On using Queueing Network Models with finite capacity queues for Software Architectures performance prediction

F. Andolfi*, F. Aquilani*, S. Balsamo**, P.Inverardi*

* Dip. di Matematica Pura e Applicata, University of L'Aquila, Italy
** Dip. di Informatica, University of Venezia, Italy

Abstract

We are interested in studying the performance of software system in early stages of development. We investigate how queueing networks with finite capacity and blocking can be applied as performance models of software architectures.
The starting point of our analysis is a software system high-level description whose dynamic behavior results in a finite state model or, as recently proposed, is described in terms of Message Sequence Charts. We use the queuing theory to obtain a performance model of the software component behaviour from the behavioral description. Our approach is to automatically derive a queuing network from a Software Architecture description. More precisely, we represent a software component or a set of components 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. For particular behavioral patterns we use also multiclass server that represents a server that can provide multiple service but only one at a time. In the examples we considered this class of queuing network models was not sufficiently expressive. In fact by using only service centers with infinite capacities we cannot model systems where there are concurrent components that can communicate synchronously. Hence, by observing the need of a more accurate definition of the performance models of software architectures to capture some features of the communication systems, we consider queueing networks with finite capacity and blocking to represent some synchronization constraints.
To model synchronous communication among concurrent system components, we assigns 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 blocking mechanism to the sender component in order to model synchronization. In particular we choose the Blocking After Service (BAS) blocking mechanism. Thanks to this kind of modeling we can describe more complex contexts where the components are simultaneously active but also situations in which a component C_1 attempts to communicate with another component C_2, when the latter is still working. Indeed, the BAS mechanism permits to block component C_1 waiting for C_2 to complete its service. In the paper we will discuss our experience in using queueing network models and their adequacy for the problem at hand in terms both of modeling and of its evaluation.

1. Introduction

Queueing network models have been extensively applied in the last decades as a powerful tool for modelling and performance evaluation and prediction of computer and systems, as well as production and manufacturing systems [Kl76, La82, LZGS84, Ka92]. 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 [AP89, Ba94, On90, On93, Pe89, Pe94].
More recently, 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 of software systems. Several approaches have been proposed, notably the software performance engineering method introduced by Smith as an integration of software engineering and performance management [Sm90, SW93, Wo98].
Software Architectures (SA) describe software system structures at a high level of abstraction [SG96] and they have been devised as the appropriate design level to carry out quantitative analysis [BCK98, HNS99, Wo98]. They represent at an early stage of development the phase in which basic

choices of components and interactions among components are made. This choice is driven by several non-functional issues, which include important factors such as system performance and reliability. We are interested in studying and evaluating the expected performance of software architectures. 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 specification formally described in Chemical Abstract Machine formalism [BIM98, ABI00]. From this description we derive a finite state model of the global system behavior whose analysis lead to the definition of queuing network model. More recently we have proposed a new approach based on Message Sequence Charts as the software system high-level description from which we derive the performance model [AABI00]. This approach tries to overcome a drawback of the previous one, that is the state space explosion problem of the finite state model of the SA description.
We use queueing networks to obtain a performance model of the software component behaviour from the behavioral description. From our perspective of software architectures, 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,

In the paper we will discuss our experience in using queueing network models and their adequacy for the problem at hand in terms both of modeling and of its evaluation. Specifically, we observe the need of a more accurate definition of the performance models of software architectures to capture some features of the communication systems. We consider queueing networks with finite capacity and blocking to represent some synchronization constraints.
First we have considered queueing network models with infinite capacity queues to obtain simple product form BCMP networks that can be efficiently analyzed [Kl76, La82, LZGS84, Ka92]. In particular queueing networks with infinite capacity can model SA in the two following cases: (i) sequential components with synchronous communication, (ii) concurrent components with asynchronous communication through buffers.
More precisely, we represent a software component or a set of components 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.
In the examples we considered, the class of queuing network models with infinite capacity queues was not sufficiently expressive. In fact by using only service centers with infinite capacities we cannot model systems with concurrent components that can communicate synchronously.
Hence we investigate the application of queueing networks with blocking as performance models of software architectures with concurrent components and synchronous communication.

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 briefly summarizes the approach of software performance analysis based on queueing network models derived by Message Sequence Charts 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 [AP89, On90, Pe94, VD91a, VD9b]. 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, On90, P94]. 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 [BD94].

