PERFORMANCE ANALYSIS AT THE SOFTWARE ARCHITECTURAL DESIGN LEVEL

FEDERICA AQUILANI, SIMONETTA BALSAMO*, PAOLA INVERARDI

ABSTRACT. In this paper we present an approach to evaluating the expected performance of a software architecture (SA). The method allows the derivation of a performance evaluation model, based on a Queueing Network Model, from a software architecture formal specification modelled as a Labeled Transition System (LTS). The goal of this approach is to provide a set of measures to compare the performance of two or more competing software architectures, even at their high level of abstraction. This means that, differently to most approaches in software performance, we do not assume to have, or to expect to obtain, further information on the system under development from the subsequent development steps, e.g. design, implementation and deployment. Rather we stick at the level of abstraction of the SA and even in the absence of relevant pieces of information we want to be able to carry on an evaluation which should allow the validation of the possibly critical design choices made at the SA level. We aim at deriving mathematical relations among the parameters characterizing the competing models. These relations are obtained by using implementation scenarios to relate the performance indices which can then be suitably evaluated. Besides assisting in the choice of a software architecture, these relations give information on how to carry on the development process in order to maintain the given performance. To a certain extent, the information obtained by means of the evaluation provides further requirements to be fulfilled by an implementation in order to satisfy the chosen performance criteria. This last step suggests the possibility of developing performance models at the SA design level that can then be used to derive the performance models related to further development steps.

1. INTRODUCTION

Software Architectures (SA) describe software system structures at a high level of abstraction [23, 31]. They represent at an early stage of development the phase in which basic choices of components and interactions among components are made. Their main feature is that, besides a description of the system static structure, a description of the dynamic behavior of the system should be provided. Both the static and the dynamic descriptions are very high level but embody very relevant design choices that, if wrong, can invalidate the successful development and deployment of the final system.

Designers are often faced with the problem of choosing among different functionally equivalent software architectures. This choice is driven by non-functional factors such as performance, reliability, and topological/economical constraints. Among these, performance is one of the most influential factors to be addressed. Several authors have claimed the importance of an as earliest as possible in the software life cycle, analysis of the functional and non-functional behavior of the system under development [31, 6, 36]. Software Architectures represent the first system abstraction that allow the effective validation of these properties.

As far as performance is concerned, SAs offer a first dynamic model of the system which should permit the validation of the gross-grain software architectural decisions. These decisions concern the decomposition choices of the system in several subsystems with defined functionalities and the decisions related to the way the various subsystems coordinate and communicate. Several pieces of information which might become relevant to performance measurements are left out from the SA description, due to the intrinsic high abstraction nature of the SA, nevertheless supporting quantitative analysis at this abstraction level is not only advisable but mandatory. In practice software designers do make their choices among different SAs and very often in order to support their decisions they use performance arguments which are made on the base of experience, both on the kind of system under development and on their own experience, without the help of any rigorous conceptual framework. In this paper we present a method that allows the derivation of a performance evaluation model, based on a Queueing Network Model (QNM), from a software architecture formal specification modeled as a Labeled Transition System (LTS). The goal of our approach is to provide a quantitative performance analysis of two or more competing SA, even at their high level of abstraction. Specifically, we derive a performance model from a SA formal specification model and we compare various competing SAs under different scenarios. This means that, differently to most approaches in software performance, we do not assume to have, or to expect to obtain, further information on the system under development from the subsequent development steps, e.g. design, implementation and deployment. Rather we stick at the level of abstraction of the SA and even in the absence of relevant pieces of information (e.g. software detailed design, hardware architecture) we want to be able to carry on an evaluation which should allow the validation of the possibly critical design choices made at the SA level. Due to the high level of abstraction of SA, we aim at deriving a simple performance model at the appropriate level of abstraction and that can be symbolically evaluated. Hence we consider the class of Queueing Network models that has been extensively applied for system performance evaluation at a high abstraction level and that allows simple closed form solution under certain constraints [18, 19, 20].

Whereas the model cannot be symbolically evaluated, we carry on the performance analysis through numerical solution or by simulation.

In the better situations, the results are a set of mathematical relations characterizing the competing models. These measures can be used either to directly compare the models or to devise the performance of the given architectures under further hypothesis by suitably instantiating the symbolic parameters. Parameter instantiations identify potential implementation scenarios and analyzing the behaviour of the system in these scenarios can provide useful insights on how to carry on the development process in order to satisfy the chosen performance criteria. To a certain extent, this evaluation process provides further requirements to be fulfilled by an implementation in order to satisfy the chosen performance criteria. This is a very relevant feature of our approach since a common problem is to be able to carry on along the development process the performance requirements. Usually performance requirements are expressed at the level of the global system. Breaking down a global performance requirement into performance requirements that apply at subsystem level allows the definition of a sound refinement process for global performance requirements with respect to the chosen validated SA.

Summarizing in our methodology we pursue two goals. The first is to be able to measure the performance of SAs at their high abstraction level. The second is to be able to interpret the evaluation results in terms of the system development process. That is, the result of bad measures should allow for an easy identification of the wrong design choices. Viceversa the result of good measures should allow for the definition of new performance requirements on the subsystems and connectors that have still to be realized.

1.1. **Related Work.** The impact that quantitative analysis of software system can have in software design has been recognized long ago in the software engineering community. Comprehensive approaches to this topic exist, notably the work by Smith [32, 33]. There is more and more concern that quantitative analysis of software system is important and useful for requirements and design, and the earlier it can be performed in the software development life cycle the better it is [27]. To this respect we can distinguish two branches of research. On one side there are people that cast their work in the entire software life cycle, from requirements to design to implementation and to deployment. In this group we have [32, 33, 25, 11, 14, 28, 38]. All these approaches consider performance analysis integrated in a complete software life cycle, with more and more emphasis towards object-oriented software development. They assume to manage a certain number of software artifacts, from requirements specifications (Use cases, etc.) to deployment diagrams which describe the mapping of software components onto the hardware architecture, that can be used to drive the performance evaluation process.

Most of the approaches consider QNM as performance model whose topology and service center parameters are obtained from the information on the physical architecture, while the information on software components is used to define the model workload [11, 14, 15, 25, 26, 28, 32, 33, 34, 36, 37, 38, 39].

The second branch of research identifies a precise software stage, typically the design specification and assumes to have a formal behavioral software specification. Usually this specification is modelled by Stochastic Petri Nets [2, 30] or Stochastically Timed Process Algebras [7, 9, 29] and allows for both behavioral and performance analysis by means of a unique reference model.

Recently, software architectures have been devised as the appropriate design level for carrying out quantitative analysis [36, 6, 35] and much work appeared aiming at addressing this issue. Besides the more software engineering methodologies for performance evaluation of SA, like the one proposed at the Software Engineering Institute [6], works representative of the two research branches mentioned above, appeared [34, 29, 26, 9, 15, 30]. These works applied their respective approaches to SA. In the first case the SA is explicitly modeled and becomes part of the software life cycle and then it contributes to the construction of the QNM model and its workload. In the second case the SA is modeled as a Stochastic Petri Net specification or a Stochastically Timed Process Algebra term and suitably parameterized in order to evaluate/simulate the corresponding Markov models. The first approach thus does not measure the performance of the SA itself, rather it uses the SA description to derive the performance model of the final software system. The second approach instead evaluates the SA specification, the problem with this approach is that the evaluation of the performance model allows for getting a measure but does not permit easy feedback of the results at the SA design level. The gap between the SA modelling in terms of components and connectors and the performance models is too large to allow easy identifications of design feedbacks.

Our approach differs from the two mentioned above in various respects, that we will try to highlight and motivate in the following. First we want to measure the SA performance like in the second approach. Then we want to be able to properly interpret the result of the measures in terms of the SA design. The latter has driven our choice of QNM as performance model. Thus we assume a behavioral specification of the SA specification, in our approach we assume to have a Labeled Transition System that models the global dynamic behavior of the SA. Then we analyze this model in order to identify the subsystems that actually can behave concurrently. Only these ones will correspond to components of the QNM. As we will see in the following, already the generation of the QNM allows for a sort of validation of the decomposition and connection choices of the SA from the performance perspective. Then we can proceed in the evaluation of the model as shortly described above. Due to the high level of abstraction we obtain a QNM not completely specified, so we can choose appropriate conditions to simplify the model evaluation possibly by symbolic analysis. We are then able to get feedback from these results both in the positive and in the negative cases. We presented our approach in [4, 5]. In these works we were relying on a specific Architectural Description Language (ADL), namely CHAM [10, 16], and we were analyzing a restricted set of possible interaction patterns among the architectural components. In this paper we extend and better motivate our approach. First we formulate the approach independently of a specific ADL but only relying on a finite state model representation. Second we deal with more complex interaction patterns thus enlarging the class of QNM we can generate. Moreover we deal more extensively with the quantitative analysis, showing how to analyze the interesting evaluation indices.

The paper is organized as follows: Section 2 introduces the necessary background and the conceptual framework our approach refers to. Our performance evaluation method of Software Architectures is described in Section 3 and in Section 4 it is exemplified by using the well-known example of the multiphase compiler. Section 5 carries out the evaluation phase. Section 6 adds some concluding remarks and discusses future work.

2. Foundation of the approach and Conceptual Framework

In this section we briefly recall the basic notions and notations our approach is based on.

Our approach to Performance Evaluation is exemplified in Figure 1. It assumes to start with a SA description, possibly formally defined. Then it focuses on the SA dynamic behavioral model and analyzes it in order to identify a suitable faithful performance model. During this phase we try to identify the real degree of concurrency among the components. The idea here is that the SA designer when proposing a SA implicitly proposes the ideal abstract machine on which the system structured that way should run. Components in fact are assumed as independent grain of computation which might remain independent when assembled in a composed system if the way they are forced to interact does not constrain their potentially concurrent behavior. Once the Performance model has been obtained,

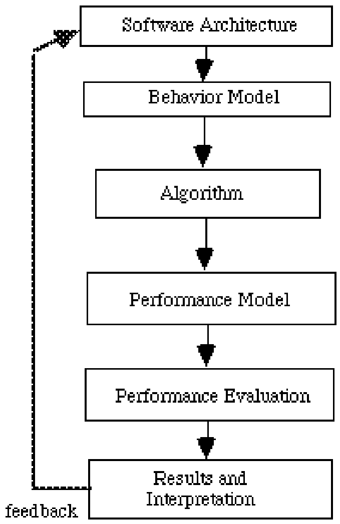


FIGURE 1. Phases of the proposed approach

its evaluation can be carried on. The result of the evaluation should then allow for feedback on the previous steps of the process.

