AN APPROACH TO PERFORMANCE EVALUATION OF SOFTWARE ARCHITECTURES

FEDERICA AQUILANI, SIMONETTA BALSAMO*, PAOLA INVERARDI

ABSTRACT. In this paper we present an approach to evaluating the expected performance of a software architecture. The method automatically derives a performance evaluation model, based on a Queueing Network Model, from a software architecture specification formally described in CHAM (CHemical Abstract Machine). The goal of this approach is to provide a quantitative performance comparison of two or more competing software architectures, even at their high level of abstraction, i.e. to derive mathematical relations among the parameters — the performance indices — characterizing the competing models. 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 relations provide further requirements to be fulfilled by an implementation in order to satisfy the chosen performance criteria.

1. INTRODUCTION

Software Architectures (SA) describe software system structures at a high level of abstraction [14, 16]. They represent at an early stage of development the phase in which basic choices of components and interactions among components are made. Ideally, those choices reflect any constraints imposed by the hardware architecture upon with the system is to run.

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. In this paper we present an approach to evaluating the expected performance of a software architecture. The method automatically derives a performance evaluation model, based on a Queueing Network Model [7, 8, 9], from a software architecture specification formally described in CHAM (CHemical Abstract Machine)[5, 6]. Due to the high level of abstraction of SA, the evaluation of the model can only be carried out symbolically. The results are a set of mathematical relations characterizing th e 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, 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. The impact that quantitative analysis of software system can have in software design

Key words and phrases. Software Architecture, CHAM, performance evaluation, Queueing Network Model, performance indices.

has been recognized long ago in the software engineering community. Comprehensive approaches to this topic exist, notably the pioneeristic work by Smith [17, 18]. There exists a general agreement 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. Recently, much attention has been devoted to this problem [21] and software architectures have been devised as the appropriate design level for carrying out quantitative analysis [19, 20]. Software architecture represents the first system abstraction in which both the static decomposition of the system in subsystems and the dynamic interactions among subsystems is defined. In the software practice the problem is that software is complex and the evaluation process, in terms of model definition, is very empirical [4], thus it is not easy to apply to software systems the same evaluation approaches used for hardware systems. We believe that, in order to be successful, this kind of analysis has to be automatized as much as possible. The software architecture level of software modeling although simple is very informative, since important design choices have already been taken. Our approach is based on the assumption that, in building the evaluation model, as much work as possible has to be automatically derived by the Software Architecture description. Our choice is to use the CHAM description of a SA, but the approach is not committed to it. 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 we will be able to derive mathematical relations among the parameters—that is, the evaluation indices—characterizing the competing models. 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 relations provide 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 goes in the direction of *compositional* quantitative analysis. The paper is organized as follows: in Section 2, the CHAM formalism and its use to specify software architecture are introduced. The approach is exemplified by using the well-known example of the multiphase compiler, which will be used throughout the paper to illustrate the performance evaluation method. Section 3 , briefly summarizes the characteristics of Queueing Network Models (QNMs) and of their creation process. Section 4, presents the algorithms to derive from the CHAM specification of a Software Architecture a partially defined QNM. The completion of the QNM definition and its evaluation, are carried out in Section 5. Section 6 and Section 7 adds some concluding remarks and discusses future work.

2. Cham For Software Architectures

In this section we briefly introduce the CHAM formalism and its use for the description of Software Architectures as introduced in [6]. We refer to that paper for more details. A Chemical Abstract Machine is specified by defining a set M of *molecules* m_1, $m_2, \dots$, terms of a syntactic algebra that derive from a set of constants and a set of operations and S, a set of *solutions* S_0, S_1, $S_2, \dots$ of molecules.

Molecules constitute the basic elements of a Cham, while solutions are multisets of molecules interpreted as defining the *states* of a Cham. A CHAM specification contains *transformation rules* $T_1, T_2, \ldots$ of the form $m_1, m_2, \ldots, m_k \longrightarrow_{T_i} m'_1, m'_2, \ldots, m'_k$ that define a transformation relations $S_1 \longrightarrow_{T_i} S_2$, dictating the way solutions can evolve (i.e., states can change) in the CHAM. The transformation rules (also called *reaction rules*) can be of two kinds: *general laws* that are valid for all CHAMs and *specific rules* that depend on the particular CHAM being specified. At any given point, a CHAM can apply as many rules as possible to a solution, provided that their premises do not conflict that is, no molecule is involved in more than one rule. In this way it is possible to model parallel behaviors by performing parallel transformations. When more than one rule can apply to the same (set of) molecule the CHAM makes a nondeterministic choice as to which transformation to perform. The CHAM description of a software architecture [6] consists of a syntactic description of the static components of the architecture (the *molecule*), a solution representing the initial state of the architecture, and of a set of reaction rules which describe how the system dinamically evolves through reaction steps. The syntactic description of the components is given by an algebra of molecules. Following Perry and Wolf [14], we distinguish three classes of components: data elements, processing elements, and connecting elements. The initial solution is a subset of all possible molecules that can be constructed using the syntax. It corresponds to the initial, static configuration of the system. Transformation rules applied to the initial solution define how the system dynamically evolves.

