ARTICLE IN PRESS

ELSEVIER

Performance Evaluation 974 (2002) 1–20

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

A review on queueing network models with finite capacity queues for software architectures performance prediction

Simonetta Balsamo [a], Vittoria De Nitto Personè [b], Paola Inverardi [c,*]

[a] *Dip. di Informatica, University Ca'Foscari of Venezia, Venice, Italy*
[b] *Dip. di Informatica, Sistemi e Produzione, University of Roma Tor Vergata, Rome, Italy*
[c] *Dip. di Matematica Pura e Applicata, University of L'Aquila, L'Aquila, Italy*

Abstract

A review is carried out on how queueing network models with blocking have so far been applied into the performance evaluation and prediction of software architectures (SAs). Queueing network models with finite capacity queues and blocking have recently been introduced and applied as more realistic models of systems with finite capacity resources and population constraints. Queueing network models have been often adopted as models for the evaluation of software performance. Starting from our own experience, we observe the need of a more accurate definition of the performance models of SAs to capture some features of the communication systems. We consider queueing networks with finite capacity and blocking after service (BAS) to represent some synchronization constraints that cannot be easily modeled with queueing network models with infinite capacity queues. We investigate the use of queueing networks with blocking as performance models of SAs with concurrent components and synchronous communication. Queueing theoretic analysis is used to solve the queueing network model and study the synchronous communication and performance of concurrent software components. Our experience is supported by other approaches that also propose the use of queueing networks with blocking. Directions for future research work in the field are included.

Keywords: Queueing network; Blocking after service; Software

1. Introduction

Queueing network models have been extensively applied in the last decades as a powerful tool for modeling and performance evaluation and prediction of computer and systems, as well as production and manufacturing systems [25,27,30,33]. Queueing network models with finite capacity queues and blocking have been introduced and applied as more realistic models of systems with finite capacity resources and population constraints [3,7,8,35,36,39,40]. In the last few years, there is growing interest in integrating performance evaluation tools in early stages of software development life cycle and queueing network models have been used for quantitative analysis and validation of software systems. Several approaches have been

* Corresponding author.
E-mail address: inverard@univaq.it (P. Inverardi).

0166-5316/02/$ – see front matter © 2002 Published by Elsevier Science B.V.
PII: S0166-5316(02)00099-8

proposed, notably the software performance engineering method introduced by Smith as an integration of software engineering and performance management [37,45,46,52,53]. Recently the software architecture (SA) description of a software system has received much interest as a potentially good candidate to conduct early predictive performance analysis of the system under development [13,14,22,26,52,53].

SA describe software system structures at a high level of abstraction [47] and they have been devised as the appropriate design level to carry out quantitative analysis [13,21,22,52]. They represent at an early stage of development the phase in which basic choices of components and interactions among components are made. These choices are driven by several non-functional issues, which include important factors such as system performance and reliability. In this setting, among others, we have been studying and evaluating the expected performance of SAs. We have proposed a methodology for software performance evaluation at the SA level that derives a performance evaluation model, based on a queueing network model, from a SA formal specification [4,12]. Although we have used different models for describing the dynamic behavior of a SA [5,12,16] our approach can be considered independent from the specific model and can be summarized as follows. We analyze the dynamic behavior of a SA in order to define the queuing network model representing the performance model of the SA behavior. Then we use a scenario-based technique to parameterize and evaluate the obtained QNM. From our perspective of SAs, queueing network models provide a powerful tool that can be defined, parameterized and evaluated at a low cost and with a level of abstraction that allows a faithful modeling.

More recently we have been extending our approach by investigating the use of queueing networks with finite capacity and blocking in order to more accurately model interaction capabilities of SAs.

Specifically, we observe the need of a more accurate definition of the performance models of SAs to capture some features of the communication systems. We consider queueing networks with finite capacity and blocking to represent some synchronization constraints.

In this work we discuss our experience in using queueing network models and their adequacy for the problem at hand in terms both of modeling and its evaluation.

Besides reporting on our own experience we review other approaches [19,31,32,42] that have recently proposed the use of queueing network models with blocking in the software performance area.

The paper is organized as follows. Section 2 introduces the definition and analysis of queueing network models with finite capacity queues and the blocking type definition. Section 3 discusses some recent approaches of software performance analysis based on queueing network models with blocking and briefly summarizes the method that derives a QNM with BAS blocking by message sequence charts (MSCs) as SA description. Section 4 presents the application of queueing network models with finite capacity to model synchronous communications between software components. An application example is illustrated in Section 5 and conclusions are presented in Section 6.

2. Queueing network models with finite capacity queues

Queueing networks with finite capacity queues have been introduced to represent systems with finite capacity resources and population constraints. When a queue reaches its maximum capacity then the flow of customers into the service center is stopped, both from other service centers and from external sources in open networks, and the blocking phenomenon arises. Various blocking mechanisms have been defined and analyzed in the literature to represent distinct behaviors of real systems with limited resources [3,35,40,48,49]. Performance analysis of queueing networks with blocking can be exact or approximate.

Exact solution algorithms have been proposed to evaluate both average performance indices, queue length distribution, and passage time distribution [B94] [8,35,40]. Under exponential assumption one can define and analyze the continuous-time Markov chain underlying the queueing network. In some special cases queueing networks with blocking show a product-form solution, under particular constraints, for various blocking types [11].

2.1. The model

Consider a queueing network model formed by M service centers or nodes and a set of customers. For simplicity we consider single class of customers. Queueing networks can be open or closed. In a closed network a constant number of customers N circulate into the network. For an open network we define an exogenous arrival process at each node i, $1 \leq i \leq M$. Let λ denote total arrival rate at the network and p_{0i}, $1 \leq i \leq M$, the probability that an exogenous arrival tries to enter node i. Then the Poisson arrival process at node i has parameter λp_{0i}. Let $P = [p_{ij}]$ $(1 \leq i, j \leq M)$ denote the routing matrix where p_{ij} is the probability that a job leaving node i tries to enter node j. For each service center we define the number of servers, the service time distribution, the queue capacity and the service discipline. Examples of interarrival and service distributions are exponential (M), phase-type with n exponential stages (PH_n), general (G) and generalized exponential distribution (GE). A Poisson arrival process allows simplifying the model analytical solution. However, one can consider more realistic arrival processes such as, for example, compound Poisson processes (CPP) or Markov modulated Poisson processes (MMPP) to represent batch arrivals, bursty traffic and state dependent Poisson processes.

Let $\mathbf{S}_i$ denote the state of node i, which includes the number of customers in node i, denoted by n_i, and other components depending both on the node type (service discipline and service time distribution) and the blocking type. Let μ_i denote the service rate of node i, i.e. $1/\mu_i$ is the average service time. Arrival rate and service rate can be load dependent. Let B_i denote the maximum number of customers admitted at node i, that is, in the queue and in the servers ($B_i = c + s$), $1 \leq i \leq M$. Thus the total number of jobs in node i satisfies the constraint $n_i \leq B_i$. When the queue reaches the finite capacity ($n_i = B_i$) the node is said to be full and blocking arises.

2.2. Blocking types

Various blocking types have been defined to represent different system behaviors. We now recall three of the most commonly used blocking types [3,35,36,40].

