{r_1} + \frac{\lambda}{r_2} + \frac{0.5\lambda}{r_3}$, $D_1 = \frac{4.3\lambda}{r_1}$, $D_2 = \frac{4.1+\lambda}{r_2}$, $D_3 = \frac{0.5\lambda}{r_3}$

Step 5: Result Analysis It is now possible to answer a number of questions as may be posed by a software architect seeking understanding about the system performance. Consider, for example, the following questions:

Question 1 *Assume that the processing rates are all equal to 1 ($r_1 = r_2 = r_3 = 1$). Which resource (component) is the bottleneck device in our system?*

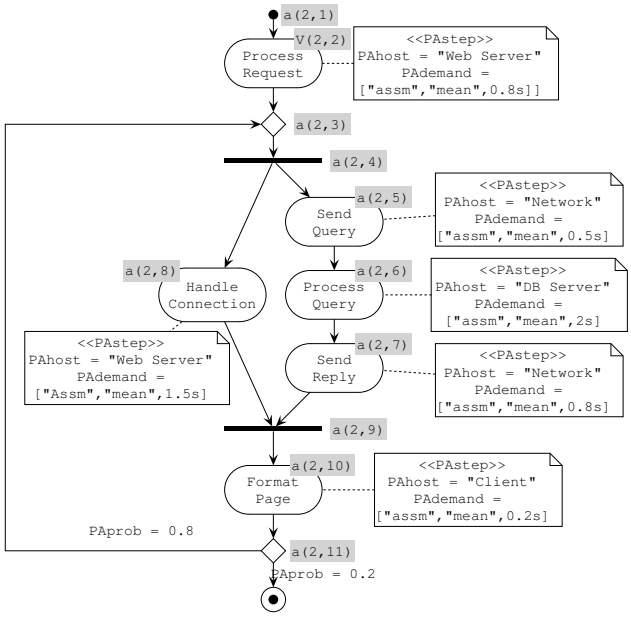

Figure 5. Activity diagram A_2 for the "Purchase" Use Case

Answer The device with the greatest service demand is the bottleneck device. The resource demands D_1, D_2, D_3 for resources R_1, R_2 and R_3 respectively, are $D_1 = 4.3\lambda$, $D_2 = 4.1 + \lambda$, $D_3 = 0.5\lambda$. We then get that the bottleneck device is R_2 if $0 < \lambda \leq 41/33$, and is R_1 if $\lambda > 41/33$.

Question 2 *Usability practices require that the mean response time of interactive Web users should be less than 6 seconds. Is this requirement satisfied in our example, assuming processing rates $r_1 = r_2 = r_3 = 1$ a customer arrival rate of $\lambda = 2$?*

Answer We apply the Response Time bound for the open population of user requests corresponding to workload U_2. The total service demand for interactive users is $D^2 = 5.8\lambda = 11.6$; from the Response Time bound we have that $R^2(\mathbf{T}) \geq 11.6s$. Then, the requirement of $R^2(\mathbf{T}) \leq 6s$ cannot be satisfied with the current system configuration.

In order to identify the bottleneck device for workload U_2, let us rewrite D^2 as a function of the processing rates r_1, r_2 and r_3: $D^2 = \frac{8.6}{r_1} + \frac{2}{r_2} + \frac{1}{r_3}$.

It turns out that, in order to get a response time of at most $6s$ the only possibility is to improve the processing rate r_1 of resource R^1 or, alternatively, select a different Web Server component with lower service demand. If $r_1 \leq 43/15$ the requirement of $R^2 \leq 6s$ cannot be satisfied. The same result can be obtained by replacing the "Web Server" component with a faster one.