2.1. **SA and models.** Software architectures represent the overall system structure by modeling individual components and their interactions. *Components* model subsystems, that are seen as black boxes whereas only their interaction capabilities with respect to other components is outlined. Interactions can take the form of simple basic interaction mechanisms, e.g. synchronous or asynchronous communications, or can be described as more sophisticated communication protocols. In the SA notation, interactions become first class citizens and are referred as *connectors*. Components and Connectors are the constituting elements of a software architecture description [31].

Therefore an SA description provides a complete system model by focussing on architecturally relevant abstractions. A key feature of SA descriptions is their ability to specify the dynamics. Finite State Machines (FSMs), Petri Nets or Labelled Transition Systems (LTSs) can be used to model the dynamics (behaviour) of a Software Architecture description, describing the set of all possible SA behaviors as a whole.

In the following we will not rely on a particular Architectural Language, rather we will directly refer to the system architecture dynamic model that we assume to be a Labelled Transition System.

Concurrent system description using LTS is widely used in system specification and SA modeling (e.g. [13, 12]). In the following we will briefly recall the definition of LTS we use and all the assumptions we do on our modeling.

The precise definition of the LTS model follows below:

Definition 1 (Labelled Transition Systems)**.** *A system C is modelled by a labelled transition system $< S, L, \rightarrow, s, P$, where S is a set of states; s is the initial system state; L is a set of labels representing the system communication;* $\rightarrow \subseteq (P \times Act \times P)$

is a transition relation that describes the behavior of the system and P is a set of state labels that describe the state of each single component in that system state.

We assume to obtain this LTS model out of a SA description in which components are modeled as communicating concurrent subsystems. Thus a system state is obtained by observing the concurrent behavior of all the subsystems. The set P allows us to record inside each system state the state of every single component involved in that system transition. The set L allows transitions to be identified, thus recording the kind of communication associated to that system transition. We recall again that at the architectural level we only observe system behaviors concerning interaction activities among components, thus transitions are only due to communication activities among components.

This kind of modeling is common in any compositional approach in which a complex system is described as the parallel composition of its communicating component subsystems. With the above assumptions in the following we can abstract away from the use of a specific ADL and concentrate on the SA modeling issues relevant to SA performance evaluation purposes only.

2.2. **The Queueing Network Model.** We consider Queueing Network Models (QNMs) as performance model of SA. QNMs have been extensively applied to represent and analyze the performance of resource sharing systems, such as communication and computer systems. They provide a powerful tool for system performance evaluation and prediction. QNMs achieve a good balance between accuracy in the performance results and efficiency in model analysis and performance evaluation [18, 19, 20]. Queueing networks provide a simple model at a high level of abstraction, intuitively understandable and that clearly represents resource contention. System performance evaluation with QNMs consists in the definition and the parametrization of the model and in its solution to evaluate a set of performance indices, such as throughput or customer's response time. Quantitative evaluation is based on analytical methods or simulation. The meaning of the performance indices and the interpretation of the performance results depend on the considered level of abstraction in model definition and the correspondence of system and model components.

QNMs as models of computer systems have been often used to represent resource contention at a physical level, e.g. at hardware architecture level. Several approaches of quantitative analysis for software performance evaluation are based on QNMs and consider an integration of the performance analysis in the software life cycle. They define a QNM whose structure usually represents the physical architecture, that is the hardware platform, and whose workload parameters are derived by the software model. We shall now apply QNMs as models at the SA level of abstraction.

Informally, a queueing network is defined by the service centers, the customers and network topology. Service center characteristics include the service time, the buffer space with its queueing scheduling and the number of servers. Customers are described by their number for closed models and by the arrival process to each service center for open models, the service demand to each service center and the types of customer. Network topology models how the service centers are interconnected and how the customers move between them. Different types of customer in the queueing network model can model different behaviors of the customers. This allows representing various types of external arrival process, different service demands and different types of network routing. A chain gathers the customers of the same type. A chain consists of a set of classes that represent different phases of processing in the system for a given type of customer. Each chain can be open or closed depending on whether external arrivals and departures are allowed. Multiclass or multichain networks may be open or closed if all the chains are open or closed, respectively. A mixed network has both open and closed chains. Multiclass models can be used for a more precise representation of system behavior and to obtain more detailed performance indices.

The buffer space of each service center can be finite to represent finite capacity system resources or population constraints. Basic QNM have been extended to model special features of more complex systems, such as finite capacity queues, concurrency and synchronization constraints and simultaneous resource possession.

QNMs with finite capacity queues are more appropriate models for systems with finite buffer and blocking, and to represent some synchronization constraints. Their analysis is in general more complex than basic QNMs and it is often carried on with approximated techniques [24]. This kind of QNMs are referred to as Queueing networks with Blocking [3, 18, 19, 20, 22, 24]. 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 system. Hereafter we consider one type of blocking named Blocking After Service (BAS) [3, 22, 24] that can be used to model some synchronous communication constraints.

Under this policy when a user, after having completed the service at a service center i, tries to enter a saturated node j, it is forced to wait at node i, till 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. When more than one node is blocked by the same saturated node, a scheduling discipline must be considered to define the unblocking order of the blocked nodes. We refer to FBFU (First Blocked First Unblocked) discipline, where the first node blocked is the first unblocked one.

In the following we consider queueing networks with blocking as performance models of software architectures with concurrent components and synchronous communication

For simplicity, hereafter we will only deal with open models, therefore we only consider features of open models.

Informally, the creation of a QNM can be split in three steps: *definition*, which normally encompasses the definition of service centers, their number, class of customers and topology; *parameterization*, to define the alternatives of the study, e.g. by selecting the arrival processes and service rates, *evaluation*, to obtain a quantitative description of the system behaviour described by the QNM. This step is based on the computation of a set of figures of merit or performance indices, such as resource utilization, system throughput and customer response time. These indices can be *local* to a resource or *global* to the whole system.

We assume that the system reaches a steady-state behaviour, that is a stationary condition holds, and we analyze the steady-state system model. The analysis of a QNM is carried out depending on its characteristics. Under general assumptions it can be reduced to the analysis of an underlying stochastic Markov process. Under some constraints the QNMs analysis can be greatly simplified.

For the *product-form* class of QNM efficient solution methods have been defined [18, 19, 20]. Product form network analysis is based on a set of assumptions on the system parameters that lead to a closed form expression of the stationary state distribution.

Hence we usually try do derive models that belong to this class of QNMs.

In some cases we shall consider QNM with blocking as performance models of SA. Such models have a product form solution only under particular constraints also depending on the blocking type, and they can be in general analyzed by approximate analytical methods or simulation [3, 24].

2.3. **QNM for SA.** In the proposed framework we consider QNMs as performance models of SAs. The goal of the proposed approach is to evaluate and compare SA specification to derive performance results that can be interpreted in terms of SA design.

Therefore, unlike other approaches that consider QNMs at a physical architecture level, we consider QNMs as performance models at a higher level, the software architecture one. The abstraction process that leads to the definition of the QNM from the SA specification identifies SA components and connectors as possible model components, i.e., service centers. The dynamic activation of software components and connectors is modeled by customers' service time in the QNM. Concurrent component activation in SA is represented by customers' concurrent activity in the QNM that possibly compete for the use of shared resources. Synchronous communication of concurrent components can be modeled by service centers with finite capacity queues and an appropriate blocking protocol, as in QNM with BAS blocking.

The definition of a QNM at the SA level could be not completely specified, because of the high level of abstraction. If some parameters and characteristics of the performance model are not specified we can choose the appropriate conditions that leads to an efficient evaluation of the model, possibly by symbolic analysis.

The analysis of the QNM of the SA leads to the evaluation of some performance indices that have to be interpreted according to the model abstraction level, that is at the SA level. The model parameter instantiations correspond to potential implementation scenarios and the performance evaluation results can provide useful insights of how to carry on the development process at the SA design in terms of potential performance. Performance indices are interpreted at the SA level, by considering the meaning of the parameters instantiation. Note that this approach does not refer to a performance model at the hardware implementation level, but we aim at deriving performance measure and comparison of competing software architectures at a high level of abstraction.

3. Proposed Framework

We shall now describe the proposed approach introduced in the previous section by considering the chosen behavioral and performance model of SA. The phases of the proposed approach are illustrated in Figure 2. Starting from the formal specification of the SA dynamics modeled as a LTS we apply an algorithm to derive the performance model, that is the QNM. The complete definition of the QNM is also based on some additional information about the state annotation and the type of communication among SA components. Then we evaluate the QNM by considering parameter instantiation determined by possible scenarios. Finally, we obtain performance evaluation results by the analysis of the QNM of the SA and by their interpretation we can derive feedback on the development process at SA level. We now focus on the three main phases: the algorithm to derive the QNM from the SA specification, the QNM performance evaluation and the feedback analysis.

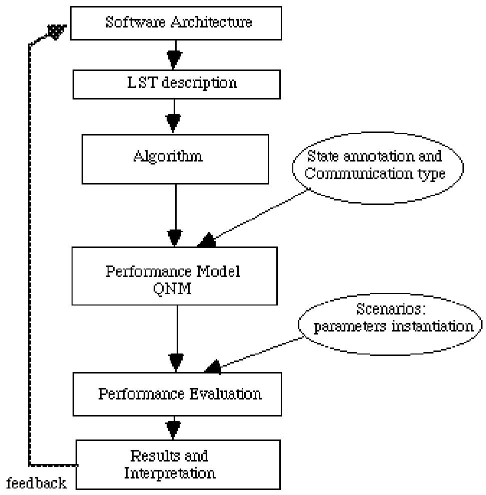


FIGURE 2. Phases of the instantiated approach

3.1. **From the SA specification model to the QNM.** In this section we present a scheme of the algorithm to derive the performance evaluation model out of the LTS specification of software architecture. We consider a LST independently of a specific ADL but only relying on a finite state model representation. A LTS models the global dynamic behaviour of the SA. We analyze this model in order to identify the subsystems that actually can behave concurrently. Only these ones will correspond to components of the QNM. The algorithm actually performs the first step of the modeling process described in the previous section, i.e. the QNM definition.

The algorithm is based on the analysis of the transition system (LTS) that represents the dynamic behavior of the SA. We assume to deal with LTS descriptions that conforms to the one given in Section 2.1.

We derive the QNM from the analysis of the interactions among system components that are dynamic information that can be derived only from a behavioral model of the SA system.

Without loss of generality, we assume to deal with a LTS in which the transitions corresponding to non-deterministic behavior are marked in order to distinguish interactions corresponding to an actual non-deterministic choice in the computation and interactions corresponding to concurrent communications that are modeled in the LTS as non-deterministic computations. This can be easily obtained when building the LTS from the system description.