- *Blocking after service* (*BAS*). If a job attempts to enter a full capacity queue j upon completion of a service at node i, it is forced to wait in node i server, until the destination node j can be entered. The server of source node i stops processing jobs (it is blocked) until destination node j releases a job. Node i service will be resumed as soon as a departure occurs from node j. At that time the job waiting in node i immediately moves to node j. If more than one node is blocked by the same node j, then a scheduling discipline must be considered to define the unblocking order of the blocked nodes when a departure occurs from node j.
- *Blocking before service* (*BBS*). A job declares its destination node j before it starts receiving service at node i. If at that time node j is full, the service at node i does not start and the server is blocked. If a destination node j becomes full during the service of a job at node i whose destination is j, node

i service is interrupted and the server is blocked. The service of node *i* will be resumed as soon as a departure occurs from node *j*. The destination node of a blocked customer does not change. Two subcategories distinguish whether the server can be used as a service center buffer when the node is blocked: *BBS-SO* (server occupied) and *BBS-SNO* (server is not occupied).

- *Repetitive service blocking* (*RS*). A job upon completion of its service at queue *i* attempts to enter destination queue *j*. If node *j* is full, the job is looped back into the sending queue *i*, where it receives a new independent service according to the service discipline. Two subcategories have been introduced depending on whether the job, after receiving a new service, chooses a new destination node independently of the one that it had selected previously: *RS-RD* (random destination) and *RS-FD* (fixed destination).

Other blocking mechanism due to finite capacity of subnetworks or subnetwork population constraints have been analyzed and compared in [48,49]. Queueing networks with finite capacity queues and blocking can deadlock, depending on the blocking type. Deadlock prevention or detection and resolving techniques must be applied.

Deadlock prevention for closed queueing networks with blocking types BAS, BBS and RS-FD requires that the overall network population N is less than the total buffer capacity of the nodes in each possible cycle in the network. For RS-RD blocking it is sufficient that routing matrix P is irreducible and N is less than the total buffer capacity of the nodes in the network [7,35]. Moreover, to avoid deadlocks for BAS and BBS blocking types we assume $p_{ii} = 0$, $1 \leq i \leq M$.

Deadlock prevention for open queueing networks requires the following additional population constraint for each possible cycle in the network with finite capacity queues and where each node in the cycle has a blocking mechanism different from RS-RD. The population of the subnetwork formed by the nodes of such a cycle has to be less than the total buffer capacity of the nodes in the cycle. In the following we shall consider deadlock-free queueing networks in steady-state conditions.

Blocking mechanisms have been applied to model various types of systems with finite capacity resources in several fields. For example, blocking mechanisms can model specific communication protocols of store-and-forward communication networks. Blocking mechanisms have been extensively applied to production and manufacturing systems. For example, in a manufacturing system when a station completes the processing, it cannot off-load the "part" until the material handling device (MHD) becomes available and arrives at the station. This is known as the so-called interference problem that can be modeled by blocking queueing networks [SUDE97]. Production lines with assembly and disassembly (A/D) stations are often considered to improve the production rate. A station assembles components but the semi-finished product cannot join the next station if this is full, so blocking occurs [18].

Only in the most recent literature, a few papers deal with queueing networks with blocking as performance models in the field of SA, as we discuss with more details in Section 3.

2.3. Analysis of queueing networks with blocking

Under general assumptions a queueing network model with finite capacity can be represented by a Markov process. Let $\mathbf{S} = (\mathbf{S}_1, \ldots, \mathbf{S}_M)$ denote the state of the network and let E be the state space, i.e. the set of all feasible states. The network model evolution can be represented by a continuous-time ergodic Markov chain M with discrete state space E and transition rate matrix Q. The stationary and transient behavior of the network can be analyzed by the underlying Markov process. Under the hypothesis of

an irreducible routing matrix P, there exists a unique steady-state queue length probability distribution $\boldsymbol{\pi} = \{\pi(\mathbf{S}), \mathbf{S} \in E\}$, which can be obtained by solving the homogeneous linear system of the global balance equations

$$\boldsymbol{\pi} Q = \mathbf{0} \tag{1}$$

subject to the normalizing condition $\sum_{\mathbf{S}\in E}\pi(\mathbf{S}) = 1$ and where $\mathbf{0}$ is the all zero vector. The definition of state space E and transition rate matrix Q depends on the network definition and on the blocking type of each node [8,11,35,36,40,48,49]. From vector π one can derive π_i, the queue length distribution of node i and other average performance indices of node i, such as throughput (X_i), the average queue length (L_i) and the mean response time (R_i).

The numerical solution based on the Markov chain analysis is seriously limited by the space and time computational complexity that grows exponentially with the number of model components. When the Markov chain is infinite and unless a special regular structure of matrix Q allows to derive closed form expression of the solution π, one has to approximate the solution on a truncated state space. For closed networks the time computational complexity of liner system (1) is determined by the space state E cardinality that grows exponentially with the buffer sizes ($B_i \leq N$, $1 \leq i \leq M$) and M. Although the state space cardinality of the process can be much smaller than that the process of the same network with infinite capacity queues (which is exponential in N and M), it still remains numerically untractable as the number of model components grows.

In some special cases, queueing networks with blocking have a product-form solution, under particular constraints and for various blocking types. A survey of product-form solutions of networks with blocking and equivalence properties among different blocking network models is presented in [11]. Some efficient algorithms for some closed product-form networks with blocking have been recently defined [10,44]. These algorithms provide the model solution with a time computational complexity linear in the number of model components, i.e., the number of service centers and the number of customers. However, general queueing networks with blocking do not have a product-from solution and approximate analytical methods or simulation have to be applied [3,6,8,9,17,20,28,29,34,35,36], [PERR 98], [40,41].

We consider queueing networks with blocking as a performance model of SAs. In the next section we summarize the framework of performance modeling of SA and in Section 4 we discuss how queueing networks with finite capacity and blocking can be used to model synchronous communications between software components.

3. Performance models of SAs

SA represent high level system description in terms of subsystems (components) and the way they interact (connectors). Usually they provide two type of descriptions: a static one modeling the topology of the system and a dynamic one modeling the way components can interact when the system is in execution. As far as performance measurements are concerned we are mainly interested in the dynamic description, since the way components dynamically behave and interact with each other determines the values of the performance indices. The SA static description, on the contrary, provides information useful to build the QNM performance model. Intuitively a component is the natural candidate to become a service center, although its precise nature and structure will depend on the way the component interacts at execution time with the rest of the system.

By analyzing the SA dynamic behavior it is possible to single out the real degree of concurrency and synchronization among components and this information allows for the generation of meaningful (i.e. faithful with respect to the SA description) QNM performance models.

For example, queueing networks with infinite capacity queues naturally model SA in the two following cases: (i) sequential components with synchronous communication, (ii) concurrent components with asynchronous communication through buffers.

In this case we can obtain simple product-form BCMP networks that can be efficiently analyzed [25,27,30,33]. More precisely, a software component or a set of components can be represented with a simple server or complex server. A complex server represents the service given by the associated components, where the associated service time is the summation of the single service time associated to each component.

On the other hand if we need to model SAs with concurrent components and synchronous communication, since the class of queueing network models with infinite capacity queues is not sufficiently expressive, queueing networks with blocking as performance models can be used.

As discussed in Section 1, only recently queueing network models with blocking have been considered for performance evaluation of SAs [4,5,19,31,32,42]. Some authors consider client–server architectures and propose different blocking mechanisms to model synchronous communication between client and server [19,31,32,42].

In particular, Ramesh and Perros [42] consider a client–server system where clients and servers communicate by means of synchronous and asynchronous messages. The authors propose a multi-layered queueing network model where service times have a Coxian distribution. They consider multiple classes and chains to take into account the different characteristics of the clients and to route a synchronous message in a server back to the originating client. In the model the service at client's queue represents the execution of some segment of software code at the client's CPU. Following the completion of this service, a synchronous or asynchronous message is sent to one of the servers. In the synchronous case the client gets blocked, i.e. it cannot process that request further, nor can it process any other request and it remains blocked until a response is received from the corresponding server. This behavior is modeled by using BAS blocking for the clients' queues. Under these assumptions, the queueing network does not have an exact solution and the authors use a two-node decomposition algorithm and an iterative method to solve the model.