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, $1 \le i \le M$, which is usually assumed to be Poisson. Let λ denote total arrival rate at the network and p_{0i}, $1 \le i \le M$, the probability that an exogenous arrival tries to enter node i. Then the Poisson arrival process at node i has parameter λp_{0i}. Let $P=[p_{ij}]$ $(1 \le i,j \le M)$ 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. According to Kendall's notation a service center is denoted by A/B/s/c/D, where A and B are respectively the customer interarrival time and the service time distribution, s the number of identical servers, c the queue capacity and D the service discipline. Examples of service distributions are exponential (M), phase-type with n exponential stages (PH_n), general (G) and generalized exponential distribution (GE). 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 ($B_i=c+s$), $1 \le i \le M$. Thus the total number of jobs in node i satisfies the constraint $n_i \le B_i$. 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 [AP89, On90, On93, Pe94].

- *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).

Closed 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 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, whereas 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 [On90, Ba94]. Moreover, to avoid deadlocks for BAS and BBS blocking types we assume $p_{ii}=0$, $1 \le i \le M$. In the following we shall consider deadlock-free queueing networks in steady-state conditions.

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 behaviour 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 $\pi = \{\pi(\mathbf{S}),\ \mathbf{S} \in E\}$, which can be obtained by solving the homogeneous linear system of the global balance equations

$$\pi Q = \mathbf{0} \qquad (1)$$

subject to the normalising condition $\Sigma_{\mathbf{S} \in E} \pi(\mathbf{S}) = 1$ 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 [BD94, On90, On93, Pe94, VD91a, VD91b]. 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 of the Markov chain analysis is seriously limited by the space and time computational complexity that grows exponentially with the model number of 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 ($B_i \le N$, $1 \le i \le M$) 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 [BD94]. Some efficient algorithms for some closed product-form networks with blocking have been recently defined [BC95]. These algorithms provide the model solution with a time computational complexity linear in the number of network 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 [AP89, BR00, On90, On93, Pe98, Pe94].

We consider queueing networks with blocking as a performance model of software architectures. In the next section we summarize the framework of performance modelling 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 Software Architectures

We consider software performance analysis based on queueing network models derived by Message Sequence Charts as software architecture description. In this section we briefly summarize an

approach to automatically derive a performance evaluation model, based on a queueing network model, from a SA specification described by Message Sequence Charts (MSC). A detailed description of this method is given in [AABI00].
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.

3.1 Message Sequence Charts 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 [ITU94]. 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 a 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.

3.2 Deriving Queueing Network Models from Message Sequence Charts

Starting from the high level description of software architectures 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 events 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*. For the sake of simplicity we assume one to one communication between components and that the associated automata is complete, i.e. from every state it is possible to do all possible actions. Hence we can get the parallelism between components.

We encode the MSC as follows. Let $M = \{M_1, M_2,\ldots, M_n\}$ denote the set of MSC describing the architectural behavior of the set of components $C_1,\ldots,C_m$. An event e_j on a MSC M_i is a communication between two components, denoted by an arrow. We define a *visual ordering* of events e_j and e_k denoted by $e_j <_i e_k$ if e_j precedes e_k in the MSC M_i temporal sequence. 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 = s) or asynchronous (c = a). Given a set of ordered events $e_1,\ldots,e_j$ the associated trace is a regular expression of the corresponding labels $l_1,\ldots,l_j$ where each l_k is the label of e_k or a regular expression $\{b_1,\ldots,b_q)^n$ where $b_1,\ldots,b_q$ is the trace associated to a block of events in a repeat cycle. From the MSC we generate the labels of the events and then we build the traces according to the visual ordering given by the event sequences in the MSC. During the labels generation we produce the Based Interaction Sets representing all the communication in the MSC. In particular if we generate a label $S(C_1, C_2)^c$ at step i, then we put the Interaction Pair $(C_1, C_2)^c$ in a set called Based Interaction Set I_i. 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. A detailed description of the method and the algorithm is given in [AABI00].
In order to derive the queueing network model of the SA described by the MSC, from the trace analysis we obtain the Structured Interaction Sets that are examined to associate elements of the queueing network model. 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.
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.
We start from considering each component as an autonomous server. 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.
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. 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. Modelling 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.
In the examples we considered, the class of queuing network models with infinite capacity queues was not sufficiently expressive to model systems with synchronous communication between concurrent components. Hence, by observing the need of a more accurate definition of the performance models of software architectures to capture such feature of the communication systems, we consider queueing networks with finite capacity and blocking to 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 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 Figure 1. Assume that all the components can be active at the same