The CHAM formalism allows for different analysis techniques which permit to analyze and verify various properties of the system under specification. In particular, it is possible, taking advantage of its operational flavor, to derive transition systems from specifications and then reason at the transition-system level. Here, we will base the derivation of the evaluation model on the transition system but it will also be necessary to make use of the algebraic specification.

2.1. **The three competing SA for a multiphase compiler.** Let us recall the well known multiphase compiler architectures [6, 14]. The data elements are characters, tokens, phrases, correlated phrases (i .e.,phrases signifying name uses related to phrases signifying name declarations), and object code. The processing elements are the text (i.e., the producer of source characters), the lexer (i.e., lexical analyzer), the parser, the semantor (i.e., semantic analyzer), and the code generator. An optional processing element is the optimizer. The 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 1, fits the processing elements together sequentially in the obvious way, with each processing element running its phase to completion before passing data elements on to the next processing element. New test can start the compilation only when the current one finished it. A rather different architecture, see Figure 2 fits the processing elements together via concurrent access to a shared repository. The processing elements run their phases opportunistically and in parallel so that, for example, the **semantor** can correlate phrases at the same time as the **lexer** is creating new tokens.

Finally we define a third architecture for the multiphase compiler; we refer it as intermediate architecture. This last architecture has essentially a sequential structure but a new text can start the compilation even if the current one haven't completed it. The difference respect the sequential architecture stay in the restarting time of the processing elements. The CHAM specification of the sequential

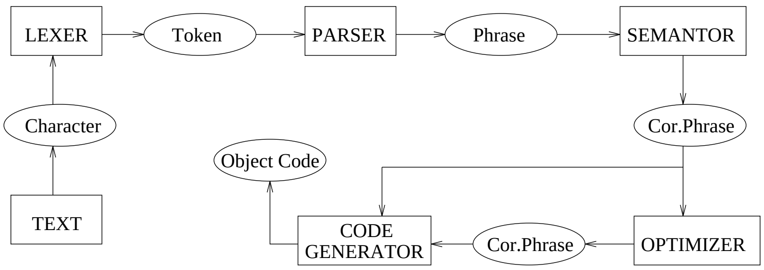


FIGURE 1. Sequential multiphase compiler architecture

architecture consists of the molecule syntax Σ_{seq}, transformation rules T_1, T_2 and T_3 and the initial solution S_1. The syntax Σ_{seq} is defined as follows:

M::= P | C | M $\diamond$M

P ::= **text** | **lexer** | **parser** | **semantor** | **optimizer** | **generator**

D ::= char | tok | phr | cophr | obj

C ::= i(D) | o(D)

where P is the set of processing elements, D is a set of constants representing the data elements and C is the set of connecting elements of the compiler. At the molecules level, we also introduce an infix operator $\diamond$ that we use to express the status of a processing element with respect to its input/output behavior. For example, the molecule i(tok) $\diamond o$(phr) $\diamond$**parser** represents the **parser** in the state in which it can consume tokens as input and then go in a state in which it can produce phrases as output. The molecule **parser** $\diamond i$(tok) $\diamond$ o(phr) represents the **parser** in the state of having consumed all input tokens and produced all output phrases. As we said in the previous section, a state change is achieved by applying suitable reaction rules.

T$_1$=**text**$\diamond$ o(char) $\longrightarrow$ o(char)$\diamond$ **text**

T$_2$ = i(d) $\diamond$ m_1, o(d) $\diamond$ m_2 $\longrightarrow$ m_1 $\diamond$ i(d), m_2 $\diamond$ o(d)

T$_3$ = o(obj) $\diamond$ **generator** $\diamond$ i(cophr) $\longrightarrow$ i(char) $\diamond$ o(tok) $\diamond$ **lexer**,

i(tok) $\diamond$ o(phr) $\diamond$ **parser**, i(phr) $\diamond$ o(cophr) $\diamond$ **semantor**,

i(cophr) $\diamond$ o(cophr) $\diamond$ **optimizer**, i(cophr) $\diamond$ o(obj) $\diamond$ **generator**

where m_1, $m_2 \in M$ and $d \in$ D. The next step is to define an initial solution S_1. This solution is a subset of all possible molecules that can be constructed under Σ_{seq} and corresponds to the initial, static configuration of a system conforming to the architecture. Transformation rules applied to the solution define how the system dynamically evolves from its initial configuration.