Informally, in order to derive the QNM from the LST we identify the SA components and connectors that can be represented by service centers of the QNM, and the interactions among system components that determine the QNM topology.

The algorithm is organized into two sequential steps. The first step (PHASE 1) examines all the LTS paths and the states along the path in order to single out all the pairs of interacting components. The second phase (PHASE 2) analyzes and compares these interaction pairs to derive the service centers and the network topology of the evaluation model.

Specifically, let P denote the set of SA components and connectors whose dynamic behavior is described by the LTS. Let $G = (S, A, T)$ denote the labeled directed graph representing the LST, where $S = \{S_1, \ldots, S_N\}$ is the set of states, $A = \{a_1, \ldots, a_M\}$ is the set of directed arcs, $a_i = (S_j, S_k)$, $S_j, S_k \in S$ and $T = \{T_1, \ldots, T_M\}$ is the set of labels that represent the communications, T_i is the transition label of arc a_i. Transitions in the LTS describe explicit interactions between system elements or implicit interactions between some system components and the environment or external components. Hence we define a partition of T in two sets of internal and external transitions, denoted by IT and ET, respectively. When there are external interactions, i.e. $ET \neq \emptyset$, we introduce a new element called ENV in P to represent the environment. For a systems with only internal interactions we usually derive a closed QNM, otherwise we obtain an open network. The algorithm derives the QNM that represents the real level of concurrency of the system. The goal is to identify the components that are strongly synchronized so that their behavior is sequential and the components that are independent and can be concurrently active.

We consider each component as an autonomous server first, thus modelling the maximum level of concurrency. Then we derive a simple structure of the QNM by analyzing the true level of concurrency and the communication types.

The algorithm can be sketched as follows.

PHASE 1

In this phase the algorithm performs a LTS visit by considering all the paths in graph G in order to derive interaction sets formed by interaction pairs. The algorithm examines all the adjacent states of a visited state and the connecting transitions.

If the transition corresponds to a communication the algorithm generates a pair, called *Interaction Pair (IP)*. An IP denoted by (p_1, p_2) signals a flow of data elements p_1 to p_2, where p_1 is the system component that acts as sender in the communication, while p_2 is the component that acts as receiver. For synchronous communication both elements of the pair are system components, while asynchronous communication is modeled by introducing connecting elements with buffer, called *Passive Connecting Elements* (PaC), that are elements of the corresponding interaction pair. Moreover, the algorithm marks the interaction pairs corresponding to a system non-deterministic behavior, so to correctly distinguish, and model, concurrency and non-determinism, in the second phase. All the interaction pairs derived at the i-th visit step are put in Interaction Set I_i, that represents all the communications among system elements that can happen at a given time.

PHASE 2

The second phase examines sets I_i defined in Phase 1 to generate the service centers and the topology of the QNM.

Besides the element ENV in P that represents the environment, we have to distinguish the system components that are represented as internal or external elements of the QNM. Hence we identify a subset A of set P, that contains those SA elements to be represented as external environment in the QNM.

Elements in A can be seen as sources which model the production of system customers or elements from which the system communicates the results to the environment. If we have introduced the element ENV, then it belongs to the set A. From A we can determine whether the QNM is open or closed.

Informally, the second phase algorithm analyzes the interactions among system components, i.e. interaction sets I_i, to identify their *real* level of concurrence. The goal is to identify components that are strongly synchronized and have a sequential behavior, even if they are modeled in the SA description as independent entities, and independent components that can be concurrently active.

First each component is considered as an autonomous server, thus modeling the maximum level of concurrency, then it can become part of a more structured QNM element.

Intuitively, if an *interaction pair* (p_1, p_2) corresponds to a synchronous communication, then the algorithm defines a corresponding QNM element that is a complex server with a unique service composed of p_1 followed by p_2 to represent the sequence of operations.

If an *interaction pair* (p_1, p_2) corresponds to an asynchronous communication, then the component that receives data is modeled as a service center with a infinite buffer that implicitly models the communication channel. Moreover from this interaction pair we can build the customer transition in the QNM.

The interaction pair (p_1, p_2) with an external element, i.e. where p_1 or p_2 belong to A, corresponds either to a service center with exogenous arrivals (if $p_1 \in A$) or from which there are departures (if $p_2 \in A$).

To model synchronous communication among concurrent system components the algorithm assigns distinct service centers to the communicating components in order to model their independence. It also associates to the receiver component a service center with a zero capacity buffer and assumes BAS blocking mechanism for the sender component in order to model synchronization.

The algorithm models a non-deterministic computation by introducing multi-customer service centers that at the end are transformed in simple-customer service centers whose service times depend on the service times of the original classes.

One to many and many to one communications are modeled assigning to the involved components distinct service centers, with a zero capacity buffer if the communication is synchronous.

When each interaction set I_i has been examined the algorithm proceeds by performing *merging* operations to reach the final configuration of the service centers. Several merging operations reduce the number of service centers of the QNM to model only the necessary concurrency of the system components. The result of these merging operations is the final set of interconnected service centers of the QNM.

The Appendix gives a more detailed description of the two-phase algorithm and the analysis of its complexity, as well as the definition and graphical notation of the QNM elements, that refer to the SA elements.

3.2. **Evaluation.** From the algorithm we do not obtain a completely specified QNM, because of the high level of abstraction of the SA model. We still need to define some of the quantitative parameters that characterize the model, that is we have to perform the parameterization step of the modeling process. The parameters to be still defined are:

- the service time distribution for every center;
- the arrival time distribution of the customers arrivals to every center;
- the network routing probability.

The complete specification of the QNM has to be done by the designer according to the information gained in running the algorithm and his knowledge of the QNMs solution methods. So we can choose appropriate conditions to simplify the model evaluation possibly by symbolic analysis. Hence we determine the quantitative parameters so that the resulting QNM can be efficiently solved. To this aim we usually choose exponential service and arrival time distribution and routing probability that leads to the definition of a product-form QNM, whose analysis is obtained by efficient solution methods [18, 19, 20]. When the QNM has finite capacity service centers and blocking, we consider single class models whose analysis is based mostly on approximate analytical methods [3, 24]. We usually refer to the steady-state analysis of the QNM in order to perform the evaluation process for a system in equilibrium. Nevertheless in some cases a transient analysis, which leads to a high computational solution cost, could be considered if we want to model a transient behavior of the SA.

The complete definition of the parameters of the QNM includes the instantiation of the average of the service time and arrival time distribution at every service center.

Note that the QNM of the SA is defined at a high level of abstraction and this determines the meaning of the mean service times of the service centers and mean arrival rates. Parameter instantiations identify potential implementation scenarios and the evaluation process is carried out possibly by symbolic analysis. The system designer defines the implementation scenarios that may be determined also by the knowledge or assumptions on the final implementation platform. This system evaluation and comparison of the performance results obtained by the analysis of the QNM under various scenarios can provide useful insights on how to carry on the development process in order to satisfy a given performance constraint.

3.3. **Analysis feedback.** The evaluation of the QNM of the SA provides a set of results in terms of performance indices for each set of parameters that represent a possible scenario. Hence the designer has to interpret these quantitative results as feedback at the SA design level, according to the proposed framework shown in Figure 2. Such quantitative results may suggest confirming or modifying architectural choices at the design level according to given performance requirements or criteria.

Therefore it is important to provide a clear and direct correspondence between the software architecture design and the derived performance model. The high level of abstraction of the QNM allows to relate SA components and model components, so making easier the feedback process.

In the following we illustrate the proposed framework with some examples, focusing in the next sections on the derivation of the QNM from the SA specification the parameterization, evaluation and feedback, respectively.

4. Deriving the QNMs for the three multiphase compiler architecture

In this section we illustrate our methodology by using three different architectures for a well known system, namely, the sequential, concurrent and intermediate architectures of the multiphase compiler.

Let us recall the well known multiphase compiler architecture [16, 23]. 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 components are the text (i.e., the producer of source characters), the lexer (i.e., lexical analyzer), the parser, the semantor (i.e., semantic analyzer), and the code generator. An optional component is the optimizer. The connecting elements consist of two operations, i and o, acting on the data elements and representing the communication ports, input and output respectively.

The *classical* multiphase compiler architecture, see Figure 3, fits the components together sequentially in the obvious way, with each component running its phase to completion before passing data elements on to the next component. A new text can start the compilation only after the completion of the current one. A rather different architecture, see Figure 4, fits the components together via concurrent access to a shared repository. The components 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.

A third intermediate architecture has essentially a sequential structure but a new text can start the compilation even if the current one has not yet been completed.

We shall now focus on the derivation of the QNM from the SA specification for the three software architectures of the multiphase compiler.

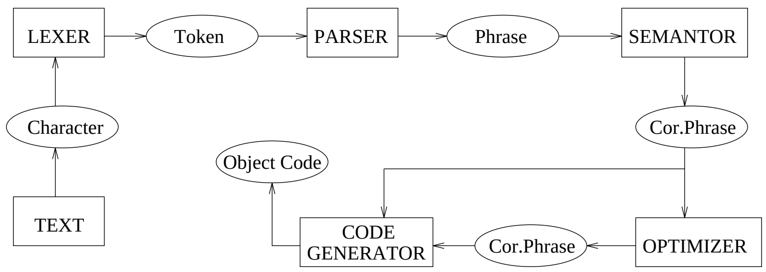


Figure 3. Sequential multiphase compiler architecture

4.1. The model for the sequential architecture.

From a specification of the sequential architecture like in [16], the LTS in Figure 5 can be derived. The labels, $S_1, S_2, \ldots$ are state labels with information describing the global state in terms of the single components state. The labels on arc denote interactions between states. The $*$ symbol is used to mark non-deterministic transitions. Transitions are suitably partitioned in sets IT and ET, as described in Section 3.1.

We can now apply the algorithm presented in Section 3.1, to carry on the first step of the modeling process.

For simplicity let us denote as T the **Text**,L the **Lexer**, P the **Parser**, S the **Semantor**, O the **Optimizer** and G the **Generator**.