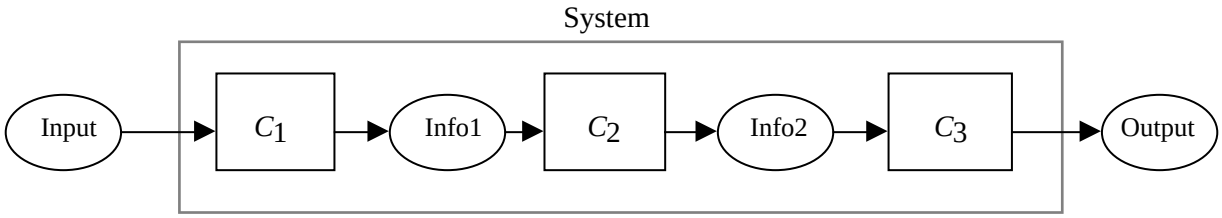

Figure 1 - Concurrent system with synchronous communication

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_1tries 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. Modelling this synchronous communication between the two concurrent components is illustrated in Figure 2.

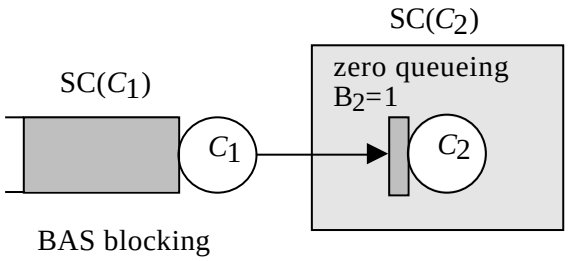

Figure 2 - Modelling synchronous communication between concurrent components

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.

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 Figure 3. Service center SC(C_1) can have in turn finite capacity if C_1 is also a destination component of a synchronous communication.

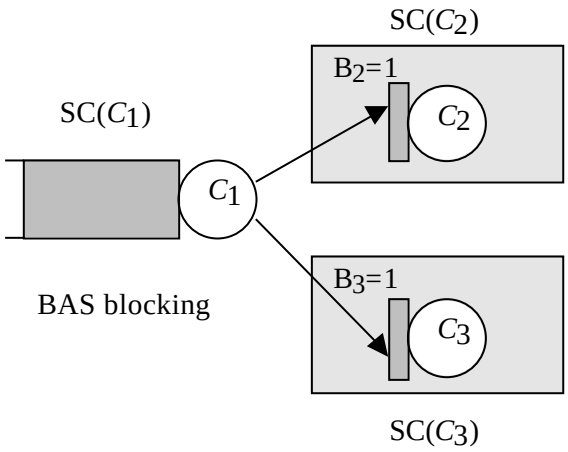

Figure 3 - Modelling one to two synchronous communication between concurrent components

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 maintaining the order of communication request times. Figure 4 illustrates two to one communication model for synchronous communication, i.e. with c1=c2=s.

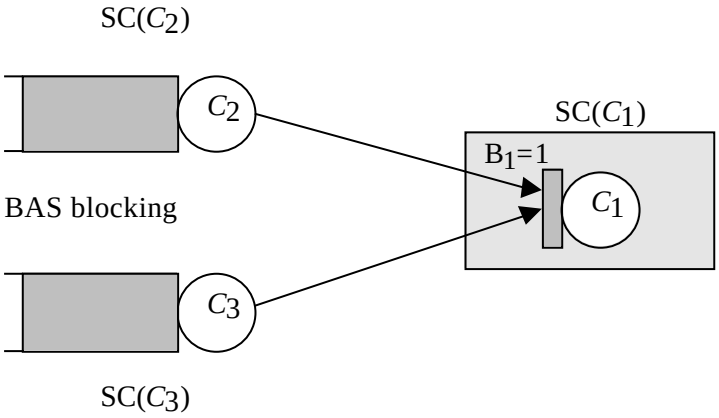

Figure 4 - Modelling two to one synchronous communication between concurrent components

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 capacity queue, because it models the buffer for asynchronous communication from an other component to component C_1 (i.e. $c1=a$). 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. $c1=c2=a$) 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 modelling 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 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 have still to perform the parameterization step of the modelling 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.
To this aim for queueing networks with blocking we usually derive single class models. Moreover we observe that exact analysis can be applied in particular cases and approximate solution is often necessary [Ba94, On90, On93, Pe94]. In particular the queueing network can be solved via the numerical solution of the underlying of the 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 ($B_i \le N$, $1 \le i \le M$) 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 [Ba94, BD94].
Various approximate solution methods for queueing networks with blocking have been proposed by

several authors [AP89, BR00, On90, Pe89, Pe94]. 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 [AK88a, Ak88b]. Several methods can be applied for open networks with blocking, depending on the topology, mostly based on the decomposition principle as in [BJ88, DF93, HB67, KX89, LBDF95, PA86].
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 examples of software architectures.