Action	a_{12}	a_{14}	a_{16}	a_{22}	a_{25}	a_{26}	a_{27}	a_{28}	$a_{2\ 10}$
Visit Ratio V_{ij}	1	10	1	λ	$\lambda/0.2$	$\lambda/0.2$	$\lambda/0.2$	$\lambda/0.2$	$\lambda/0.2$
Service Demand d_{ij}	$0.05/r_2$	$4/r_2$	$0.05/r_2$	$0.8\lambda/r_1$	$0.25\lambda/r_3$	λ/r_2	$0.25\lambda/r_3$	λ/r_1	$2.5\lambda/r_1$

Table 2. Visit ratios and service demands

Question 3 *What is the maximum system throughput, assuming a closed user population of $N = 5$ processes, external delay $Z = 600$, open customer arrival rate of $\lambda = 2$ and processing rates $r_1 = 2, r_2 = r_3 = 1$?*

Answer We apply the Throughput bound for the whole system. For the given parameters we have $1/D_{max} \approx 0.1639$, $N/(D^1 + Z) \approx 0.0083$; hence $X(\mathbf{T}) \leq \min(0.1639, 0.0083) = 0.0083$. The global system throughput is bound by 0.0083 requests/second. This means that the actual system throughput is less than or equal to the computed quantity.

The previous examples show some of the questions which can be answered with the proposed bound analysis algorithm. For those situations in which bound results alone are not enough to provide a suitable answer, the proposed technique might be coupled with other model-based software performance evaluation approaches. Bounds can be used to get a first approximation of system performances, which can be later investigated further with other more accurate (but possibly more compute-intensive) techniques.

7. Conclusions

In this paper we described an approach for performance prediction of component-base software systems at the architectural level based on operational analysis of QN models. We showed how asymptotic bounds for system throughput and response time can be efficiently derived without the need to derive a QN model from the software specification. Software performance bounds can be used to answer several performance-related questions. Once the performance annotations are available, the proposed approach can be fully automated, which means that it can be integrated into existing UML-based CASE tools.

Future works includes the actual integration of bound analysis into a CASE tool, as well as its validation by application to real case studies. Our short-medium term goal is the realization of a general framework for computer assisted discovery and composition of software components, encompassing the evaluation of non-functional properties. For this reason it would be very useful to integrate the bound analysis described here with other software performance modeling approaches based on the UML SPT profile (such as CB-SPE). In this way the software designer could choose the performance evaluation tool best suited for the kind of analysis to be performed.

References

[1] G. Balbo and G. Serazzi. Asymptotic analysis of multiclass closed queueing networks: Multiple bottlenecks. *Performance Evaluation*, 30:115–152, 1997.

[2] 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):295–310, 2004.

[3] S. Becker, L. Grunske, R. Mirandola, and S. Overhage. Performance prediction of component-based systems: A survey from an engineering perspective. In R. Reussner, J. Stafford, and C. Szyperski, editors, *Architecting Systems with Trustworthy Components*, volume 3938 of *LNCS*, 2006. to appear.

[4] A. Bertolino and R. Mirandola. CB-SPE Tool: Putting Component-Based Performance Engineering into Practice. In *CBSE 2004*, volume 3054 of *LNCS*, pages 233–248. Springer, 2004.

[5] S. Chen, I. Gorton, A. Liu, and Y. Liu. Performance Prediction of COTS Component-Based Enterprise Applications. In *Proceedings CBSE 2002*, 2002.

[6] G. Denaro, A. Polini, and W. Emmerich. Early Performance Testing of Distributed Software Applications. In *Proc. WOSP 2004*, pages 94–103, Redwood Shores, California, Jan. 14–16 2004. ACM Press.

[7] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, 1978.

[8] V. Grassi and R. Mirandola. A Model-Driven Approach to Predictive Non-Functional Analysis of Component-Based Systems. In *Proc. NfC-04*, Lisbon, Portugal, Oct. 12 2004. ISSN 1430-211X.

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

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

[11] A. Solberg, K. E. Husa, J. . Aagedal, and E. Abrahamsen. *QoS-aware MDA*. Elsevier, 2004. ISSN:1571-0661.

[12] K. C. Wallnau, J. Stafford, S. Hissam, and M. Klein. On the relationship of software architecture to software component technology. In *Proceedings WCOP'01, Budapest, Hungary, 19 June 2001*, 2001.

[13] S. M. Yacoub. Performance Analysis of Component-Based Applications. In *Proceedings of SPLC 2*, volume 2379 of *LNCS*, pages 299–315. Springer, Aug. 19–22 2002.