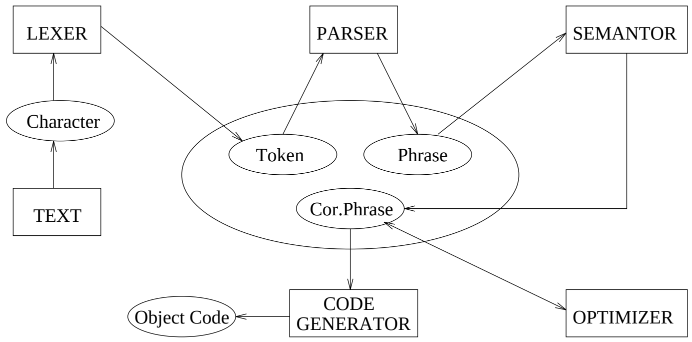


FIGURE 4. Multiphase Compiler Architecture with Repository

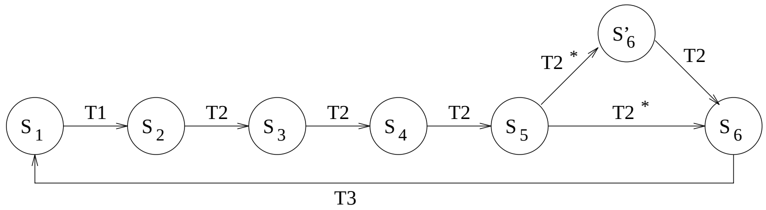


FIGURE 5. LTS of the Sequential Multiphase Compiler

I_1	I_2	I_3	I_4	I_5	I_6
(T,L)	(L,P)	(P,S)	(S,O) (S,G)	(O,G)	(G,ENV)

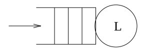
TABLE 1. Output of the algorithm Phase 1 for the Sequential Multiphase Compiler

For PHASE 1, we have as input the sets P={ T, L, P, S, O, G }, $PaC = \emptyset$ and the LTS of Figure 5, where IT = { T_1, T_2 }, ET = { T_3 },
The execution of PHASE 1 goes as follows: first we add the ENV element to P because ET is not empty, i.e. $P = P \cup \{ENV\}$. Then we visit the LTS and obtain as output the set of interaction sets I_i, $1 \leq i \leq 6$, shown in Table 1.
PHASE 2 receives as input sets P' = {L,P,S,O,G}; A = {T,ENV}; $PaC = \emptyset$ and the set of interaction sets $I = \{I_1, I_2, \ldots, I_6\}$, where $P' = P - A$ and A contains the components representing the exogenous customers and the environment, in this case Text and ENV. PaC is empty as in PHASE 1 and I contains the sets I_i shown in Table 1.

We shall now sketch the application of PHASE 2 that incrementally builds the model elements, as described in the Appendix, where we introduce some definitions with their graphical representation.

We synthesize PHASE 2 as follows:

- from I_1 it is generated the exogenous arrivals service center CdS_e(L) and the partial path CP([T, L]);



[T L]

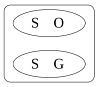
- from I_2 the complex server SC(L, P) and the partial path CP([L, P]);

[L P]

- from I_3 the complex server SC(P, S) and the partial path CP([P, S]);

[P S]

- from I_4 the multiclass server SM((S, O)(S, G)) and the partial paths CP([S, O]) and CP([S, G]);



[S O]
[S G]

- from I_5 the complex server SC(O,G) and the partial path CP([O, G]);

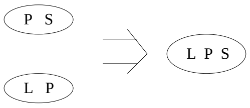
[O G]

- from I_6 the passing of resulting data from G to the environment.

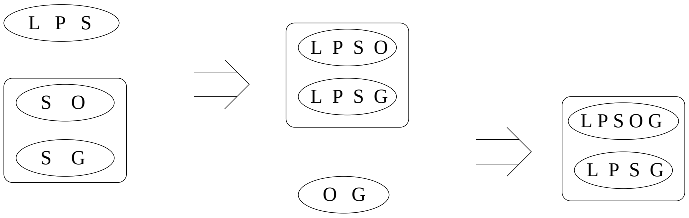
$$\mathrm{G} \longrightarrow ENV$$

Then we perform merging operations in order to reduce the number of model components and define the final model topology. The set of all the merging operations is listed in the Appendix.

- merge all the compatible complex servers; from SC(L, P), SC(P, S) and SC(O, G) we derive the complex servers SC(L, P, S) and SC(O, G)

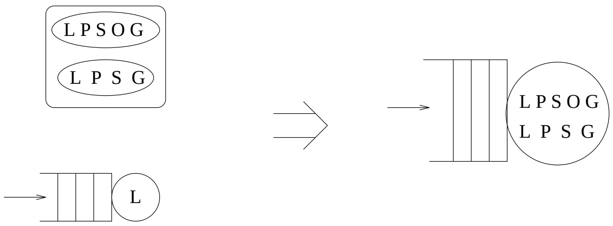


- merge all the compatible complex and multiclass servers; from the complex servers SC(L, P, S) and SC(O, G) and from the multiclass server SM((S, O)(S, G)) we generate multiclass server SM((L, P, S, O, G)(L, P, S, G)).



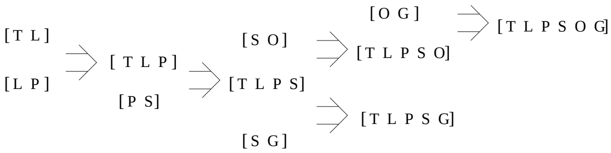
The merging operations "merge all complex servers with service centers", "merge all multiclass servers with service centers" and "merge all complex servers with exogenous arrivals service centers" have no effect since we do not have any other service center or complex server;
Then we perform the following operation:

- merge the multiclass servers and the exogenous arrivals service centers, thus generating from CdS_e(L) and SM((L, P, S, O, G)(L, P, S, G)) the exogenous arrivals service center CdS_e((L, P, S, O, G)(L, P, S, G)).



After merging the structures which define the QNM topology, we perform the merging operations among paths, that allow the characterization of the service type.

- merge all the compatible partial paths, from partial paths CP([T, L]), CP([L, P]), CP([P, S]), CP([S, O]), CP([S, G]) and CP([O, G]) to generate partial paths CP([T, L, P, S, O, G]) and CP([T, L, P, S, G]).



After the analysis of the I_i sets and merging operations we have to eliminate all the multiclass server in order to obtain a single class model, that allows more efficient solution methods.

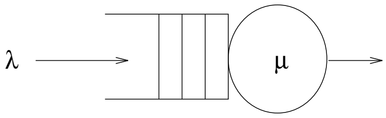


FIGURE 6. QNM for the Sequential Multiphase Compiler architecture

The resulting model is shown in figure 6. It is a very simple open queueing model with a unique service center that offers to the exogenous arrivals a unique service whose mean service time corresponds to a weighting average of two types of service times. One type of service time represents the execution of the five compilation phases, optimization included; the other type of service corresponds to the execution of the four compilation phases without optimization. The weights in the average allow emphasizing which offered services is more required.

The two tuples [T, L, P, S, O, G] and [T, L, P, S, G] characterize the two types of services, they state that a customer requires services to the components in the specified order in the tupla. The first element of both tuples is an element in A reflecting that the network is open. We assume $FCFS$ discipline.

4.2. **The model for the intermediate architecture.** Now let us consider the intermediate architecture of multiphase compiler, in order to see what happens if there are concurrent and synchronized components. Figure 7 shows a portion of the LTS specification of this architecture.
We can now apply the algorithm presented in Section 3.1, to carry on the first step of the modeling process.

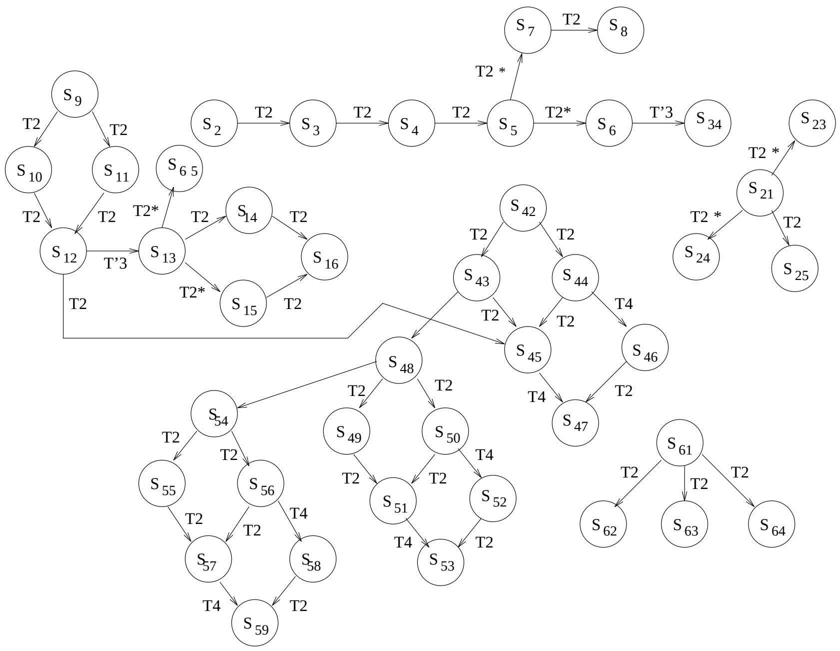


FIGURE 7. Portion of the LTS of the Intermediate Multiphase Compiler

I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
(T,L)	(L,P)	(P,S)	(S,O)* (S,G)*	(G,ENV)	(O,G)	(T,L) (P,S)	(L,P) (S,G)* (S,O)*
I_9	I_{10}	I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}
(T,L) (S,G)* (S,O)*	(T,L) (O,G)	(T,L) (G,ENV)	(L,P) (O,G)	(L,P) (G,ENV)	(P,S) (O,G)	(P,S) (G,ENV)	(T,L) (P,S) (O,G)

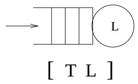
TABLE 2. Output of algorithm Phase 1 for the Intermediate Multiphase Compiler

PHASE 1 has input P={ T, L, P, S, O, G }, IT = { T_2, T_3, T_3' }, ET = { T_4 }, $PaC = \emptyset$ and the LTS of Figure 7. In PHASE 1 first we add ENV element to P because ET is not empty, i.e. $P = P \cup \{ENV\}$. Then we visit the LTS to obtain the set of interaction pairs shown in Table 2, that is the output of the first phase from which the second phase may be applied.

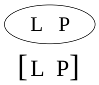
PHASE 2 inputs are $P' = \{$L, P, S, O, G$\}$, A ={T,ENV}, $PaC = \emptyset$ and the set of interaction sets $I = \{I_1, I_2, \dots, I_{16}\}$, where $P' = P - A$ and A contains the components representing exogenous customers and the environment, that are Texts and ENV. PaC is empty as in PHASE 1 and I contains the sets I_i shown in Table 2.