S_1 : **Text** $\diamond$ o(char),
i(char) $\diamond$ o(tok) $\diamond$ **Lexer**,
i(tok) $\diamond$ o(phr) $\diamond$ **Parser**,
i(phr) $\diamond$ o(cophr) $\diamond$ **Semantor**,
i(cophr) $\diamond$ o(cophr) $\diamond$ **Optimizer**,
i(cophr) $\diamond$ o(obj) $\diamond$ **Generator**.

The first rule makes source text available for compiling. The second rule describes pairwise input/output communication between processing elements. Completion of the communication, that is, the result of the transformation, is indicated by a rewriting of the molecule such that the data port is moved to the right-most position of the molecule. The third transformation rule restores the processing elements to their initial states after the code generator has completed its task and offered object code as output. This allows the system to begin processing a new source text. Let

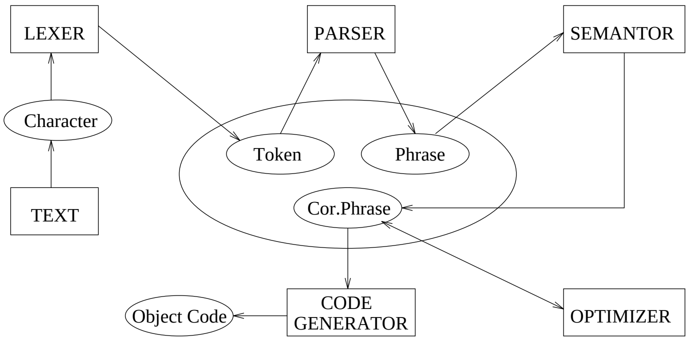


FIGURE 2. Multiphase Compiler Architecture with Repository

us now briefly present the CHAM specification of the second multiphase compiler architecture. The primary difference is that there are new elements among the connecting elements, to represent the use of the repository. In particular, there is a repository element and two new operations, ir and or, whose roles are analogous to the i and o operations already introduced, but apply to the repository rather than to the processing elements. In addition, the structure of the molecules is enriched by introducing an infix operator $\|$ to syntactically represent a composed molecule that can be broken down into parallel subcomponents, thus allowing multiple reactions to occur simultaneously. In more familiar terms, it can be intuitively interpreted as a parallel operator. The syntax Σ_{con} of this architecture is obtained by augmenting Σ_{seq} as follows.

$\mathrm{M} ::= \mathrm{P} \mid \mathrm{C} \mid \mathrm{M} \diamond \mathrm{M} \mid \mathrm{M} \parallel \mathrm{M}$

$\mathrm{P} ::= \textbf{Text} \mid \textbf{Lexer} \mid \textbf{Parser} \mid \textbf{Semantor} \mid \textbf{Optimizer} \mid \textbf{Generator}$

$\mathrm{D} ::= \mathrm{char} \mid \mathrm{tok} \mid \mathrm{phr} \mid \mathrm{cophr} \mid \mathrm{obj}$

$\mathrm{C} ::= i(\mathrm{D}) \mid o(\mathrm{D}) \mid ir(\mathrm{D}) \mid or(\mathrm{D}) \mid \mathrm{rep}_i$

where $i = 1, 2, 3$. Next, we need a solution to represent the initial configuration. Let us call it S_1':

$$
\begin{aligned}
S_1'\!:&\textbf{Text} \diamond o(\mathrm{char}),\\
&i(\mathrm{char}) \diamond o(\mathrm{tok}) \diamond \textbf{Lexer},\\
&i(\mathrm{tok}) \diamond o(\mathrm{phr}) \diamond \textbf{Parser},\\
&i(\mathrm{phr}) \diamond o(\mathrm{cophr}) \diamond \textbf{Semantor},\\
&i(\mathrm{cophr}) \diamond o(\mathrm{cophr}) \diamond \textbf{Optimizer},\\
&i(\mathrm{cophr}) \diamond o(\mathrm{obj}) \diamond \textbf{Generator},\\
&ir(\mathrm{tok}) \diamond \mathrm{rep}_1 \parallel ir(\mathrm{phr}) \diamond \mathrm{rep}_2 \parallel ir(\mathrm{cophr}) \diamond \mathrm{rep}_3.
\end{aligned}
$$

The solution contains a molecule that represents concurrent access to the repository at the granularity of the three kinds of data elements stored in the repository. To complete the specification, a new set of transformation rules is defined.

$$
\begin{aligned}
T_5 : {}& i(\mathrm{char}) \diamond o(\mathrm{tok}) \diamond \textbf{Lexer}, o(\mathrm{char}) \diamond \textbf{Text} \longrightarrow\\
& o(\mathrm{tok}) \diamond \textbf{Lexer} \diamond i(\mathrm{char}), \textbf{Text} \diamond o(\mathrm{char})
\end{aligned}
$$