The first example is the design of a Compressing Proxy system as described in [CIW99]. 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 Figure 5. Components are denoted by square boxes and processes by ovals. The filters communicate using a function-call-based stream interface.

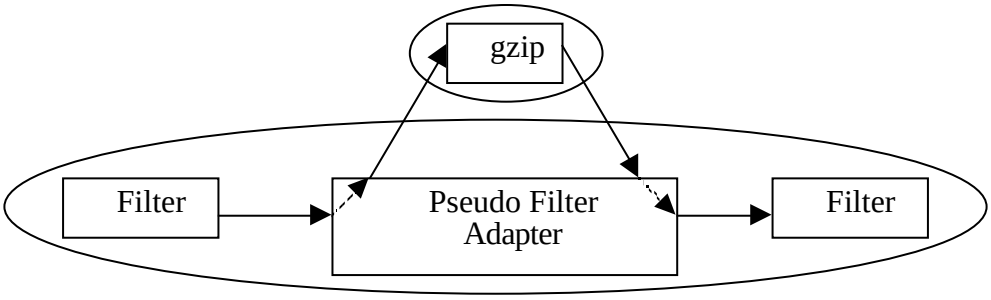

Figure 5 - The Compressing Proxy SA

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, 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 [AABI00]. 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. By considering the modelling 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 Figure 6.

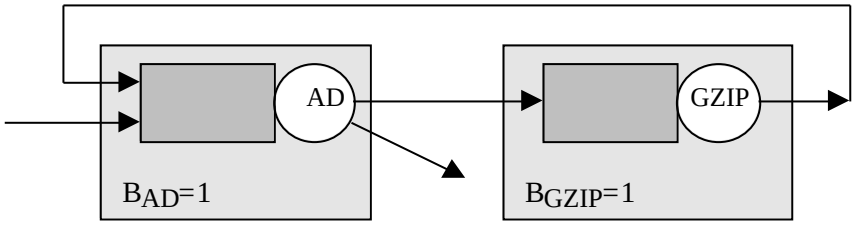

Figure 6 - The queueing network model of the Compressing Proxy system

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 tuple [FILTER1, AD, GZIP, AD, FILTER2] 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.
The queueing network model 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.
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 [IW95, PW92]. 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. We consider an optimized architecture of the multiphase compiler architecture illustrated in Figure 7.

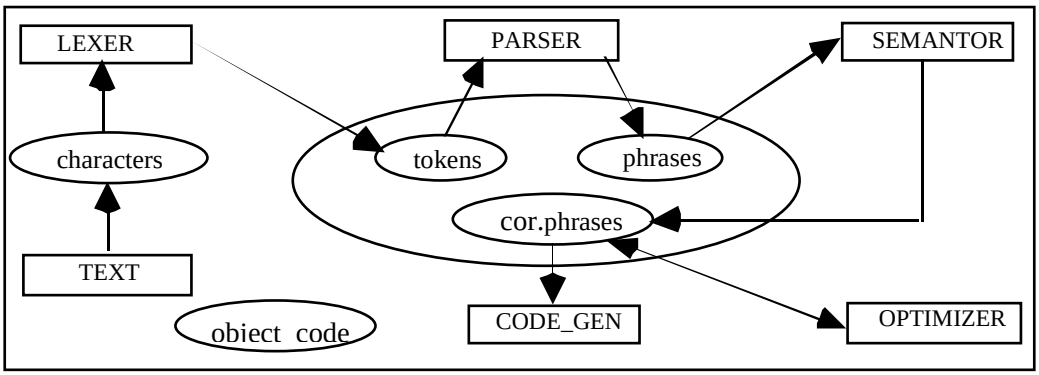

Figure 7 - The concurrent multiphase compiler architecture

It fits the processing elements together via concurrent access to a shared repository. The processing elements run their phases opportunistically and in parallel so that, for example, the semantor can correlate phrases at the same time as the lexer is creating new tokens.

We can apply the algorithm to derive the interaction sets and then from the trace analysis we eventually obtain the performance model. The derived queueing network is illustrated in Figure 8.