PHASE 2 generates the following model elements:

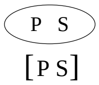
- from I_1 it generates the exogenous arrivals service center CdS_e(L) and partial path CP([T, L]);



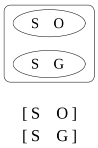
- from I_2 the complex server SC(L, P) and partial path CP([L, P]);



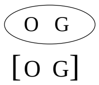
- from I_3 the complex server SC(P, S) and partial path CP([P, S]);



- from I_4 the multiclass server SM((S, O)(S, G)) and partial paths CP([S, O]) and CP([S, G]);



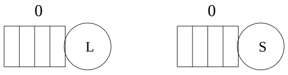
- from I_5 the complex server SC(O,G) and partial path CP([O, G]);



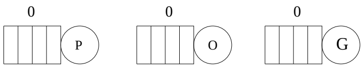
- from I_6 the passing of resulting data from G to the environment.

$$\mathrm{G} \longrightarrow ENV$$

- from I_7 the service center with zero capacity buffer $CdS_1(L)$ and the set $CONC$ ={ {L,P},{L,S} }.



- from I_7 the service centers with single capacity $CdS_1(P)$, $CdS_1(G)$ and $CdS_1(O)$, while set $CONC$ becomes $CONC = CONC \cup \{\{P,G\},\{L,G\},\{L,S\},\{P,S\},\{P,O\},\{L,O\}\}$.



The remaining interaction sets build copy of the elements above, while set $CONC$ becomes $CONC$ = {{L,P},{L,S},{L,O},{L,G},{P,S},{P,O},{P,G}, {S,O},{S,G}}. Then the algorithm analyzes each CdS_1 in order to single out those actually needed.

- Delete CdS_1(L), because there are not concurrent components that communicate with it, i.e. there is no pair in $CONC$ with component L as second element.
- CdS_1(P), delete SC(L,P) and build L $\longrightarrow$ P, because {L,P} $\in CONC$.
- CdS_1(S), delete SC(P,S)and build P $\longrightarrow$ S, because {P,S} $\in CONC$.
- CdS_1(O), delete SM((S,O),(S,G)) and build S $\longrightarrow$ O and SC(S,G), because {S,O} $\in CONC$
- CdS_1(G), delete SC(S,G) and SC(O,G) and build S $\longrightarrow$ G and O $\longrightarrow$ G, because {S,G},{O,G} $\in CONC$.

The resulting QNM, shown in Figure 8, is an open single class network with five service centers with FIFO service discipline. The first service center has external arrivals and an infinity capacity buffer, while the other ones have zero capacity buffer and we assume BAS blocking to model synchronous communication of concurrent components.

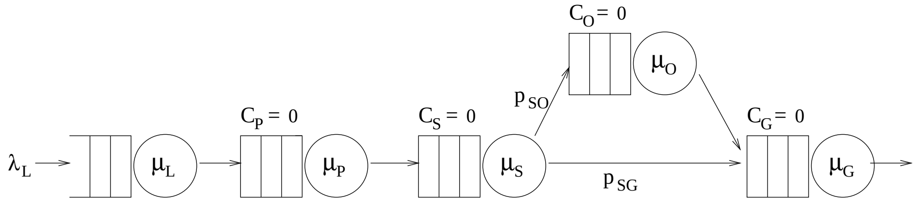


FIGURE 8. QNM for the Intermediate Multiphase Compiler architecture

4.3. **The model for the concurrent architecture.** The technique to derive the model for the multiphase concurrent compiler architecture carries on analogously to the previous case, thus we only summarize the resulting model. We assume a FIFO discipline at each service center.

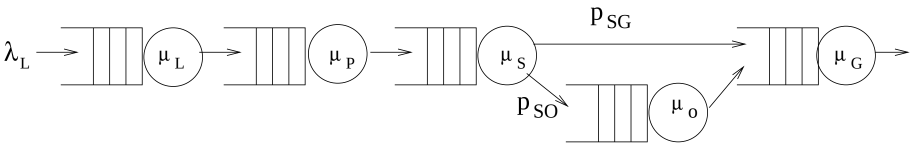


FIGURE 9. QNM for the Multiphase Compiler Architecture with Repository

The QNM is formed by five service centers and its topology is shown in Figure 9. From the algorithm we obtain two tuples [T, L, P, S, O, G] and [T, L, P, S, G] that characterize two service types as in the sequential architecture. The resulting model is an open single class network.

4.4. **Parameterization.** In order to complete the definition of the QNM obtained by the previous step, we have to specify some parameters. In particular we have to characterize the workload and define the arrival process and the service process of each service center of the model, that is we have to define the inter-arrival and the service time distribution and their parameters. We consider as system parameters the arrival rate and the mean service time of each service center. Moreover we have to define the routing probabilities for the models of the intermediate and concurrent architectures, i.e., probabilities p_{SG} and p_{SO} in QNMs shown in Figures 8 and 9. Since we are at a very high level of abstraction we consider symbolic parameters. They will assume a relative meaning during the comparison with the other competing software architectures if the derived models can analyzed in symbolic way, or a defined one when they are instantiated.

For the Sequential Multiphase Compiler Architecture, the algorithm defines the mean service time of the service center of the QNM according to the weighting average of the two service types as described in section 4.1 We denote with :

- λ, mean arrival rate of the customers at the system;
- $\frac{1}{\mu_N}$, mean service time for a service of the type N (without optimization);
- $\frac{1}{\mu_O}$, mean service time for a service of the type O (with optimization);
- $\frac{1}{\mu} = \frac{1}{a\mu_N+(1-a)\mu_O}$, mean service time for the whole model, where a $\in$ [0,1] is a parameter.

For the concurrent architecture we define the parameters of the arrival process to the network and the service time for each service center as follows:

- λ_L, mean arrival rate of the customer to service center L;
- $\frac{1}{\mu_j}$, mean service time at the service center $j \in \{L, P, S, O, G\}$ of a customer.
- p_{SO} and p_{SG} routing probabilities that a customer goes from service center S to service center O and service center G, respectively.

Finally, for the intermediate architecture we have the same parameters defined for the concurrent architecture.

5. Evaluation

In this Section we perform the evaluation step of the proposed approach for the three multiphase compiler SAs. We have to completely specify the model parameters, i.e. the arrival time and service time distributions and rates and the routing probabilities. In order to derive a QNM with product form solution, we assume exponential service time distribution at each service center, FCFS service discipline, Poisson arrivals and independence between services and arrival times. Then we obtain product form QNMs of the sequential and concurrent architectures that can be easily analyzed and compared by symbolic evaluation, in terms of several performance indices. On the other hand, the QNM of the intermediate architecture has service centers with finite capacity nodes and blocking and it does not have product form solution. Then we evaluate the performance indices of this QNM by approximate analytical methods or by simulation. Therefore we can compare the intermediate SA with the other two SAs through numerical evaluation by instantiating the model parameters.

We shall now present the symbolic performance analysis and comparison of the sequential and concurrent SAs and we conclude with the analysis of the intermediate architecture.

5.1. **Symbolic Evaluation.** In this section we derive the symbolic evaluation of the QNMs of the sequential and concurrent architecture of the multiphase compiler.

We start analyzing the QNM for the sequential architecture shown in Figure 6. The network is stable if the unique node is stable, that is if the arrival rate is smaller than the service rate, i.e. $\lambda < \mu$. We assume that this condition holds. Let n, denote the random variable (r.v.) number of customers in the model and ρ=$\frac{\lambda}{\mu}$, the traffic intensity of the model, defined as the ratio between the mean arrival rate and the mean service rate. Given the hypothesis discussed above, we observe that our model can be evaluated by the M/M/1 queueing system whose simple closed-form solution is well-known in the literature [18, 19]. The number of customers n is geometrically distributed with parameter ρ and we can easily and symbolically evaluate several average performance indices as follows:

- mean number of customers in the system:

$$N = \frac{\rho}{1-\rho}$$

- system throughput:

$$X = \lambda$$

- mean response time of the system:

$$R = \frac{N}{\lambda} = \frac{1}{\mu(1-\rho)}$$

- utilization of the system:

$$U = \text{Prob}\{n > 0\} = \rho$$

For the concurrent architecture, by assumptions introduced above we obtain the product form QNM network shown in Figure 9. Then we can easily derive symbolic expressions of the average performance indices for each service center, by decomposing the network into five M/M/1 systems with appropriate parameters. The network model is stable if all the nodes are stable, that is if the following condition holds:

$$\lambda_L < min(\{\mu_j \mid j \in \{\text{L}, \text{P}, \text{S}, \text{G}\}\}, \frac{\mu_O}{p_{SO}})$$

Under this constraint we derive the model solution. Let n_j denote the r. v. number of customers at node $j \in \{\text{L}, \text{P}, \text{S}, \text{O}, \text{G}\}$, ρ_j the traffic intensity at service center defined as the ratio between the mean arrival rate and the mean service rate at j, where $\rho_j = \frac{\lambda_L}{\mu_j}$ for $j \in \{L, P, S, G\}$, $\rho_O = p_{SO}\frac{\lambda_L}{\mu_O}$ for $j = O$.

Similarly to the model of the sequential architecture, for the QNM of the concurrent architecture we obtain the following local performance indices for every node:

Performance Index	for node $j \in \{\text{L}, \text{P}, \text{S}, \text{O}, \text{G}\}$
Mean number of customers	$N_j = E[n_j] = \frac{\rho_j}{1-\rho_j}$
Throughput	$X_j = \lambda_L$ for $j \neq O$ and $X_O = p_{SO}\lambda_L$
Mean response time	$R_j = \frac{1}{\mu_j - \lambda_L}$ for $j \neq O$ and $R_O = \frac{1}{\mu_O - \lambda_L p_{SO}}$
Utilization	$U_j = \rho_j$

and the following average performance indices for the entire system:

Performance Index	Overall System
Mean number of customers in the system	$N = \sum_{j \in \{\text{L,P,S,O,G}\}} N_j = \sum_{j \in \{\text{L,P,S,O,G}\}} \frac{\rho_j}{1-\rho_j}$
System throughput	$X = \lambda_L$
Mean system response time	$R = \frac{N}{X} = \sum_{j \neq O} \frac{1}{\mu_j - \lambda_L} + \frac{p_{SO}}{\mu_O - \lambda_L p_{SO}}$

5.2. **Comparing the sequential and concurrent Software Architectures.** In this section we compare the sequential and concurrent software architectures in terms of the symbolic evaluation of the average performance indices obtained above. In order to compare the two QNMs one has to consider the same workload, i.e. the inputs to these models have to be equally specified. More precisely, we assume the same workload in the two models, given by customer arrival rates and total service demands.

Let us denote with:

- λ^s arrival rate of customers in the sequential model;
- λ^c arrival rate of customers in the concurrent model;
- $\frac{1}{\mu^s}$ mean service time of a customer in the sequential model;

- $\frac{1}{\mu_j^c}$ mean service time of a customer in center $j \in \{L, P, S, O, G\}$ in the concurrent model.