In [31,32], Lladò and Harrison consider a particular client–server system, that is a new component architecture for the development and deployment of object-oriented, distributed, enterprise-level applications. The authors develop an analytical model for the central scheduler of this system, in particular the objective is to analytically model the method call execution process in order to predict its performance. There is a limit to the number of method calls that can be concurrently executing and when this limit is reached the request has to wait until one of the executing methods finishes. The authors use a non-standard form of blocking to model the synchronization among methods.

In [19], Goomaa and Menascé consider the component interconnection in client–server systems, that is the way client and server components communicate with each other. The authors consider synchronous and asynchronous communication and starting from UML description they provide the performance annotation using an XML-type notation. Then, following a SPE-based approach [45,46], this design model is mapped to an open queueing network with a multiple classes and some capacity constraints on subnetwork population to model synchronous communication. The authors apply the decomposition principle to solve the entire network by using a Markov chain model.

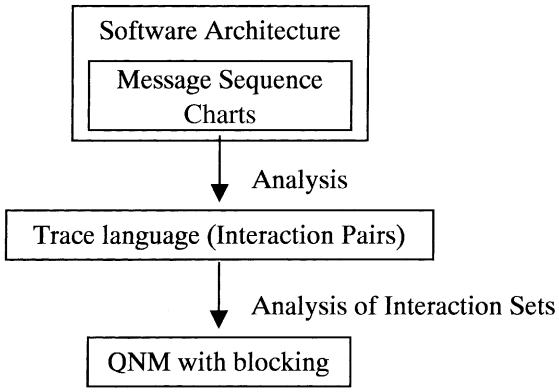

Fig. 1. Scheme of the approach based on MSC.

In the following we briefly summarize our approach to automatically derive a performance evaluation model, based on a queueing network, from a SA dynamic specification.

We start from a formal description of SA that models static and dynamic views. We assume to deal with behavioral models that can be completely described as finite state machines (FSMs) or partially described as MSCs, or sequence diagrams in the UML terminology. In the following we will deal with MSC. We refer to [5] for a detailed description of this method.

MSC have been widely used as a design tool in many event-based distributed systems. Our approach assumes the SA is described by MSC whose objects are components and interactions are modeled as message sequences, that is, how messages are sent and received between a number of objects. We analyze MSC in terms of the trace languages that they generate trying to single out the real degree of parallelism among components and their dynamic dependencies. This information is then used to build a faithful queueing network model representing the SA behavior that forms the basis to conduct software system performance analysis. Fig. 1 illustrates the scheme of this approach.

3.1. MSCs as SA description

MSC illustrate the interactions among a set of objects focusing on message sequences, that is, how messages are sent and received between them [24]. The sequence diagram describes the specific interaction between objects that happens at a certain time during the system's execution. On the horizontal axis there are the objects involved in the sequence, represented by an object rectangle with the object name. The vertical dashed line represents the object's lifeline. Horizontal arrows between object's lifelines represent communications between objects. The end of arrows indicates the type of communication. In our approach the objects are substituted by architectural components and the arrows indicate the time when communication occurs.

We assume that each component is contained in an MSC and the MSC contain state information about each component. MSC can contain repeat cycles, refer to the same initial configuration and must be representative of the major system behavior. In Fig. 2(c) the MSC describes an execution scenario involving four components, User, Router, Server and Timer. The User sends and AlarmUR to the Router then after receiving a message clock from the Timer component, it sends a Check to the Router, after

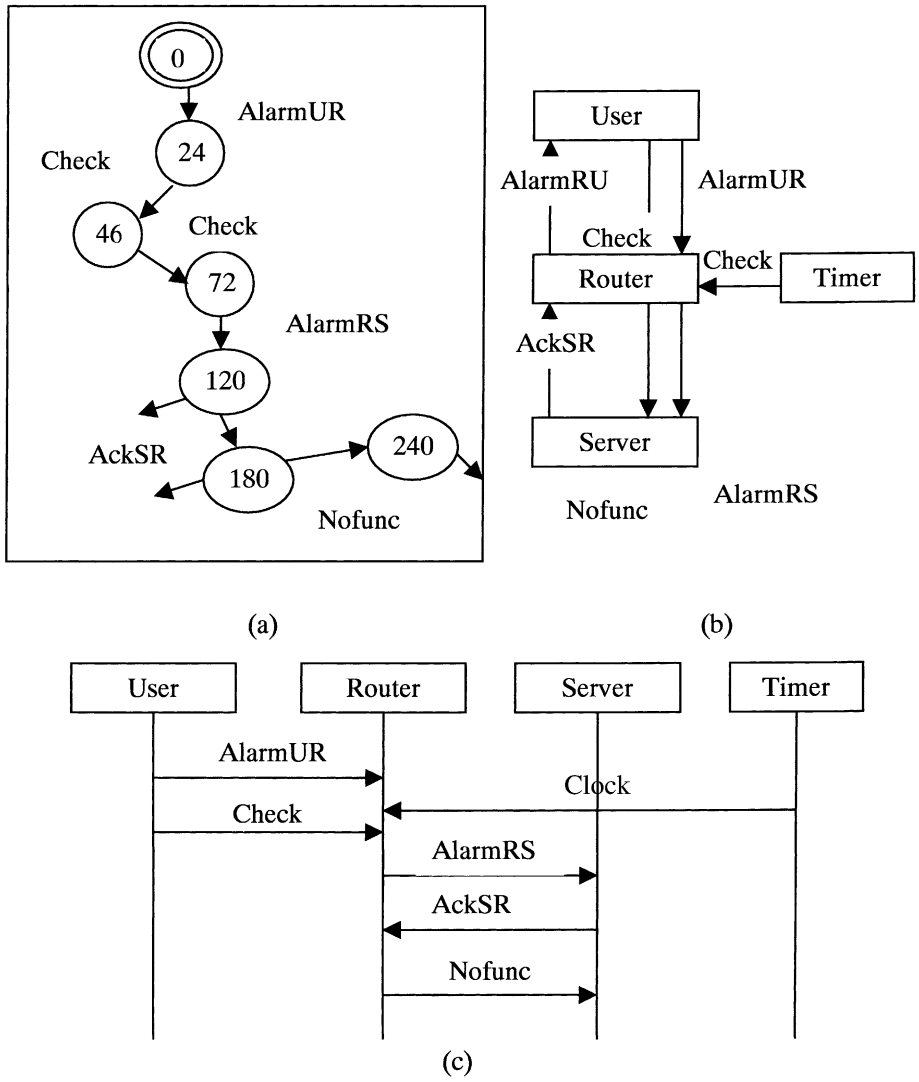

Fig. 2. Dynamic views (a) FSM, (b) static view, (c) MSC.

receiving which the Router forwards the alarm to the Server, AlarmRS, which acknowledges its reception to the Router, AckSR. After that the Router can communicate to the Server that it stopped working, Nofunc.

3.2. Deriving queueing network models from MSCs

Starting from the high level description of SAs based on MSC we derive a queueing network model to evaluate and predict SA performance. We derive dynamic information such as communication among components, communication types, concurrency and non-determinism among components. The approach is based on the derivation from each MSC of a regular expression describing the event sequence performed

in the MSC. Such expressions are analyzed and compared to find out the common maximum prefix. Then we analyze the remainder parts of the regular expressions to identify particular structures that complete the needed information. This information is collected into sets, called *interaction sets*. Hence we can get the parallelism between components.