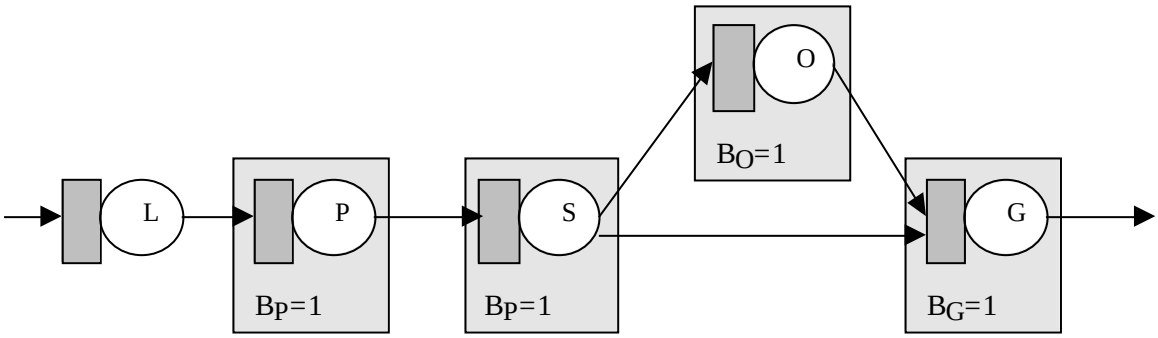

Figure 8 - The queueing network model of the Compressing Proxy system

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=P,S,O,G. We assume BAS blocking.
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 of the multiphase compiler architecture under various scenarios.
The queueing network model can be analyzed by approximate methods, as recalled in section 4.1. We can apply one of the algorithms based on the decomposition approach. For example we can analyze the queueing network with BAS blocking by the approximate algorithm described in [LBDF95] for acyclic networks.
Hence we can evaluate the network throughput and other average performance indices.

6. Conclusions

We have presented the application of the class of queueing networks with finite capacity and blocking as performance models of software architectures. In the framework of a methodology for performance evaluation of software architectures we have considered the problem of defining an appropriate modelling for the communication system.
Starting from a high level description of a SA by the Message Sequence Charts we obtain a trace analysis and we derive a queueing network as a performance model. However, we have observed that queueing networks with infinite capacities are not sufficiently expressive to model systems where there are concurrent components that can communicate synchronously. Hence, by observing the need of a more accurate definition of the performance models of software architectures to capture some features of the communication systems, we have proposed queueing networks with finite and single capacity queues and BAS blocking 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 in the construction of a flexible environment for performance evaluation of software architectures.

Acknowledgments

All the authors have been supported by MURST Project Research Fund (Progetto MURST di rilevante interesse nazionale) Saladin. http://www.saladin.dm.univaq.it.

References

[Ak88a] "On the exact and approximate throughput analysis of closed queueing networks with blocking" IEEE Trans. on Soft. Eng., Vol. 14 (1988) 62-71.

[Ak88b] Akyildiz, I.F. "Mean value analysis of blocking queueing networks, IEEE Trans. on Soft. Eng." Vol. 14 (1988) 418-129.

[AP89] Akyildiz, I.F., and H.G. Perros Special Issue on Queueing Networks with Finite Capacity Queues, Performance Evaluation, Vol. 10, 3 (1989).

[ABI00] Aquilani, F., S. Balsamo, P. Inverardi "An Approach o Performance Evaluation of Software Architectures" Res. Report, University of L'Aquila, March 2000.

[AABI00] Andolfi, F.,F. Aquilani, S. Balsamo, P. Inverardi "Deriving Performance Models of Software Architectures from Message Sequence Charts" Proc. WOSP 2000, Second Int. Workshop on Software and Performance, Ottawa, Canada, Sept. 17-20, 2000.

[Ba94] Balsamo, S. "Properties and analysis of queueing network models with finite capacities", in Performance Evaluation of Computer and Communication Systems, Lecture Notes in Computer Science, 729, 1994, Springer-Verlag.

[BR00] Balsamo, S., Rainero A. "Closed queueing networks with finite capacity queues: approximate analysis" Proc. ESM 2000, SCS European Simulation Multiconference, Ghent (B) 22-26 May, 2000.

[BC95] Balsamo, S., C. Clò "A Convolution Algorithm for Product Form Queueing Networks with Blocking" Annals of Operations Research, Vol. 79 (1998) 97-117.

[BD94] Balsamo, S. and V. De Nitto Personè, A Survey of product-form queueing networks with blocking and their equivalences, Annals of Operations Research, 48, (1994) 31-61.