$T_6 : ir(\mathrm{d}) \diamond \mathrm{rep}_i, o(\mathrm{d}) \diamond m \longrightarrow ir(\mathrm{d}) \diamond \mathrm{rep}_i, m \diamond o(\mathrm{d}), or(\mathrm{d}) \diamond \mathrm{rep}_i$
$T_7 : i(\mathrm{d}) \diamond m, or(\mathrm{d}) \diamond \mathrm{rep}_i \longrightarrow m \diamond i(\mathrm{d}), or(\mathrm{d}) \diamond \mathrm{rep}_i,$
$T_8 : i(\mathrm{d}) \diamond m, or(\mathrm{d}) \diamond \mathrm{rep}_i \longrightarrow m \diamond i(\mathrm{d})$
$T_9 : m_1 \parallel m_2 \parallel \cdots \parallel m_n \longrightarrow m_1, m_2, \ldots, m_n$
$T_{10} : p \diamond i(\mathrm{d}_1) \diamond o(\mathrm{d}_2) \longrightarrow i(\mathrm{d}_1) \diamond o(\mathrm{d}_2) \diamond p$
$T_{11} : m \diamond i(\mathrm{d}) \longrightarrow i(\mathrm{d}) \diamond m$
$T_{12} : \mathbf{Text} \diamond o(\mathrm{char}) \longrightarrow o(\mathrm{char}) \diamond \mathbf{Text}$
$T_{13} : \mathbf{Text} \diamond o(\mathrm{char}) \longrightarrow \mathbf{Text}$
where $m, m_1, m_2, \ldots, m_n \in M$, $d, d_1, d_2 \in D$, $p \in P$ e $i = 1, 2, 3$.
T_5 is a specialization of the generic reaction rule T_2 from the sequential compiler architecture that describes the input of text to the **lexer**. T_6 through T_8 are reaction rules that generically describe the communication of data elements between the repository and the processing elements; flow of data from a processing element into the repository is described by T_6 while T_7 and T_8 describe the reverse. T_9 breaks apart a complex molecule into its (parallel) components, which can then participate in (parallel) reactions; T_{10} reactivates an inert processing element; T_{11} reactivates the input port for a processing element before that processing element performs output; and T_{12} makes more of the source text available for compilation, thereby continuing the whole compilation process for a given source text. Finally , T_{13} is a reaction rule that terminates character input, indicating that the source text has been exhausted. The effect of T_{13} is to eventually cause all processing of a particular source text to stop.

Finally we specify the intermediate architecture. Molecules syntax Σ_{int} and the initial solution are equal to the syntax Σ_{seq} and the solution S_1 of the sequential architecture, respectively; The transaction rules, instead, are as follows:
$T_1 : \mathbf{Text} \diamond o(char) \longrightarrow o(char) \diamond \mathbf{Text}$
$T_2 : i(d) \diamond m_1, o(d) \diamond m_2 \longrightarrow m_1 \diamond i(d), m_2 \diamond o(d)$
$T_3' : p \diamond i(d_1) \diamond o(d_2) \longrightarrow i(d_1) \diamond o(d_2) \diamond p$
$T_4 : o(obj) \diamond \mathbf{Generator} \diamond i(cophr) \longrightarrow i(cophr) \diamond o(obj) \diamond \mathbf{Generator}$
where $m_1, m_2 \in M,\ d, d_1, d_2 \in D\ e\ p \in P$. Rules T_3' and T_4, realize the independent restarting of processing elements, the other rules have the same meaning as in the sequential case.

3. The Queueing Network Model

Queueing Network Models (QNMs) have been extensively applied to represent and analyze the performance of resource sharing systems, such as production, communication and computer systems. A QNM is represented as a *network of queue* which is quantitatively evaluated through analytical or simulation methods to obtain a set of measures, the *performance indices*. A network of queue is a collection of *service centers*, which represent system resources, and a set of *customers*, which represent users sharing the resources. A QNM can be represented as a directed labeled graph where nodes are service centers, arcs represent transitions of customers along nodes and labels on arcs define the routing probability. Different types or classes of customers with various characteristics of customer service and routing behaviour can be modeled by multiclass queueing networks. In single class QNM all the customers have the same behaviour. QNM can be open, closed or mixed. In an open network model there is at least one arc entering and at least one arc going out the network. In an open network each customers arrives from outside, asks for service at a service center, possibly queueing for it, and then it either leaves the system or asks for service to some other service center. In a closed network there is no external arrival and exit arc and There is a constant number of customers circulating in the system. Multiclass networks can be either open, closed or mixed. A mixed network is open with respect to some customer types and closed with respect to others. The queueing capacity of Each service center is formed by a set of servers and a queue to keep waiting customers. The queue capacity can be finite to represent system resources with limited waiting rooms or population constraints. QNMs with finite capacity queues are more appropriate models for systems with finite buffer and blocking, but their analysis is in general more complex than infinite capacity QNMs and it is often carried on with approximated techniques [15]. This class of QNMs is referred to as Queueing networks with Blocking [2, 7, 8, 9, 10, 12, 15]. When a user tries to enter a saturated node, i.e. a node node whose queue has attained its maximum capacity, then there is a *block*. In the literature several proposals exist with respect to different types of blocking mechanisms. We just recall the Blocking After Service (BAS) mechanism that we shall apply in Section 5 to model system synchronization. Under BAS policy when a customer, after completion of service at node i, tries to enter a saturated node j, is blocked and has to wait at node i, so blocking the service. The service remains blocked untill a buffer position becomes available in node j, i.e. as soon as a customer leaves node j. When more than one node are blocked by a saturated node, we have to define an unblocking discipline. We refer to FBFU (First Blocked First Unblocked) discipline, where the first unblocked node is the first one that has been blocked. For simplicity, hereafter we only deal with open models, therefore we only consider features of open models. Informally a QNM can be described by three basic elements:

- **service centers** also called nodes. We specify the *number* of service centers and for each node:
  - *capacity queue*, i.e. the number of users that can wait for service in the service center's queue.
  - *service discipline*, i.e. the scheduling of the customers in the queue;
  - *number of servers*, which represents the maximum number of customers that can be simultaneously served by the node. Each server executes a service and all servers are identical;
  - the *service rate*, i.e. units of services per unit of time;
- **customers** we define:
  - the *arrival process to each node* , i.e. the arrival of the customers at the system; this is usually a stochastic process for open networks, while it reduces to the total customer population for closed models;
  - *customers required services* to each node, expressed in service units; this is usually a random variable;
  - the *classes (or chain or type) of customers* that partition customers in subsets with similar behavioral characteristics;
- **node interconnection structure** (or topology) represents the possible customer transitions among the nodes and their behaviour that can be either stochastic or deterministic. Non-deterministic customer behavior is represented by a stochastic routing matrix whose entries are the routing probabilities between nodes ant that is defined for each class of customers.

The modelling process with a QNM can be split in three steps: (1) *definition*, which normally encompasses the definition of service centers, their number, class of customers and topology; (2) *parameterization*, to define the alternatives of the study, e.g. by selecting the arrival processes and service rates; (3) *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. We assume that the system satisfies the stationary condition and we analyze its steady-state behavior. 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 process. Specifically, for a class of QNMs, one can define a discrete-space continuous-time Markov process that can be analyzed by some solution techniques. Under some special constraints the QNMs analysis can be further greatly simplified. For the class of *product-form* QNMs efficient solution methods have been defined, with a polynomial computational complexity in the number of network components [7, 8, 9]. System performance evaluation is carried out By the analysis of the QNM to evaluate a set of *performance indices*. Such indices can be *local* to a resource or *global* to the whole system. We consider the following local indices for a node i:

- n_i the random variable number of customers at node i;
- $N_i = E[n_i]$ mean of n_i;
- R_i mean response time of a customer at node i ;
- U_i utilization of node i;
- X_i throughput of node i.

and the global indices :

- N mean number of customers in the network;
- R mean network response time of a customer;
- U overall utilization;
- X overall throughput.

where N is a random variable because the network is open.

4. From The CHAM SA To The QNM

In this section we present the algorithm to derive the performance evaluation model out of the CHAM specification of a software architecture. The algorithm actually performs the first step of the modeling process described in the previous section, i.e. the definition of the QNM. The algorithm is based on the analysis of the transition system (TS) that represent the dynamic behavior CHAM architecture and can be automatically derived from the CHAM specification [13]. In fact the construction of the QNM is carried out from the analysis of the interactions among system components and among customers and system that are dynamic information that can be derived only from a behavioral model of the system. TS can be represented by a directed labeled graph whose nodes are the system states, arcs are the transitions between states and labels are the transition rules which allow state transitions. Without lose of generality, we refer to a TS in which the transitions corresponding to non-deterministic behavior are marked in order to distinguish interactions corresponding to a choice in the computation and interactions corresponding to a contemporary communication, i.e. 1 to N or N to 1 communication.

The algorithm is organized into two sequential steps. The first step (PHASE 1) examines all the TS paths, it looks at the states along a path (i.e. solutions) in order to single out all the pairs of element which are involved in an interaction. The second phase (PHASE 2) uses these interaction pairs to device the topology of the evaluation model.