Let $\mathbf{M} = \{M_1, M_2, \ldots, M_n\}$ denote the set of MSC describing the architectural behavior of the set of components $C_1, \ldots, C_m$. An event e_j on an MSC M_i is a communication between two components, denoted by an arrow. A label $S(C_1, C_2)^c$ is associated to an event e_j where C_1 and C_2 are the sending and receiving components, respectively, and c denotes the communication type, i.e., synchronous ($c = \mathrm{s}$) or asynchronous ($c = \mathrm{a}$).

Schematically we proceed as follows:

1. The set of MSC is analyzed.
   - 1.1. We first produce based interaction sets representing all the communications in the MSC. In particular if we generate a label $S(C_1, C_2)^c$, then we put the interaction pair $(C_1, C_2)^c$ in a set called based interaction set.
   - 1.2. Then by trace analysis we generate the structured interaction sets composed of two interaction pairs and corresponding to concurrent components, one to one, two to two and alternative communications. This trace analysis is based on a comparison of pairs of traces searching a matching among their prefixes. Indeed a trace singles out an automata computation, so a common prefix corresponds to the time in which two computations begin to have distinct behaviors. For a detailed description of the method and the algorithm see [5].
2. In order to derive the queueing network model of the SA described by the MSC, we examine the obtained structured interaction sets to associate elements of the queueing network model.
   - 2.1. We identify internal and external components so determining whether the queueing network is open or closed. External components can be seen as sources that model the production of system customers or processing elements from which the system communicates the results to the environment.
   - 2.2. Then, we analyze interaction among system components, i.e., the interaction sets to understand their real level of concurrency and consequently generate the corresponding queueing network. This is crucial to obtain a model that faithfully represents the given system. The goal is to understand which components are strongly synchronized so they result in a sequential behavior, and which are independent from others and can be concurrently active.
   - 2.3. We start from considering each component as an autonomous server.
   - 2.4. Then along the computation it can become part of a more structured element. Informally, an interaction set $I_i = \{(C_1, C_2)^c\}$ is modeled as a complex server representing a unique service composed of C_1 followed by C_2 and that expresses the sequence of operations. The service center with an infinite buffer implicitly models the (infinite) communication channel.
   - 2.5. Moreover, the algorithm builds the transition from C_1 to C_2 in the network topology, defines external arrivals if C_1 is an external element and a departure if C_2 is an external element.
   - 2.6. The algorithm models a non-deterministic computation by introducing a multi-customers service center that at the end of the generation process is transformed in a simple-customer service center whose service time depends on the service times of the original classes.

The complete definition of the queueing network model also depends on how we represent the type of communication among components. In particular, we observe that by using only service centers with

infinite capacities we cannot model systems with concurrent components that can communicate synchronously. In the next section we deal with queueing networks with finite capacity and blocking to model synchronous communication among software components.

4. Modeling synchronous communication by QNM with finite capacity

In this section we discuss how queueing network with finite capacity queues and blocking can be used as performance models of SA. As discussed in Section 3, some recent approaches apply queueing networks with blocking as software performance models of SA and they consider in particular BAS blocking mechanism to model synchronous communication.

In [42], the authors consider multi-layered queueing networks with finite capacity and BAS blocking to model the synchronous messages from a client to the server. However, this model is based on layered queueing networks (LQN) introduced in [43,50,51] and blocking behavior is modeled in the underlying Markov process by appropriately augmenting the service times at each client to account for the blocking delays. Then the client subsystem is solved as a $C_2/C_2/1$ queue using a matrix geometric algorithm, in the framework of an approximate decomposition method for multiclass tandem queueing networks with blocking based o the solution of two-node networks.

The approach presented in [19] is different. The authors model the synchronous communication by considering a constraint on the requests that the client can send to the server, i.e., they consider a population constraint on the subnetwork that models the server.

Then the model is analyzed by decomposition. The subnetwork that models the server is analyzed in isolation and solved by mean value analysis by varying the population value. This solution is used to define a model for the entire software system architecture, where the server component includes the queue of requests exceeding the population constraint due to synchronization. The solution of the entire model is based on a Markov chain analysis.

The approach presented in [31,32] considers an interesting but specific software system architecture where client–server synchronization is based on remote method invocations. Blocking arises because of the finite number of method calls that can be concurrently executing and a non-standard form of blocking is considered in the queueing network. The approach is based on decomposition and approximate aggregation method to evaluate the blocking time due to the finite concurrency level. They apply the flow equivalent server method [30,33] to analyze the system model and calculate the blocking time. In particular, the solution of the extended queueing network is obtained by aggregating subnetworks into single nodes with queue length dependent service rates that capture the essence of the blocking effects.

The approach proposed in [5] and recalled in the previous section is based on the observation that queueing networks with infinite capacity queues are not sufficiently expressive to model systems with synchronous communication between concurrent components. QNM with finite capacity and blocking can capture such feature of the communication systems and represent some synchronization constraints. To model synchronous communication among concurrent system components, we assign distinct service centers to the communicating components in order to model their independence. We associate to the receiver component a service center with a zero capacity buffer and impose a BAS blocking mechanism to the sender component in order to model synchronization.

Specifically we model the components that can receive a synchronous communication with a finite capacity service center, with one server and where the buffer capacity is set to one. That is, we model a

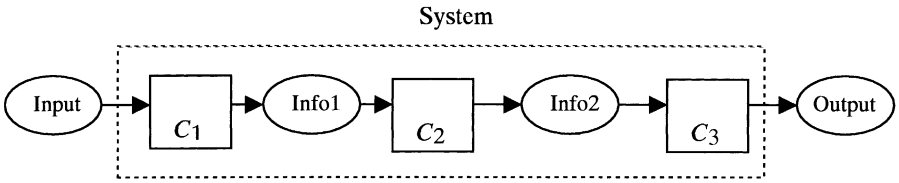

Fig. 3. Concurrent system with synchronous communication.

component C_i with a single server service center i, where we allow no queueing by setting $B_i = 1$ as the queue capacity, i.e., the maximum number of customers admitted at service center i, that is, in the queue and in the server.

We assume BAS blocking as the mechanism for the sending nodes of the finite capacity service center. We choose to model such components as service centers to exploit component concurrency in the SA. Finite capacity queues and BAS blocking mechanism allow to model synchronous communication.

For example, consider a system that consists of three components C_1, C_2 and C_3 with synchronous communication, as illustrated in Fig. 3. Assume that all the components can be active at the same time.

Our goal is to define a performance model that represents the parallel execution of the three components and also synchronous communication. For example, the model must represent the case where component C_1 tries to communicate with component C_2 which is active on some operations; then component C_1 becomes blocked until component C_2 is free and can receive information from C_1. Modeling this synchronous communication between the two concurrent components is illustrated in Fig. 4.

In particular, we have synchronous communication between the two components C_1 and C_2. We model this communication pattern by associating a service center to each component, say SC(C_1) and SC(C_2), respectively. Then we associate to SC(C_2) a single capacity queue and to SC(C_1) the BAS blocking mechanism. Thanks to this kind of modeling we can describe more complex contexts where the components C_1 and C_2 are simultaneously active but also situations in which C_1 attempts to communicate with C_2, when the latter is still working. In fact the BAS mechanism permits to block the component C_1 waiting for C_2 to complete its service.