We assume the same customer arrival rate for the two models, i.e. $\lambda^c = \lambda^s = \lambda$. As far as the total service demand is concerned, we observe that a customer in the sequential architecture requires μ^s mean service time corresponding to a weighting average of N and O service types, while in the concurrent architecture the mean service time is the summation of the mean service times corresponding to the various service centers. Thus the mean service time for a customer that requires service type N is given by $\sum_{j\in\{L,P,S,G\}} \frac{1}{\mu_j}$, and $\sum_{j\in\{L,P,S,O,G\}} \frac{1}{\mu_j}$ if it requires service type O. Therefore the relation between the service demands of the two models is the following:

$$\frac{1}{\mu^s} = a(\frac{1}{\mu_L^c} + \frac{1}{\mu_P^c} + \frac{1}{\mu_S^c} + \frac{1}{\mu_O^c} + \frac{1}{\mu_G^c}) + (1-a)(\frac{1}{\mu_L^c} + \frac{1}{\mu_P^c} + \frac{1}{\mu_S^c} + \frac{1}{\mu_G^c}). \tag{1}$$

We can now compare the performance indices of the two models. To this purpose we present some examples by considering three different scenarios, which allow us to instantiate some of the quantitative measures.

1. The mean service times of the nodes in the concurrent model have the same degree of magnitude. This corresponds to assume that all the compilation phases in the SA perform comparable elaboration.
2. The mean service time at some center in the concurrent model is of the same degree of magnitude of the mean service time of the unique center in the sequential model. This corresponds to a system where one of the compilation phases is so expensive that the others become negligible.
3. The mean service times of the centers in the concurrent model range in an interval between a minimum and a maximum mean service time.

We shall now compare the performance indices of the two models in the given scenarios. Note that when we compare the software architectures in terms of throughput, due to the equal workload assumptions above, it is meaningless to simply compare the resulting measures. Indeed, this index directly derives from the mean arrival rates that are assumed identical in the sequential and concurrent models. Therefore the throughput index for the two models is identical. However, it is meaningful to compare the two models by considering the maximum throughput that they can support, provided that the stationary condition holds. Hereafter X_{seq} and X_{con} denote the maximum throughput in the sequential and concurrent model, respectively.

The stationarity condition for customers is $\lambda^s < \mu^s$ in the sequential model, and $\lambda^c < min(\{\mu_j \mid j \in \{L, P, S, G\}\}, \frac{\mu_O}{p_{SO}})$ in the concurrent model. Thus we obtain the maximum throughput as follows:

$$X_{seq} = \mu^s \tag{2}$$

in the sequential model and

$$X_{con} = min(\{\mu_j^c \mid j \in \{L, P, S, G\}\}, \frac{\mu_O^c}{p_{SO}}) \tag{3}$$

in the concurrent model. We immediately observe by equation (1) that the maximum throughput of the sequential model is less than that in the concurrent model, that is $X_{seq} < X_{con}$.

We compare the two software architectures in terms of some other average performance indices. We can compare the utilization of each server in the concurrent model with the utilization of the single service center in the sequential model.

The mean number of customers is a global performance index in the QNM of the sequential model, while we can derive both the local and the global index in the concurrent model. Hence in order to compare the two software architectures we consider the overall mean number of customers in both systems.

Finally, note that the model comparison in terms of average response time of the two models provides the same results as for the mean number of customers, because the two systems have the same throughput and by Little's theorem.

Let now consider us the performance comparison for the three scenarios introduced above.

1. **First scenario**.
From the hypothesis, we can write $\frac{1}{\nu} \simeq \frac{1}{\mu_j^c}$ for $j \in \{L, P, S, O, G\}$, where ν represents the common value of the mean service rates of service centers, up to a negligible approximation error. Hence for the concurrent model, by (3) it follows that $X_{con} = \nu$.

On the other side for the sequential model we can write $X_{seq} = a\,\frac{\nu}{5} + (1-a)\frac{\nu}{4}$. Thus, as expected, we obtain that:

(4) $$\frac{X_{con}}{X_{seq}} = \frac{20}{5-a}.$$

The utilization of node $j \in \{L, P, S, O, G\}$ in the concurrent model is given by $U^j_{con} = \delta_{jO}\frac{\lambda}{\nu}$ where $\delta_{jO} = p_{SO}$ if $j = O$ and $\delta_{jO} = 1$ if $j \neq O$.

For the sequential model we can write: $U_{seq} = \frac{\lambda}{\mu^s} = \frac{(a+4)\lambda}{\nu}$.

Hence, by comparing utilization U_{seq} with any U^j_{con} for $j \in \{L, P, S, O, G\}$, we can write:

(5) $$\frac{U^j_{con}}{U_{seq}} = \frac{\delta_{jO}}{(a+4)}$$

The average number of jobs in the concurrent model is given by:

$$N_{con} = \sum_{i \in \{L,P,S,G\}} \frac{\lambda}{\nu-\lambda} + \frac{p_{SO}\lambda}{\nu - p_{SO}\lambda} = \lambda\left(\frac{4}{\nu-\lambda} + \frac{p_{SO}}{\nu - p_{SO}\lambda}\right).$$

For the sequential model by formula (1) we derive $\frac{1}{\mu^s} = \frac{a+4}{\nu}$. Hence we can write the following mean number of jobs:

$$N_{seq} = \frac{\lambda}{\mu_s - \lambda} = \frac{(a+4)\lambda}{\nu - (a+4)\lambda}.$$

To compare N_{con} and N_{seq}, let us define the following function:

(6) $$Rapp(\lambda,\nu) = \frac{N_{seq}}{N_{con}} = \frac{(a+4)}{\frac{4(\nu-(a+4)\lambda)}{\nu-\lambda} + \frac{p_{SO}(\nu-(a+4)\lambda)}{\nu-p_{SO}\lambda}}$$

which is a non negative function, due to the stability condition in the sequential model.

To illustrate the meaning of the performance comparison given by formulas (4)-(6) consider for example the case of equally required service types N and O, that is $a \simeq 0.5$. Under this hypothesis, by formulas (4) and (5), we can say that the maximum throughput in the concurrent model is 4.5-times bigger than that of the sequential one and the node utilization in the concurrent model is about $\frac{2}{9}$ the utilization of the unique center in the sequential model. In other words the workload that leads the sequential model to its saturation is 4.5 times smaller than the workload that make the concurrent model saturated.

Consider now another example where we suppose that the service type N is more required than O, i.e. $a \longrightarrow 0$, then we can say that the maximum throughput in the concurrent model is about 4-times bigger than that of the sequential model and the utilization ratio is about one quarter. Vice versa if $a \longrightarrow 1$, then the maximum throughput in the concurrent model is about 5-times bigger than that of the sequential model and the utilization ratio is about one fifth.

By using the symbolic evaluation of the two models we can make some parametric performance study. For example we can study function *Rapp* to evaluate how the ratio of the mean number of jobs in the two models is affected by varying load parameters λ and ν. In particular by setting $Rapp = k$ for a given value k we identify the load and service parameters λ and ν that make the average queue length of the sequential model to be k time greater than that of the concurrent model. Consider the following system:

$$\frac{N_{seq}}{N_{con}} = \frac{(a+4)}{\frac{4(\nu-(a+4)\lambda)}{\nu-\lambda} + \frac{p_{SO}(\nu-(a+4)\lambda)}{\nu-p_{SO}\lambda}} = k \tag{7}$$

$$\nu, \lambda > 0 \tag{8}$$

$$\nu > \lambda \tag{9}$$

We observe that $k \geq 0$ because of the system stability condition, that means that the average number of jobs in the sequential model is k times greater than that in the concurrent model. Indeed, from the stability condition of the sequential model ($\lambda < \frac{a+4}{\nu}$) if $\lambda \longrightarrow \frac{a+4}{\nu}$ then $k \longrightarrow \infty$ and the average number of jobs in the sequential models is unlimited. By summarizing for this scenario from the quantitative symbolic comparison of the two software architectures we observe that:

- the maximum throughput in the concurrent model is 5-times greater than that of the sequential model, if service O is prevailing (4-times if service N is prevailing and 4.5-time if two services are equally required).
- this corresponds to the observation that the maximum workload for the sequential model is five times smaller than that of the concurrent model (four time smaller if service N is prevailing and 4.5 time smaller if two services are equally required).
- relationship (7) provide a tool to select system parameters λ (the workload) and ν (the average service rate), in order to observe the average queue length and the average response time in the sequential models k-times greater than that of the concurrent model.

2. **Second scenario**.
From the hypothesis we can write for a given center, say L:

$$\frac{1}{\mu^s} \simeq \frac{1}{\mu^c_L} \tag{10}$$

$$\frac{1}{\mu^c_j} \ll \frac{1}{\mu^c_L} \quad j \in \{\mathrm{L, P, S, O, G}\} \tag{11}$$

Hence we obtain $X_{con} = \mu^c_L$ and $X_{seq} = \mu^s$. By formula (10) we immediately observe that the two models have nearly the same maximum throughput.

From the assumptions the utilization for a given center, say L, is greater than that of the other nodes for each class. In other words the utilization of the other centers in the concurrent model is negligible with respect to that of center L and we can write $U_{con} \simeq U^L_{con} = \frac{\lambda}{\mu^c_L}$. For the sequential model we obtain the following utilization: $U_{seq} = \frac{\lambda}{\mu^s}$.

By comparing the class utilization of the two models and by formula (10) we obtain an identity. Hence we conclude that the two models for each class have the same critical workload.

We observe that also the mean number of jobs in a given center, L, is greater than that in the other centers of the model. By condition (11) we can simply compare the two models by considering center L and the unique center of the sequential model. So we can write:

$$N_{con} = \sum_{i \in \{P,S,O,G\}} N^i_{con} + N^L_{con} \simeq N^L_{con} = \frac{\lambda}{\mu^c_L - \lambda}, \quad N_{seq} = \frac{\lambda}{\mu^s - \lambda}$$

which by formula (10) yield the same value, that is the two models have the same average number of jobs.

By summarizing for the second scenario the quantitative analysis of the two software architecture states that:

- the two models have nearly the same maximum throughput, utilization, and the same critical workload; the two models have the same mean queue length and mean response time for each type of customers.

3. **Third scenario.**

From the hypothesis we can write for a node j in the concurrent model:

$$l \leq \mu^c_j \leq u \quad j \in \{\mathrm{L, P, S, O, G}\} \tag{12}$$

where

$$l = min\{\{\mu^c_j \mid j \in \{\mathrm{L, P, S, G}\}\}, \frac{\mu_O}{p_{SO}}\} \text{ and } u = max\{\{\mu^c_j \mid j \in \{\mathrm{L, P, S, G}\}\}, \frac{\mu_O}{p_{SO}}\}$$

By formula (3) this leads to $X_{con} = l$.