4.1. **PHASE 1.** This phase takes in input the TS and sets derived from the CHAM specification. These are the set of processing elements P, corresponding to system components; the sets RR and EIR of rules that describe, respectively, the explicit interactions between system elements and the implicit interaction between some system components and the environment. If the EIR set is not empty then we do not explicitly modeled the system environment, thus a new element ENV is inserted in P. In this phase the algorithm performs a TS visit and the derived information are listed in sets, called *interaction set*, corresponding to the output of this phase. The algorithm examines for each visited state all its adjacent states and the transition connecting them. If the transition corresponds to a communication then the algorithm generates a pair, called *Interaction Pair*, that signals a flow of data elements from the first to the second elements of the pair. The first element is the component that acts as sender in the communication, while the second element is the component that acts as receiver. If the communication is synchronous then both elements of the pair are system components, if the communication is asynchronous, instead, it is modeled by means of connecting elements, called *Passive Connecting Elements* (PaC), having buffering function, so in the corresponding interaction pair there are this kind of elements. Finally notice that the algorithm marks interaction pairs corresponding to a non-deterministic behavior. In this way it can correctly model concurrency and non-determinism, in the second phase. All the pairs derived at the $i-th$ visit step are collected into a unique Interaction Set. Then the output set I_i contains all the pairs obtained by checking a TS node and its adjacent nodes, therefore it represents all the communications among system elements that can happen at a given time.

4.2. **PHASE 2.** In the second phase the I_i sets computed above are examined in order to associate elements of the QNM. Additional information are necessary to determine the QN topology. In particular we need to identify internal and external components of the system. From the set P we extract a subset of those elements representing the environment (set A). Elements in A can be seen as sources which model the production of system customers or processing elements from which the system communicates the results to the environment. From set A we determine if the resulting QNM is open or closed. Let us informally introduce how the second phase algorithm works.

The idea is to analyze the interaction among system components (i.e. I_i sets) and to understand theirs *real* level of concurrence and consequently of the QNM. This is very important in order to obtain QNMs that faithfully represent the given system. The goal is to understand which components are strongly synchronized so to result in a sequential behavior, and which are independent from others and can be concurrently active. At the beginning each component is considered as an autonomous server, thus modeling the maximum level of concurrency then, along the computation it can become part of a more structured element. Intuitively, if an *interaction pair* (p_1, p_2) corresponds to a synchronous communication, then the algorithm associates to it a complex server representing a unique service composed of p_1 following p_2 and that express the sequentiality of operations. If an *interaction* *pair* (p_1, p_2) corresponds to a asynchronous communication, the receiving data component is modeled by a service center with a infinite buffer that implicitly model the communication channel. Moreover from this interaction pair we can build a corresponding transition in the topology of the QNM. The interaction pairs having an external element define the service centers in which there are exegenous arrival (the external element is the first element of the pair) and the service centers from which there are departure (the external element is the second element of the pair). 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 imposes a BAS blocking mechanism to the sender component in order to model synchronization. The algorithm models a non-deterministic computation by introducing a multi-customers service center that at the end is transformed in a simple-customer service center whose service time depends on the service times of the original classes.

One to N and N to one communications are modeled assigning to the involved components distinct service centers, with a zero capacity buffer if the communication is synchronous. When all the sets I_i have been examined the algorithm proceeds by performing *merging* operations to reach the final configuration of the service centers. There are several merging operations as listed in the last part of the algorithm. The result of these merging operations is a set of interconnected service centers. The interesting reader can find in the Appendix a complete description of the two phases algorithm with its complexity analysis, as well as the definition and graphical notation of the QNM components elements, with explicitly refer to architectures.

From the algorithm we do not obtain a completely specified QNM. This is expected since we have only performed a functional analysis of the system. We still need to establish 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 distributions characterizing the service times for every center;
- the distributions characterizing the customers arrival processes 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 algorithms and his knowledge of the QNMs. It is in fact important to select the quantitative parameters so that the resulting QNM is in a class that admits an efficient solution method. To this aim for QNMs with blocking we usually derive single class models.

5. Deriving the QNMs for the three multiphase compiler architecture

In this section we derive the three QNMs associated with the sequential, concurrent and intermediate architectures of the multiphase compiler. In the normal practice, evaluation models are derived in an empirical way, by analyzing the system. We show how we can use the CHAM specification and the algorithm defined in the previous section to derive the topology of the model and how to determine the quantitative parameters in order to carry on an efficient numerical evaluation of the three models.