By setting in SC(C_2) finite capacity $B_2 = 1$, then it can receive service requests from SC(C_1) only when its server is not occupied. When SC(C_2) is full, if SC(C_1) at the completion of its service attempts to send a customer (a request) to SC(C_2), then SC(C_1) is blocked until a departure (service completion) occurs from SC(C_2), according to BAS definition. This corresponds to the system behavior that we want to represent in the performance model.

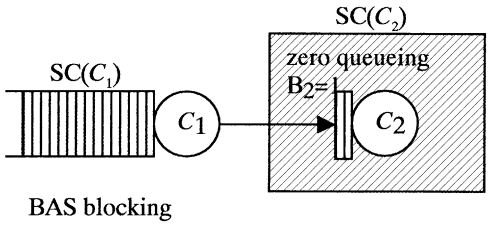

Fig. 4. Modeling synchronous communication between concurrent components.

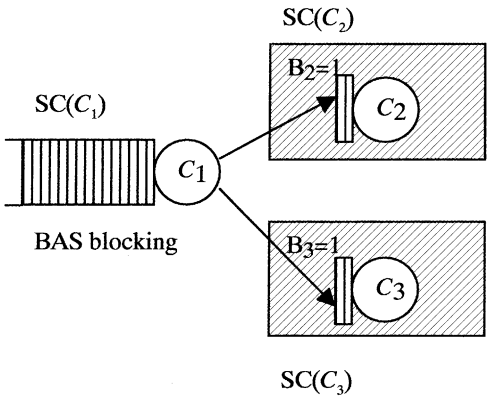

Fig. 5. Modeling one to two synchronous communication between concurrent components.

Therefore, one to two $\{(C_1, C_2)^{c1}, (C_1, C_3)^{c2}\}$ and two to one $\{(C_2, C_1)^{c1}, (C_3, C_1)^{c2}\}$ communications are modeled assigning to the involved components distinct service centers, with a single capacity queue if the communication is synchronous. One to two communication model is for synchronous communication, i.e. with $c1 = c2 =$ s, illustrated in Fig. 5. Service center SC(C_1) can have in turn finite capacity if C_1 is also a destination component of a synchronous communication.

In two to one communication $\{(C_2, C_1)^{c1}, (C_3, C_1)^{c2}\}$ for synchronous communication we have to assume a scheduling of the communication requests (the customers in the queueing network) arriving at C_1 from C_2 and C_3. This corresponds to the definition of the unblocking scheduling in BAS blocking definition. First blocked first unblocked scheduling corresponds to maintain the order of communication request times. Fig. 6 illustrates two to one communication model for synchronous communication, i.e., with $c1 = c2 =$ s.

Service centers SC(C_2) and SC(C_3) can have in turn finite capacity if C_2 and C_3 are also destination components of a synchronous communication, respectively. A different case is the two to one communication $\{(C_2, C_1)^{c1}, (C_3, C_1)^{c2}\}$ where there is a synchronous and an asynchronous communication. For example, assume that components C_1 and C_3 are active components with synchronous communication (i.e. $c2 =$ s) and C_2 is a connection element for some other component. Then SC(C_2) has infinite

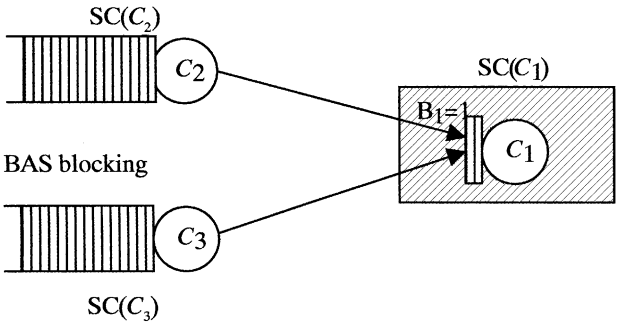

Fig. 6. Modeling two to one synchronous communication between concurrent components.

capacity queue because it models the buffer for asynchronous communication from an other component to component C_1 (i.e. $c1 = \mathrm{a}$). Hence, the customers waiting in the infinite queue of SC(C_2) model the communication requests arriving through connection component C_2. This server of this service center can be blocked by SC(C_1) when the queue is full, and this represents the attempt to send a request to a busy destination component.

Finally note that when both components C_2 and C_3 are connection elements, we have asynchronous communication to component C_1 (i.e. $c1 = c2 = \mathrm{a}$) that can be modeled by an infinite capacity service center SC(C_1). A similar case is when component C_1 is a connection element.

Hence, with this modeling approach we can complete the definition of the queueing network model as concerns the communication among components. We introduce finite capacity service centers with single queue capacity and BAS blocking only for those components that use synchronous communication.

The algorithm to derive the queueing network from the MSC trace analysis identifies the possible service centers by considering components interactions by the interaction sets, and defines the corresponding service center according to the communication type between components.

When all the interaction sets have been examined, the algorithm proceeds by performing several merging operations to reach the final configuration of the service centers. Specifically, to consider concurrent components we define a set CONC of pairs of independent execution elements. This set is used eventually to reduce the number of service centers, by deleting the useless ones. The idea is to analyze set CONC to verify whether for each service center SC(C_i) there is a concurrent component which communicate in synchronous way. This analysis is carried out by considering the complex servers and multiclass servers. If we observe that a service center with finite capacity represents a component that is not concurrent with other components, then that service center can be eliminated in the queueing network model.

4.1. Model solution

From the algorithm that we have sketched in the previous section to derive a queueing network with BAS blocking from a SA description based on MSC, we do not obtain a completely specified queueing network model, because we only perform a functional analysis of the system.

In order to solve the performance model we still have to perform the parameterization step of the modeling process. The parameters to be defined are the distributions characterizing the service times, the customer arrival process for every service center and the network routing probability.

The complete specification of the queueing network has to be done by the designer according to the system requirements and by considering the specific class of models. It is in fact important to select the quantitative parameters so that the resulting queueing network belongs to a class that allows an efficient solution method.

Only few solution methods have been proposed for multiclass queueing networks with blocking. QNM with RS blocking and multiple classes of customers can be analyzed by approximate solution methods based on decomposition and the maximum entropy analysis [6,28]. Then to simplify model solution, single class QNM with blocking can be derived and analyzed. Moreover, we observe that exact analysis can be applied in particular cases and approximate solution is often necessary [7,8,35,36,40]. In particular, the queueing network can be solved via the numerical solution of the underlying Markov chain, i.e., by solving linear system (1). However, this approach is possible only for small networks, since the space state cardinality grows exponentially with the buffer sizes ($B_i \leq N$, $1 \leq i \leq M$) and the number of service centers, M.

For some special cases, queueing networks with blocking have a product-form solution, under particular constraints depending on the blocking type, the network topology and other network parameters. For example, for BAS blocking a product-form solution holds for two-node networks with BCMP-type service centers, multiple types of customers and class independent capacities. Another case of product-form networks with BAS blocking is for arbitrary topology networks whose service centers have FIFO service discipline and exponential service time, multiple types of customers, class independent capacities, and under the additional constraint that at most one node can be blocked at a time. Details and references can be found in [8,11].

Several authors have proposed various approximate solution methods for queueing networks with blocking [3,6,8,15,17,20,28,34,35,39,40,41]. They usually provide average performance indices, such as throughput, mean queue length and mean response time and they can be applied under various assumptions. For example, approximation methods to evaluate the network throughput of arbitrary topology closed networks with BAS blocking and exponential service times have been proposed in [1,2]. Several methods can be applied for open networks with blocking, depending on the topology, mostly based on the decomposition principle as in [15,17,20,29,34,41].