Substituting relation (1) in formula (12) we have:

$$\frac{a+4}{u} \leq \frac{1}{\mu^s} \leq \frac{a+4}{l} \tag{13}$$

which leads to $\frac{l}{a+4} \leq \mu^s \leq \frac{u}{a+4}$.

Hence, by comparing X_{seq} and X_{con} we immediately obtain

$$\frac{(a+4)l}{u} \leq \frac{X_{con}}{X_{seq}} \leq (a+4).$$

For this scenario the utilization of node j in the concurrent model satisfies the following constraints:

$$\frac{\lambda}{u} \leq U^j_{con} \leq \frac{\lambda}{l} < 1 \quad j \in \{\mathrm{L, P, S, O, G}\}.$$

For the sequential model we derive the utilization U_{seq} by formula (13) and by definition as follow:

$$\frac{(a+4)\lambda}{u} \leq U_{seq} \leq \frac{(a+4)\lambda}{l} < 1.$$

The mean number of jobs in the concurrent model satisfies the following constraints:

$$4\frac{\lambda}{u-\lambda}+\frac{p_{SO}\lambda}{u-p_{SO}\lambda}\leq N_{con}\leq 4\frac{\lambda}{l-\lambda}+\frac{p_{SO}\lambda}{l-p_{SO}\lambda}.$$

On the other hand for the mean number of jobs in the sequential model, from formula (13) and by N_{seq} definition we can write:

$$\frac{(a+4)\lambda}{u-(a+4)\lambda}\leq N_{seq}\leq\frac{(a+4)\lambda}{l-(a+4)\lambda}.$$

By comparing N_{con} with N_{seq} constraints we obtain

$$\frac{\frac{(a+4)\lambda}{u-(a+4)\lambda}}{4\frac{\lambda}{u-\lambda}+\frac{p_{SO}\lambda}{u-p_{SO}\lambda}}\leq\frac{N_{seq}}{N_{con}}\leq\frac{\frac{(a+4)\lambda}{l-(a+4)\lambda}}{4\frac{\lambda}{l-\lambda}+\frac{p_{SO}\lambda}{l-p_{SO}\lambda}}. \tag{14}$$

From this formula we can say that the maximum throughput in the concurrent model is k-times bigger than that of the sequential model where $k\in[\frac{(a+4)l}{u},(a+4)]$ and $(a+4)\in[4,5]$ and this correspond to an utilization U_{seq} that is k-times greater than that of any center j in the concurrent model U^j_{con}, where $k\in[\frac{(a+4)\lambda}{u},\frac{(a+4)\lambda}{l}]$ and $(a+4)\in[4,5]$.

Relationships (14) provide an interval for the ratio of the mean number of customers in the two models as a function of system parameters (arrival an service rates). This ratio shows how bigger is the average number of customers in the sequential model with respect to that of the concurrent model.

To summarize the software performance comparison of this scenario, where the service rates of the nodes of the concurrent model are in the range $[l,u]$, yields to the following observations:

- the maximum throughput in the concurrent model is k-times bigger than that of the sequential model where $k\in[\frac{(a+4)l}{u},(a+4)]$.
- The utilization of any center j in the sequential model is k-times greater than that of the concurrent model where $k\in[\frac{(a+4)\lambda}{u},\frac{(a+4)\lambda}{l}]$.
- The relationship between the sequential and concurrent models in terms of mean queue length and mean response time is expressed by formula (14) that provides an interval for the ratio of the mean number of customers in the two models as a function of system parameters, the arrival an service rates. We can evaluate the relative performance improvement of the concurrent model over the sequential model by this ratio, with respect to the average queue length and average response time. In the development process of the software project this relation can be applied to determine the appropriate system workload and its distribution (which define the arrival and service rates in the performance model) to meet a predefined performance requirement in terms of average queue length or response time.

5.3. **Analysis of the intermediate architecture.** The QNM of the intermediate architecture shown in Fig. 8 has service centers with finite capacity nodes and BAS blocking introduced to model synchronous communication between concurrent components. This type of QNM does not have a product form solution and can be analyzed by approximate analytical methods or by simulation. Approximation algorithms are mostly based on network decomposition and iterative composition of the analytical solution of simple models formed by few service centers with appropriate parameters [24]. In particular the acyclic QNM with BAS blocking of Fig. 8 can be analyzed by an approximate analytical method that decomposes the network into single nodes analyzed in isolation as M/M/1 systems with finite capacity, whose parameters are iteratively evaluated. Then for this performance model and in general for QNM without product form solution, we cannot derive a symbolic evaluation of the performance indices because of the iterative computation the approximation algorithm.

In order to compare the intermediate SA with the other two SAs we must perform a numerical evaluation by instantiating the model parameters. Such evaluation can be carried on with performance analysis tools, to apply an approximate analytical method or simulation. An extensive numerical comparison of the intermediate architecture with the other two models is out of the scope of this paper.

Performance evaluation of the SA consists in defining variability ranges of model parameters and implementation scenarios that guide the values choice assigning to model parameters.

For every scenario, we evaluate a set of performance indices and we can interpolate on these values to obtain a numerical evaluation of the index trend within defined ranges. For example consider a scenario where all architecture components have comparable mean service time. Then the value assigned to mean service times of the service centers of the network have to be of the same degree of magnitude. If we consider, instead, a scenario in which there is a bottleneck component, then the mean service time of the corresponding service center must have a major weight respect to mean service times of other service centers.

In particular, if there are in the model joint service centers, i.e. service centers with a mean service time defined as the weighting average of mean service times, then we can define two alternatives: uniform weight distribution and a weight more relevant than the others. Then the model evaluation is carried on for each alternative. This corresponds to compare the architecture behavior when there is a uniform or balanced service requirement and unbalanced one, i.e. when a given service is more required than the others.

Finally, note that performance comparison has to be carried on between models under comparable conditions, such as system load and other parameters. Therefore, for example when we compare models with a different number of parameters, to make the comparison meaningful we have to determine some of them and compare performance indices by varying the others.

6. Conclusions

In this paper we have presented an approach to evaluate the performance of software architecture. The method allows the automatic derivation of a QNM from a SA dynamic model based on a Labeled Transition System. The model can then be evaluated either symbolically or by simulation and the results can be used to validate the architectural design choices against system performance requirements. The method allows for reasoning at the architecture level of performance aspects thus providing the designer with a rigorous, although necessarily high level, framework to derive subsequent design decisions. If the results obtained are positive, our method allows for carrying on the system development steps by providing suitable refinements of the system performance requirements. In the compiler example we showed, the choice of the concurrent compiler architecture against the sequential one, comes with a set of performance requirements that should hold at subsystem level in order to guarantee the expected system performance. In the negative case, our methodology allows for an early in the system development cycle detection of design problems. The goal of our approach is twofold: on one side we want to be able to evaluate performance indices of the SA description at its high level of abstraction. On the other side we want to be able to derive from the performance analysis substantial feedback on the design process. These two goals have driven us in the choice of the SA behavioral model and of the SA performance model. As we already discussed in the related work Section, our method substantially differs from the ones presented in the literature. Future research will go in several directions. We are working on applying our methodology to more significant case studies. In particular we are addressing two case studies coming from the telecommunication domain with a close collaboration with the interested industries. From these studies we are exploring the possibility of using more conventional design formalisms, like UML as ADL. Our experience with modeling one of them [17] has suggested the idea of using Message Sequence Charts (MSC) instead of a complete LTS to model the dynamic behavior of SA [1]. In this way we can avoid the state explosion problem and use standard design artifact like MSC to model the system dynamics from which apply our QNM derivation algorithm. Finally we are working on automatizing our methodology trying to integrate architectural design tools with performance evaluation tools.

So far, we have actually used separate tools to perform the various steps. From our experience we have found this approach encouraging and stimulating both in terms of design and analysis support.

References

1. F. Andolfi, F. Aquilani, S. Balsamo, P. Inverardi "Deriving Performance Models of Software Architectures from Message Sequence Charts" in [37]
2. G. Balbo, G. Conte and M. A. Marsan. Performance Models of Multiprocessor Systems. Series in Computer Systems, The MIT Press, (1986).
3. S. Balsamo "Properties and Analysis of Queueing Network Models with Finite Capacities" in 'Performance Evaluation of Computer and Communication Systems (L.Donatiello and R. Nelson Eds.), LNCS (Springer-Verlag), No. 729, (1994) 21-52.
4. S. Balsamo, P. Inverardi, C. Mangano, "An Approach to Performance Evaluation of Software Architectures" in [36].
5. S. Balsamo, C. Mangano, P. Inverardi, F. Russo, "Performance Evaluation of a Software Architecture: A Case Study" in IEEE Proc. 11th Int. Workshop on Software Specification and Design, Japan, 1998.
6. L. Bass, P. Clements, R. Kazman "Analyzing Development Qualities at the Architectural Level" In Software Architectures in Practice. SEI Series in Software Engineering, Addison-Wesley 1998.
7. M. Bernardo and R. Gorrieri "Extend Markovian Process Algebra" In Proc. CONCUR '96, LNCS (Springer-Verlag) No. 1119, (1996) 315-330.
8. G. Bockle, H. Hellwagner, R. Lepold, G. Sandweg, B. Schallenberger, R. Thudt, S. Wallstab "Structured Evaluation of Computer System" IEEE Computer Society, 45-51, June 1996.
9. M. Bernardo, P. Ciancarini and L. Donatiello, "AEMPA: A Process Algebraic Description Language for the Performance Analysis of Software Architectures", in [37].
10. D. Compare, P. Inverardi and A.L. Wolf "Uncovering Architectural Mismatch in Dynamic Behavior" Science of Computer Programming , January 1999.
11. V. Cortellessa, R. Mirandola, "Deriving a Queueing network based perofrmance Model from UML Diagrams" in [37].