[BIM98] Balsamo, S., P. Inverardi, C. Mangano "Performance Evaluation of Software Architectures" in ACM Proc. WOSP, Santa Fe, New Mexico 1998.

[BCK98] Bass, L., P. Clements and R. Kazman. "Analyzing Development Qualities at the Architectural Level". Software Architecture in Practice. SEI Series in Software Engineering, Addison-Wesley 1998.

[BJ88] Brandwajn, A., and Y.L. Jow "An approximation method for tandem queueing systems with blocking" Operations Research, Vol. 1 (1988) 73-83.

[CIW99] Compare, D., P. Inverardi and A. L. Wolf. "Uncovering Architectural Mismatch in Component Behavior" in Science of Computer Programming, No. (33) 2, (1999) 101-131.

[DF93] Dallery, Y., and Y. Frein "On decomposition methods for tandem queueing networks with blocking" Operations Research, Vol. 14 (1993) 386-399.

[HB67] Hillier, F.S., and R.W. Boling "Finite queues in series with exponential or Erlang service times - a numerical approach" Operations Research, Vol. 15 (1967) 286-303

[HNS99] Hofmeister, C., R.L. Nord and D. Soni. "Describing Software Architecture with UML" in 1st Working IFIP Conference on Software Architecture (WICSA1), pp. 145-159, San Antonio, Texas Berlin, Germany, 22-24 February 1999.

[KX89] Kouvatsos, D., and N.P. Xenios "MEM for arbitrary queueing networks with multiple general servers and repetitive-service blocking" Performance Evaluation Vol. 10 (1989) 106-195.

[IW95] Inverardi, P. and A. L. Wolf. "Formal Specification and Analysis of Software Architectures Using the Chemical Abstract Machine Model" IEEE Transactions on Software Engineering, Vol. 21, No. 4, 373-386, April 1995.

[ITU94]. ITU-TS. ITU-TS recommendation Z.120. Message Sequence Charts. ITU-TS, Geneva, 1994.

[Ka92] Kant, K. Introduction to Computer System Performance Evaluation. McGraw-Hill, 1992.

[Kl76] Kleinrock, L. Queueing Systems, John Wiley and Sons, New York, 1976.

[La82] Lavenberg, S.S. Computer Performance Modeling Handbook. Prentice Hall, 1983.
[LZGS84] Lazowska, E. D., J. Zahorjan, G. Scott Graham, K. C. Sevcik. Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Englewood Cliffs, (1984).
[LBDF95] Lee, H.S., A. Bouhchouch, Y. Dallery and Y. Frein "Performance Evaluation of open queueing networks with arbitrary configurations and finite buffers" Proc. Third International Workshop on Queueing Networks with Finite Capacity, Bradford, UK, 6-7 July, 1995.
[Sm90] Smith, C. U. Performance Engineering of Software Systems. Addison-Wesley Publishing Company, (1990).
[SW93] Smith, C. U. and L. G. Williams "Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives" IEEE Trans. on Software Engineering, Vol. 19, No. 7, 720-741, July 1993.
[SG96] Shaw, M. and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
[On90] Onvural, R.O. "Survey of Closed Queueing Networks with Blocking" ACM Computing Surveys, Vol. 22, 2 (1990) 83-121.
[On93] Onvural, R.O. Special Issue on Queueing Networks with Finite Capacity, Performance Evaluation, Vol. 17, 3 (1993).
[PW92] Perry, D.E. and A. L. Wolf. Foundations for the Study of Software Architecture. ACM SigSoft Software Engineering Notes, Vol. 17, No. 4, 40-52, October 1992.
[Pe89] Perros, H.G. "Open queueing networks with blocking" in Stochastic Analysis of Computer and Communications Systems (Takagi Ed.) North Holland, 1989.
[Pe94] Perros, H.G. Queueing networks with blocking. Oxford University Press, 1994.
[PA86] Perros, H.G., and T. Altiok "Approximate analysis of open networks of queues with blocking: tandem configurations" IEEE Trans. on Software Eng., Vol. 12 (1986) 450-461.
[VD91a] N. van Dijk, On ‘stop = repeat’ servicing for non-exponential queueing networks with blocking, J. Appl. Prob., 28 (1991) 159-173.
[VD91b] N. van Dijk, ‘Stop = recirculate’ for exponential product form queueing networks with departure blocking, Oper. Res. Lett., 10 (1991) 343-351.
[Wo98] WOSP'98, IEEE First International Workshop on Software and Performance, Santa Fe, New Mexico, USA, Oct. 12-16, 1998.