Hence the complete specification of the queueing network with the parameter selection should take into account also special constraints that allow to apply appropriate solution methods to evaluate performance parameters of the complete queueing network model.

We can carry out performance prediction and analysis of various potential implementation scenarios by defining the parameters and solving the queuing network model. Such performance analysis can provide useful insights for the software development process in order to meet some given performance criteria.

5. Examples

In this section we illustrate the resulting performance models, the queueing networks with finite capacity obtained by the application of our approach to two simple examples of SAs.

The first example is the design of a compressing proxy system as described in [16]. Such system is introduced with the purpose of improving the performance of Unix-based World Wide Web browsers over slow networks by an HTTP server that compresses and uncompresses data to and from the network. We have four components as illustrated in Fig. 7. Components are denoted by square boxes and processes by ovals. The filters communicate using a function-call-based stream interface.

A filter F is said to read data whenever the previous filter in the series invokes the proper interface function in F. The interface also provides a function to close the stream. The gzip program is also a filter,

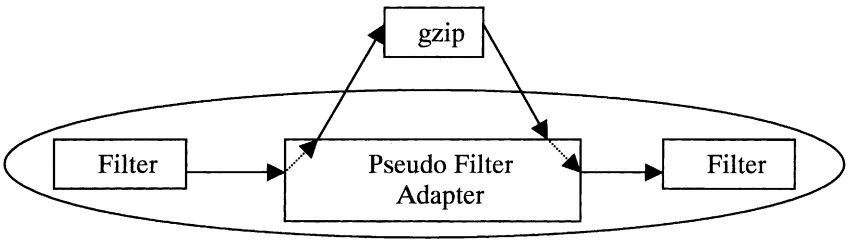

Fig. 7. The compressing proxy SA.

but at the level of a UNIX process, so using the standard UNIX input/output interface. Communication with gzip occurs through UNIX pipes. An important difference between UNIX filters, such as gzip and the HTTP filters is that the formers explicitly choose when to read, whereas the latter are forced to read when data are pushed at them. To assemble the compressing proxy from the existing HTTP server and gzip without modification, we must create an adapter. This acts as a pseudo HTTP filter, communicating with the upstream and downstream filters through a function–call interface, and with gzip using pipes connected to a separate gzip process that it creates.

From the MSC describing the SA we can obtain a queueing network model that represents the compressing proxy system. Details on MSC trace analysis is out of the scope of this paper and can be found in [5]. We assume synchronous communication. By applying the algorithm we partition the four components into internal and external elements. From the MSC we derive the traces whose analysis generates the interaction sets. Fig. 8 illustrates an example of MSC that gives rise to the following trace $\{S(Cf_u,AD)S(AD,Gzip)S(Gzip,AD)S(AD,CF_d)\}^N$.

By considering the modeling approach presented in the previous section for the communication system we eventually obtain the queueing network model with finite capacity and BAS blocking illustrated in Fig. 9.

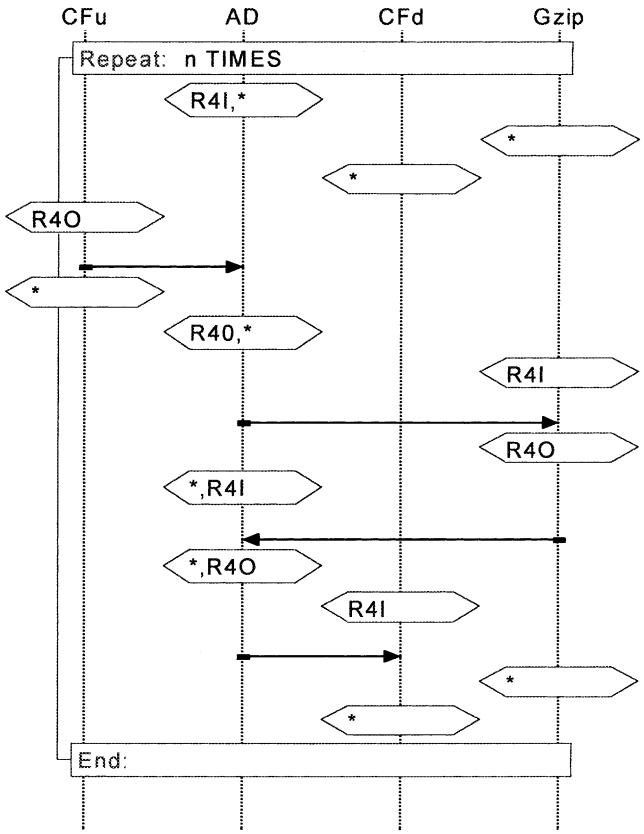

Fig. 8. Example of an MSC of the compressing proxy system.

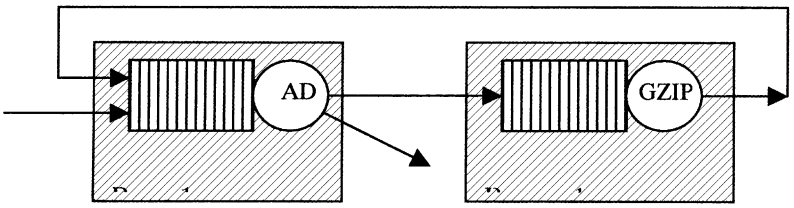

Fig. 9. The queueing network model of the compressing proxy system.

The model is an open two node network with finite capacity queues $B_{AD} = 1$ and $B_{GZIP} = 1$. We have external arrivals and departures only from the AD service center (adapter). We assume FIFO service discipline.

The service center AD represents components adapter and filter, the service center GZIP represents component gzip of the SA. Such service centers are derived by the analysis of the interaction sets, by applying the modeling approach described in Section 3.

Specifically, from the analysis we obtain the tuple [FILTER1, AD, GZIP, AD, FILTER2] that characterizes the service and it states that a customer requires services to the processing elements in that order. The first element of the tuple is an external element because the network is open.

Synchronous communication between adapter and gzip is modeled by the finite capacity service center $B_{GZIP} = 1$ and BAS blocking for the sender service center AD. Similarly, the synchronous communication between the two concurrent components gzip to adapter, i.e., from the former to the latter, leads to the finite capacity service center $B_{AD} = 1$ and BAS blocking for server GZIP. The network routing chain definition derives from the component interactions.

A customer arrival at the network represents the arrival of data at the first filter in the SA. The queueing network model represents at the architectural level the interaction and potential concurrency of the compressing proxy SA.

Hence we obtain a simple two node queueing network model with finite capacity and BAS blocking that can be solved with exact analysis based on the underlying Markov process to derive the steady-state joint queue length distribution from which one can evaluate a set average performance indices. In particular, under the constraint of exponential service time distribution, a closed-form solution of the stationary probabilities can be derived as discussed in Section 4.1 [1,8].

We can solve such queueing network model for different values of the network parameters, so comparing and predicting the performance of the compressing proxy system under various scenarios. This can provide useful insights on the design process development as concerns the meeting of quantitative performance requirements.

The second example is the well-known multiphase compiler architectures [23,38]. The data elements are characters, tokens, phrases, correlated phrases (i.e., phrases signifying name uses related to phrases signifying name declarations) and object code. The processing elements are the text (i.e., the producer of source characters), Lexer (i.e., lexical analyzer), the Parser, the Semantor (i.e., semantic analyzer), and the code generator. An optional processing element is the Optimizer. The processing elements run their phases opportunistically and in parallel via concurrent access to a shared repository, so that, for example, the Semantor can correlate phrases at the same time as the Lexer is creating new tokens. This SA example has been described in [4] where we apply the algorithm to derive the performance model at the architectural level, from a labeled transition system description, and to compare different architectures.