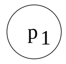
12. R. Allen and D. Garlan "A Formal Basis for Architectural Connection" ACM Transactions on Software Engineering and Methodology, 6(3): 213–249, July 1997.
13. D. Giannakopoulou, J. Kramer, and S.C. Cheung " Analysing the behaviour of distributed systems using tracta" Automated Software Engineering, special issue on Automated Analysis of Software, 6(1): 7–35, January 1999.
14. H. Gomaa and D. Menasce, "A Method for Design and performance modeling of client-server Systems", to appear in IEEE Transactions on Software Engineering, 2000.
15. H. Gomaa and D. Menasce, "Design and Performance Modeling of component Interconnection Patterns for Distributed Software Architectures" in [37].
16. P. Inverardi 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.
17. P. Inverardi, L. Marcucci, H. Muccini "Formal and Semi-formal descriptions of a Cross-Connector Software Architecture" Technical Report Universita' dell'Aquila, September 2000, submitted for publication.
18. K. Kant. Introduction to Computer System Performance Evaluation. McGraw-Hill, 1992.
19. L. Kleinrock. Queueing Systems. Vol. 1: Theory. (Wiley 1975).
20. 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, (1984).
21. F. Aquilani "Valutazione delle Prestazioni di Architetture Software" Laurea Thesis in Computer Science, University of L'Aquila, December 1999.
22. R. O. Onvural "Survey of Closed Queueing Networks with Blocking" ACM Computing Surveys, Vol. 22, No. 2, 83-121, June 1990.
23. D. E. Perry e A. L. Wolf "Foundations for the Study of Software Architecture" ACM SigSoft Software Engineering Notes, Vol. 17, No. 4, 40-52, October 1992.
24. H. G. Perros. Queueing Network with Blocking: Exact and Approximate Solutions. Oxford University Press (1994).
25. D. C. Petriu, X. Wang "From UML Description of high-level software architecture to LQN Performance Models", in AGTIVE'99, LNCS 1779, Springer-verlag, 2000.
26. D. C. Petriu, C. Shousha, A. Jalnapurkar, "Architecture-based Performance Analysis Applied to a Telecommunication System", to appear in IEEE Transaction of Software Engineering, 2000.
27. R. Pooley, "Software Engineering and Performance: A Road-map" in The Future of Software Engineering, A. Finkelstein Editor, 22 International Conference on Software Engineering.
28. R. Pooley and P. King, "The Unified Modeling Language and Performance Engineering" IEE Proceedings-Software, 146, 1 (February 1999).
29. R. Pooley and P. King, "Using UML to derive stochastic process algebra models" Proceedings 15th UK Performance Engineering Workshop, 1999.
30. P. King and R. Pooley, "Derivation of Petri Net Performance Models from UML specifications" Proceedings 11th Int. Conf. on Tools and techniques for computer Performance Evaluation, Illinois 2000.
31. M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall (1996).
32. C. U. Smith. Performance Engineering of Software Systems. Addison-Wesley Publishing Company, (1990).
33. C. U. Smith 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.
34. C. U. Smith and L. G. Williams "Performance Evaluation of Software Architectures" in [36].
35. C. Hofmeister, R. Nord and D.Soni, "Applied Software Architecture", Addison-Wesley, Object Technology Series, 1999.
36. IEEE Proceedings of the First International WOrkshop on Software and Performance, Santa Fe', October 1998.
37. IEEE Proceedings of the Second International WOrkshop on Software and Performance, Ottawa, September 2000.
38. M. Woodside, C. Hrischuk, B. Selic, S. Bayarov, "A Wideband Approach to integrating Performance prediction into a Software Design Environment", in [36].

39. M. Woodside " Software Performance Evaluation by Models", in Performance Evaluation (G. Haring, C. Lindemann, M. Reiser Eds.), LNCS 1769, 283-304, 2000.

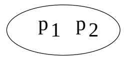
Appendix

In order to identify the model elements that are incrementally built by the algorithm, we introduce some definitions with their graphical representation.
DEFINITIONS

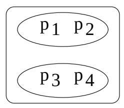
- Simple Server $SS(p_1)$, it provides a service performed by component p_1.



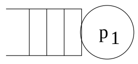
- Complex Server $SC(p_1, \ldots, p_n)$, it provides as a unique service the set of services performed by simple severs $SS(p_1), \ldots SS(p_n)$, in the given order. Its average service time results from the summation of the average service times of the simple servers.



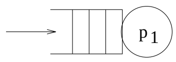
- Multiclass Server, $SM((p_1, p_2), (p_3, p_4))$, it provides different services (in this case two) where the user chooses one of the services associated to the two complex servers. The illustrated SM can be generalized with more than two SC, each one with more than two elements.



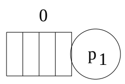
- Joint Simple Server, $SS_{med}(p)$. It enables to eliminate the multiclass servers, in order to obtain single class models.
- Service Center, $CdS(p_1)$, it provides the service offered by component p_1; differently from the simple server it has an infinite buffer capacity to queue users waiting for services.



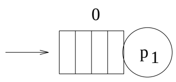
- Service Center with external arrival, $CdS_e(p_1)$, it is like $CdS(p_1)$, but it allows external arrivals.



- Service Center with unit capacity, $CdS_1(p_1)$, it has a zero capacity buffer to model synchronization.



- Service Center with unit capacity and external arrival, $CdS_{e1}(p_1)$. It is like Service Center with exogenous arrival, but with a zero capacity buffer.



- Compulsory Service Center with unit capacity, $CdS_{e1}(p_1)$, it models broadcast communication.

- Connecting Service Center, $CdS_c(p_1)$. It indicates a connecting element with infinite capacity buffer and a negligible service time.
- Transition from a to b, $a \longrightarrow b$, a user leaving a goes to b, where a can be a simple server, a service center or a service center with exogenous arrivals, and b can be a service center or a service center with exogenous arrivals.
- Temporary Right Transition, $p_1 \longrightarrow_{trt} p_2$. It indicates a transition of data from component p_1 to connecting element p_2 (right element of the transition).
- Temporary Left Transition, $p_1 \longrightarrow_{tlt} p_2$. It indicates a transition of data from connecting element p_1 to component p_2 (left element of the transition).
- Partial Path, $CP(p_1, \ldots, p_n)$ indicates a possible history of user service requests to the system components.

We shall now provide a scheme of the algorithm that derive the QNM from a SA specification by a LTS, as described in Section 3.1.

PHASE 1
INPUT:

$P = \{$ SA components $\}$
$G = (S, A, T)\{$ labeled directed graph associated to the LTS $\}$
$T = IT \bigcup ET\{$ partition of T in internal and external transitions $\}$
$PaC = \{$ Passive Connecting Elements $\}$

OUTPUT:

$I = \{I_1, I_2, \ldots\}$ where $I_i = \{$ i-th Interaction Pairs Set $\}$
Let E denote the set of nodes to be examined and V the set of visited nodes.
At the beginning $E = \{S_1\}$ and $V = \emptyset$ and at the end $E = \emptyset$ and $V = S$.

Begin
  If $ET \neq \emptyset$ **Then**
    $P = P \bigcup \{ENV\}$;
  $E = \{S_1\}$; $V = \emptyset$ $i = 1$;
  While $E \neq \emptyset$ **Do**
    $<$ Extract a node S_j from set E and move it to set V of the visited nodes $>$
    If $\mid I_i \mid > 0$ **Then** $i = i + 1$;
    $<$ define the set $AS(S_j)$ of Adjacent Nodes of (S_j);
    While $AS(S_j) \neq \emptyset$ **Do**
      $<$ Extract a node S'_j from $(AS(S_j))$;
      If $S'_j \notin V$ **Then** $<$ Insert S'_j in $V >$
      $<$ Examine transition label T_k associated to $(S_j, S'_j) >$
      If $T_k \in IT$ **Then**
        $<$ Identify the pair of elements $p_1, p_2 \in P$ related to $T_k >$;
        $<$ Build the interaction pair $IP = (p_1, p_2) >$;
      If $T_k \in ET$ **Then**
        $<$ Identify element $p_1 \in P$ related to $T_k >$;
        $<$ Build the interaction pair $IP = (p_1, ENV) >$;
      $I_i = I_i \bigcup IP$;
      If Marked(T_k) **Then** $<$ Mark the pair $IP >$;
      EndDo
  EndDo
End

The complexity of this phase is $O(N\ a)$ where N is the number of nodes of the LTS, a is the maximum number of adjacent nodes.

PHASE 2
INPUT:
$A = \{$ SA components modeled as external ones in the QNM $\}$
$P' = \{$ SA components modeled as internal ones in the QNM $\}$
$PaC = \{$ Passive Connecting Elements $\}$
$I = \{I_1, I_2, \dots\}$
OUTPUT:
$QNM =$ Queueing Network Model
Begin
For all i **Do**
For all $(p_1, p_2) \in I_i$ **do**
$<$ Examine p_1 and p_2 and build the appropriate component, that can be a Service Center $CdS_e(p_2)$ or $CdS(p_2)$ or a Complex Server $SC(p_1, p_2)$; build the partial path $>$
$<$ In each case determine the sets of pairs with common components, considering marked and unmarked cases $>$
$<$ For each pair in the sets build the appropriate model components and partial paths; service centers can be either with unit capacity, $CdS_1(p_3)$ for another component p_3, Connecting Service Centers $CdS_c(p_3)$ or Multiclass Servers SM $>$
$<$ Determine the set $CONC$ of concurrent pairs $>$
$<$ Examine all the built Connecting Service Centers $CdS_c(p)$ and if appropriate delete the temporary right or left transitions and update the partial paths $>$
$<$ Examine all the Multiclass Servers $SM((p_2, p_3), \dots, (p_2, p_k))$ and if appropriate by analyzing set $CONC$, built the appropriate Complex Server(s) $SC(p_2, p_j)$, delete the Multiclass Server and update the partial paths $>$
$<$ Examine all the Complex Servers $SC(p_2, p_3)$ and if appropriate by analyzing set $CONC$ built the appropriate transition from p_2 to p_3 and delete the Complex Server $>$
$<$ If appropriate by analyzing the set $CONC$, delete the Service Center with unit capacity $CdS_1(p_1)$; $>$
$<$ If appropriate merge the Service Center with unit capacity $CdS_1(p_1)$; and Service Center with external arrivals $CdS_e(p_1)$; in a unique Service Center $Cds_{e1}(p_1)$; $>$
$<$ If appropriate merge temporary right and left transitions into direct transitions $>$
EndDo;
EndDo;

$<$ Merge all compatible Complex Servers $>$
$<$ Merge all compatible Multiclass Servers $>$
$<$ Merge all compatible Complex and Multiclass Servers $>$
$<$ Merge all compatible Complex Servers and Service Centers $>$
$<$ Merge all compatible Multiclass Servers and Service Centers $>$
$<$ Merge all compatible partial path $>$
$<$ Merge all Multiclass Servers $SM((p_1, \dots, p_i), \dots, (p_1, \dots, p_j))$ in a Joint Simple Server $SS_{med}(p)$ with appropriate average service rate $>$
End.

The complexity of this phase is $O(n_i\, N^6)$ where n_i is the number of sets I_i, and N the number of the architecture components.