5.1. **The model for the sequential architecture.** From the CHAM specification of the sequential architecture, starting from S_1 by applying T_1, T_2 and T_3, the TS in Figure 3 can be derived where S_1 is defined in section 2.1 and where others

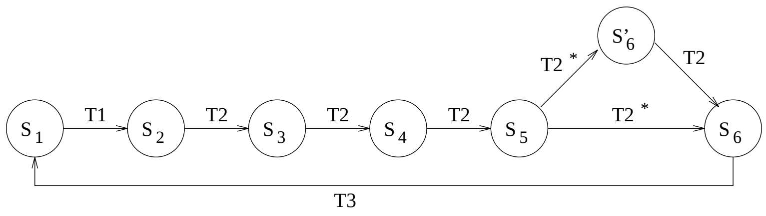


FIGURE 3. TS Sequential Multiphase Compiler

are defined as follow:

$S_2 = o(\text{char}) \diamond \textbf{text},$
$i(\text{char}) \diamond o(\text{tok}) \diamond \textbf{lexer},$
$i(\text{tok}) \diamond o(\text{phr}) \diamond \textbf{parser},$
$i(\text{phr}) \diamond o(\text{cophr}) \diamond \textbf{semantor},$
$i(\text{cophr}) \diamond o(\text{cophr}) \diamond \textbf{optimizer},$
$i(\text{cophr}) \diamond o(\text{obj}) \diamond \textbf{generator}$

$S_3 = \textbf{text} \diamond o(\text{char}),$
$o(\text{tok}) \diamond \textbf{lexer} \diamond i(\text{char}),$
$i(\text{tok}) \diamond o(\text{phr}) \diamond \textbf{parser},$
$i(\text{phr}) \diamond o(\text{cophr}) \diamond \textbf{semantor},$
$i(\text{cophr}) \diamond o(\text{cophr}) \diamond \textbf{optimizer},$
$i(\text{cophr}) \diamond o(\text{obj}) \diamond \textbf{generator}$

$S_4 = \textbf{text} \diamond o(\text{char}),$
$\textbf{lexer} \diamond i(\text{char}) \diamond o(\text{tok}),$
$o(\text{phr}) \diamond \textbf{parser} \diamond i(\text{tok}),$
$i(\text{phr}) \diamond o(\text{cophr}) \diamond \textbf{semantor},$
$i(\text{cophr}) \diamond o(\text{cophr}) \diamond \textbf{optimizer},$
$i(\text{cophr}) \diamond o(\text{obj}) \diamond \textbf{generator}$

$S_5 = \textbf{text} \diamond o(\text{char}),$
$\textbf{lexer} \diamond i(\text{char}) \diamond o(\text{tok}),$
$\textbf{parser} \diamond i(\text{tok}) \diamond o(\text{phr}),$
$o(\text{cophr}) \diamond \textbf{semantor} \diamond i(\text{phr}),$
$i(\text{cophr}) \diamond o(\text{cophr}) \diamond \textbf{optimizer},$
$i(\text{cophr}) \diamond o(\text{obj}) \diamond \textbf{generator}$

$S_6 = \textbf{text} \diamond o(\text{char}),$
$\textbf{lexer} \diamond i(\text{char}) \diamond o(\text{tok}),$
$\textbf{parser} \diamond i(\text{tok}) \diamond o(\text{phr}),$
$\textbf{semantor} \diamond i(\text{phr}) \diamond o(\text{cophr}),$
$i(\text{cophr}) \diamond o(\text{cophr}) \diamond \textbf{optimizer},$
$o(\text{obj}) \diamond \textbf{generator} \diamond i(\text{cophr})$

$S'_6 = \textbf{text} \diamond o(\text{char}),$
$\textbf{lexer} \diamond i(\text{char}) \diamond o(\text{tok}),$
$\textbf{parser} \diamond i(\text{tok}) \diamond o(\text{phr}),$
$\textbf{semantor} \diamond i(\text{phr}) \diamond o(\text{cophr}),$
$o(\text{cophr}) \diamond \textbf{optimizer} \diamond i(\text{cophr}),$
$i(\text{cophr}) \diamond o(\text{obj}) \diamond \textbf{generator}.$

We can now apply the algorithm presented in Section 4, 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**.

For PHASE 1, we have:

INPUT: P={ T, L, P, S, O, G }; RR = { T_2 }; EIR = { T_3 }; $PaC = \emptyset$;

ST = TS of Figure 3; The execution of PHASE 1 goes as follows: First we add the ENV element to P because EIR is not empty, i.e. $P = P \cup \{ENV\}$. After this we can visit the transaction system.

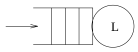
I_1	I_2	I_3	I_4	I_5	I_6
(T,L)	(L,P)	(P,S)	(S,O) (S,G)	(O,G)	(G,ENV)

TABLE 1. Output of the algorithm Phase 1

s_c	s_a	IP	I_i
S_1	S_2	none	
S_2	S_3	(T,L)	I_1 = {(T,L)}
S_3	S_4	(L,P)	I_2 ={(L,P)}
S_4	S_5	(P,S)	I_3 ={(P,S)}
S_5	S_6	(S,G)	I_4 ={(S,G)}
S_5	S'_6	(S,O)	$I_4 = I_4 \cup$ {(S,O)}
S'_6	S_6	(O,G)	I_5 ={(O,G)}
S_6	S_1	(G,ENV)	I_6 ={(G,ENV)}