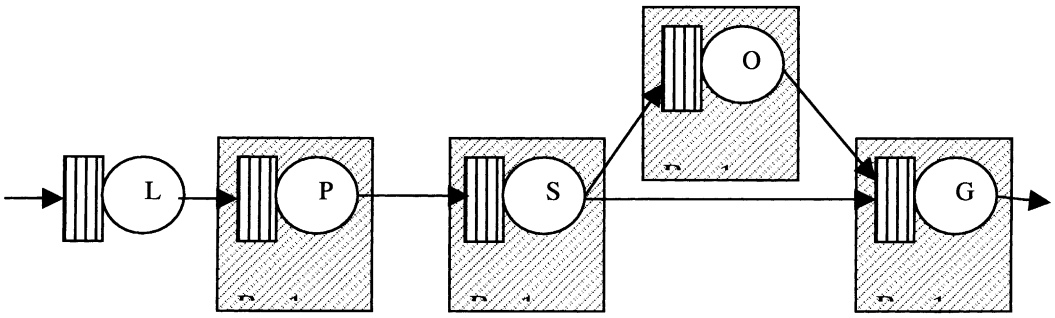

Fig. 10. The queueing network model of the concurrent multiphase compiler SA.

In particular when we consider synchronous communication between concurrent components in the multiphase compiler architecture, we can apply the modeling approach presented in the previous section. From the algorithm to derive the interaction sets and then from the trace analysis we eventually obtain the queueing network illustrated in Fig. 10.

The open queueing network is formed by five service centers and has acyclic topology. The first service center is denoted by L and represents the Lexer component, and similarly the remaining service centers denoted by P (Parser), S (Semantor), O (Optimizer) and G (Code_gen) correspond to system components. Service center L has infinite capacity queue and every other service center has finite capacity queue $B_i = 1$, $i = \mathrm{P}, \mathrm{S}, \mathrm{O}, \mathrm{G}$. A customer arrival at the network in node L represents the arrival of a new text to be compiled in the SA.

According to the modeling approach described in Section 3, synchronous communication from Lexer to Parser is modeled by finite capacity $B_{\mathrm{P}} = 1$ and BAS blocking in service center L. Similarly, synchronous concurrent communication between Parser and Semantor leads to the definition of finite capacity $B_{\mathrm{S}} = 1$ and BAS blocking for the sender service center P. The interaction pairs $\{(\text{Semantor}, \text{Optimizer})^{\mathrm{s}}, (\text{Semantor}, \text{Code\_gen})^{\mathrm{s}}\}$ corresponding to one to two synchronous communication is modeled by the distinct service centers $\mathrm{S} = \mathrm{SC}(\text{Semantor})$, $\mathrm{O} = \mathrm{SC}(\text{Optimizer})$ and $\mathrm{G} = \mathrm{SC}(\text{Code\_gen})$ with finite capacities $B_{\mathrm{O}} = 1$ and $B_{\mathrm{G}} = 1$ and BAS blocking in S as shown in Fig. 3. Finally, the two to one synchronous communication $\{(\text{Semantor}, \text{Code\_gen})^{\mathrm{s}}, (\text{Optimizer}, \text{Code\_gen})^{\mathrm{s}}\}$ is represented by the same service centers by assuming the additional constraint of finite capacity queue of service center G, i.e., $B_{\mathrm{G}} = 1$ and BAS blocking for the two sending nodes S and O as shown in Fig. 4. Moreover, in this case the scheduling unblocking discipline of BAS blocking for the two sending nodes S and O represents the scheduling of the communication requests arriving to the Code_gen from Semantor and Optimizer. For example, by the first blocked first unblocked discipline we model the policy that keeps the order of the communication request times from the two components.

By combining the various constraints on the queue capacities we obtain the open five-node queueing network model with finite capacity and BAS blocking shown in Fig. 7. It can be solved with exact analysis based on the underlying Markov process to derive the steady-state joint queue length distribution by system (1), and then compute average performance indices [8,39,40]. This can be an appropriate approach because of the small finite queue capacities. Since the network does not have a product-form solution, we can also consider an approximate method to derive average performance indices as recalled in Section 4.1. For example, we can analyze the network with BAS blocking by an algorithm based on

the decomposition approach such as the approximate algorithm described in [34] for acyclic networks that evaluates the network throughput and other average performance indices.

Similarly to the previous example, we can solve such queueing network model for different values of the network parameters, so comparing and predicting the performance, at the architectural level, of the multiphase compiler SA under various scenarios.

6. Conclusions

We have presented the application queueing network models with finite capacity queues and blocking as performance models of SAs. We have discussed various approaches recently proposed in literature that use this kind of models. In the framework of a methodology for performance evaluation of SAs we have then considered the problem of defining an appropriate modeling for the communication system.

Starting from a high level description of a SA by the MSCs we obtain a trace analysis and we derive a queueing network as a performance model. Since QNM with infinite capacities are not sufficiently expressive to model systems where there are concurrent components that can communicate synchronously, we observed the need of a more accurate definition of the performance models of SAs to capture some features of the communication systems. QNM with finite and single capacity queues and BAS blocking can be used to represent some synchronization constraints. We have discussed how this class of models can be used to represent synchronous communication between concurrent components.

We have shown two examples of the application of the methodology that leads to the definition of queueing network models with BAS blocking, whose solution can be obtained by known algorithms. This work represents a step toward the construction of a flexible environment for performance evaluation of SAs.

Future work in the filed of software performance models should focus in identifying appropriate performance models that allow an accurate description of the specific mechanisms and relevant features of SA. However, it is important to select models that can be efficiently solved and for which tools and solution methodologies are developed and available.

Acknowledgements

The authors have been supported by MURST Project Research Fund (Progetto MURST di rilevante interesse nazionale), Saladin. http://www.saladin.dm.univaq.it. The authors would like to thank Federico Andolfi and Federica Aquilani for helpful discussions.

References

[1] I.F. Akyildiz, On the exact and approximate throughput analysis of closed queueing networks with blocking, IEEE Trans. Soft. Eng. 14 (1988) 62–71.
[2] I.F. Akyildiz, Mean value analysis of blocking queueing networks, IEEE Trans. Soft. Eng. 14 (1988) 418–429.
[3] I.F. Akyildiz, H.G. Perros, Special issue on queueing networks with finite capacity queues, Perform. Eval. 10 (3) (1989).
[4] F. Aquilani, S. Balsamo, P. Inverardi, Performance analysis at the software architectural design level, Perform. Eval. 45 (2001) 147–178.
[5] F. Andolfi, F. Aquilani, S. Balsamo, P. Inverardi, Deriving performance models of software architectures from message sequence charts, in: ACM Proceedings of the WOSP 2000, Second International Workshop on Software and Performance, Ottawa, Canada, September 17–20, 2000.

[6] I.U. Awan, D.D. Kouvatsos, Approximate analysis of QNMs with space and service priorities, in: D.D. Kouvatsos (Ed.), Performance Analysis of ATM Networks, Kluwer Academic Publishers, IFIP Publication, Chapter 25, 1999, pp. 497–521.
[7] S. Balsamo, Properties and analysis of queueing network models with finite capacities, in: Performance Evaluation of Computer and Communication Systems, Lecture Notes in Computer Science, Springer, Berlin, 1994, p. 729.
[8] S. Balsamo, V. De Nitto Personè, R. Onvural, Analysis of Queueuing Networks with Blocking, Kluwer Academic Publishers, Dordrecht, 2001.
[9] S. Balsamo, A. Rainero, Closed queueing networks with finite capacity queues: approximate analysis, in: Proceedings of the ESM 2000, SCS European Simulation Multiconference, Ghent (B), May 22–26, 2000.
[10] S. Balsamo, C. Clò, A convolution algorithm for product form queueing networks with blocking, Ann. Oper. Res. 79 (1998) 97–117.
[11] S. Balsamo, V. De Nitto Personè, A survey of product-form queueing networks with blocking and their equivalences, Ann. Oper. Res. 48 (1994) 31–61.
[12] S. Balsamo, P. Inverardi, C. Mangano, Performance evaluation of software architectures, in: ACM Proceedings of the WOSP, Santa Fe, New Mexico, 1998.
[13] L. Bass, P. Clements, R. Kazman, Analyzing Development Qualities at the Architectural Level. Software Architecture in Practice, SEI Series in Software Engineering, Addison-Wesley, Reading, MA, 1998.
[14] J. Bosh, Design and Use of Software Architecture, Addison-Wesley, Reading, MA, 2000.
[15] A. Brandwajn, Y.L. Jow, An approximation method for tandem queueing systems with blocking, Oper. Res. 1 (1988) 73–83.
[16] D. Compare, P. Inverardi, A.L. Wolf, Uncovering architectural mismatch in component behavior, Sci. Comput. Program. 33 (2) (1999) 101–131.
[17] Y. Dallery, Y. Frein, On decomposition methods for tandem queueing networks with blocking, Oper. Res. 14 (1993) 386–399.
[18] Y. Dallery, S.B. Gershwin, Manufacturing flow line systems: a review of models and analytical results, Queueing Syst. 12 (1992) 3–94.
[19] H. Goomaa, D.A. Menascé, Design and performance modeling of component interconnection patterns for distributed software architectures, in: Proceedings of the WOSP 2000, ACM Second International Workshop on Software and Performance, Ottawa, Canada, September 17–20, 2000.
[20] F.S. Hillier, R.W. Boling, Finite queues in series with exponential or Erlang service times—a numerical approach, Oper. Res. 15 (1967) 286–303.
[21] C. Hofmeister, R.L. Nord, D. Soni, Describing software architecture with UML, in: Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), San Antonio, Texas, Berlin, Germany, February 22–24, 1999, pp. 145–159.
[22] C. Hofmeister, R.L. Nord, D. Soni, Applied Software Architecture, Addison-Wesley, Reading, MA, 1999.
[23] P. Inverardi, A.L. Wolf, Formal specification and analysis of software architectures using the chemical abstract machine model, IEEE Trans. Soft. Eng. 21 (4) (1995) 373–386.
[24] ITU-TS Recommendation Z.120, Message Sequence Charts, ITU-TS, Geneva, 1994.
[25] K. Kant, Introduction to Computer System Performance Evaluation, McGraw-Hill, New York, 1992.
[26] R. Kazman, M. Klein, P. Clements, Queueing Systems, Wiley, New York, 1976;
ATAM: Method for Architecture Evaluation, SEI Carnegie Mellon University, August 2000.
[27] L. Kleinrock, Queueing Systems, Wiley, New York, 1976.
[28] D.D. Kouvatsos, I.U. Awan, MEM for arbitrary closed queueing networks with R–S blocking and multiple job classes, Special Issue on Queueing Networks with Blocking, vol. 79, Baltzer Science Publishers, 1998, pp. 231–269.
[29] D.D. Kouvatsos, N.P. Xenios, MEM for arbitrary queueing networks with multiple general servers and repetitive-service blocking, Perform. Eval. 10 (1989) 106–195.
[30] S.S. Lavenberg, Computer Performance Modeling Handbook, Prentice-Hall, Englewood Cliffs, NJ, 1983.
[31] C. Lladò, P. Harrison, A new blocking problem from Java-based schedulers, in: Proceedings of the QNET 2000, Fourth International Workshop on Queueing Networks with Finite Capacity, Ilkley, UK, July 20–21, 2001.
[32] C. Lladò, P. Harrison, Performance evaluation of an enterprise Javabean server implementation, in: ACM Proceedings of the WOSP 2000, Second International Workshop on Software and Performance, Ottawa, Canada, September 17–20, 2000.
[33] E.D. Lazowska, J. Zahorjan, G. Scott Graham, K.C. Sevcik, Quantitative System Performance: Computer System Analysis Using Queueing Network Models, Prentice-Hall, Englewood Cliffs, NJ, 1984.
[34] H.S. Lee, A. Bouhchouch, Y. Dallery, Y. Frein, Performance evaluation of open queueing networks with arbitrary configurations and finite buffers, in: Proceedings of the Third International Workshop on Queueing Networks with Finite Capacity, Bradford, UK, July 6–7, 1995.

[35] R.O. Onvural, Survey of closed queueing networks with blocking, ACM Comput. Surv. 22 (2) (1990) 83–121.
[36] R.O. Onvural, Special issue on queueing networks with finite capacity, Perform. Eval. 17 (3) (1993).
[37] Special issue on performance validation of software systems, Perform. Eval. 45 (2001).
[38] D.E. Perry, A.L. Wolf, Foundations for the study of software architecture, ACM SigSoft Soft. Eng. Notes 17 (4) (1992) 40–52.
[39] H.G. Perros, Open queueing networks with blocking, in: Takagi (Ed.), Stochastic Analysis of Computer and Communications Systems, North-Holland, Amsterdam, 1989.
[40] H.G. Perros, Queueing Networks with Blocking, Oxford University Press, Oxford, 1994.
[41] H.G. Perros, T. Altiok, Approximate analysis of open networks of queues with blocking: tandem configurations, IEEE Trans. Soft. Eng. 12 (1986) 450–461.
[42] S. Ramesh, H. Perros, A multilayer client–server queueing network model with synchronous and asynchronous messages, IEEE Trans. Soft. Eng. 26 (11) (2000) 1086–1100.
[43] J.A. Rolia, K.C. Sevcik, The method of layers, IEEE Trans. Soft. Eng. 21 (1995) 689–700.
[44] M. Sereno, Mean value analysis (MVA) of product form solution queueing networks with repetitive service blocking, Perform. Eval. 36–37 (1999) 19–33.
[45] C.U. Smith, Performance Engineering of Software Systems, Addison-Wesley, Reading, MA, 1990.
[46] C.U. Smith, L.G. Williams, Software performance engineering: a case study including performance comparison with design alternatives, IEEE Trans. Soft. Eng. 19 (7) (1993) 720–741.
[47] M. Shaw, D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, Englewood Cliffs, NJ, 1996.
[48] N. van Dijk, On 'stop = repeat' servicing for non-exponential queueing networks with blocking, J. Appl. Prob. 28 (1991) 159–173.
[49] N. van Dijk, 'Stop = recirculate' for exponential product form queueing networks with departure blocking, Oper. Res. Lett. 10 (1991) 343–351.
[50] M. Woodside, Throughput calculation for basic stochastic randevouz networks, Perform. Eval. 9 (1989) 143–160.
[51] M. Woodside, J.E. Neilson, D.C. Petriu, S. Majumdar, Stochastic randevouz network model for performance of synchronous client–server-like distributed software, IEEE Trans. Comput. 44 (1995) 20–34.
[52] Proceedings of the ACM First International Workshop on Software and Performance, WOSP'98, Santa Fe, New Mexico, USA, October 12–16, 1998.
[53] Proceedings of the ACM Second International Workshop on Software and Performance, WOSP 2000, Ottawa, Canada, September 17–20, 2000.