The OUTPUT of PHASE 1 is the set of sets I_i in Table 1.
For PHASE 2 we have:
INPUT: P' = {L,P,S,O,G}; A = {T,ENV}; $PaC = \emptyset$;
$I = \{I_1, I_2, \ldots, I_5, I_6\}$.
where the set P' differs from the set P in PHASE 1 for the A elements. A contains the processing elements representing the exogenous customers and the environment, in this case Texts and ENV. PaC is empty as in PHASE 1 and I contains the sets I_i shown in table 1.
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]);

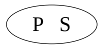


[T L]

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

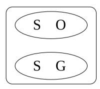
[L P]

- 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]);

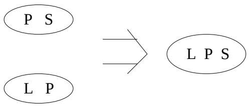
[O G]

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

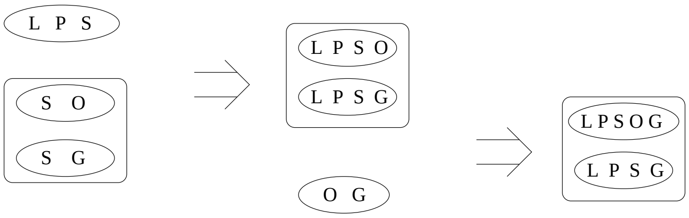
$$\mathrm{G} \longrightarrow ENV$$

Now we can start merging operations in order to define the model topology.

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

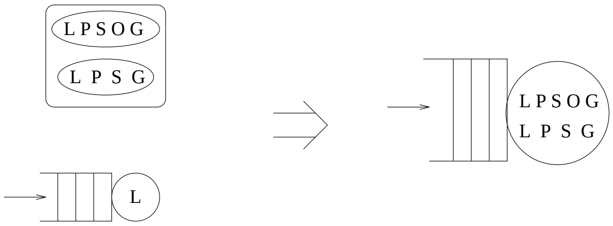


- merge all the compatible complex and multiclass servers; from the complex servers SC(L, P, S) and SC(O, G) and from the multiclass server SM((S, O)(S, G)) the multiclass server SM((L, P, S, O, G)(L, P, S, G)) is generated.



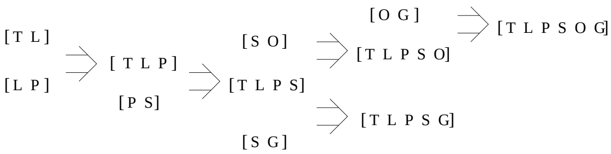
The merging operations "merge all complex servers with service centers", "merge all multiclass servers with service centers" and "merge all complex servers with exogenous arrivals service centers" have no effect since no more service centers or complex servers are left;
Then we perform the following 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 queueing network model topology, we perform the merging operations among paths. These allows the characterization of the service type.

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



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 phase 1 algorithm for intermediate architecture

After the analysis of the I_i sets and merging operations we have to eliminate all multiclass servers in order to obtain only single class models. This is made because when we obtain a blocking network the analysis becomes more complex for multiclass models.

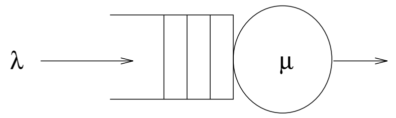


FIGURE 4. QNM for the sequential multiphase compiler architecture

The resulting model is shown in figure (4). Its topology is a unique service center with exogenous arrivals that can offer a unique service having a mean service time corresponding to a weighting average of two service times. One is the service time corresponding to the service consisting in the five compilation phases, optimization included; the other is the service time corresponding to the service consisting in the four phases of compilation, excluding the optimization. The weights in the average allow to point out the more required offered services. The two tuples [T, L, P, S, O, G] and [T, L, P, S, G] characterize such two services and they state that a customer requires services to the processing elements in the order indicates in the tupla. The first element of both tuples is an element in A reflecting that the network is open. We assume a $FCFS$ discipline.

5.2. **The model for the intermediate architecture.** Now let us consider the intermediate architecture of the multiphase compiler, in order to see what happens if there are concurrent and synchronized components. In figure 5 we show the partial TS derived from the CHAM specification of the architecture.
We can now apply the algorithm presented in Section 4, to carry on the first step of the modeling process.
For PHASE 1, we have:
INPUT: P={ T, L, P, S, O, G }; $RR = \{ T_2 \}$; $EIR = \{ T_4 \}$; $PaC = \emptyset$;
ST = TS of Figure 5; The execution of PHASE 1 goes as follows: First we add the ENV element to P because EIR is not empty, i.e. $P = P \cup \{ENV\}$. After this we can visit the transaction system as in the sequential case. Table 2 shows the output of the first phase from which we can apply the second phase.
For the second phase we have:
INPUT: $P' = \{$L, P, S, O, G$\}$; A ={T,ENV}; $PaC = \emptyset$;

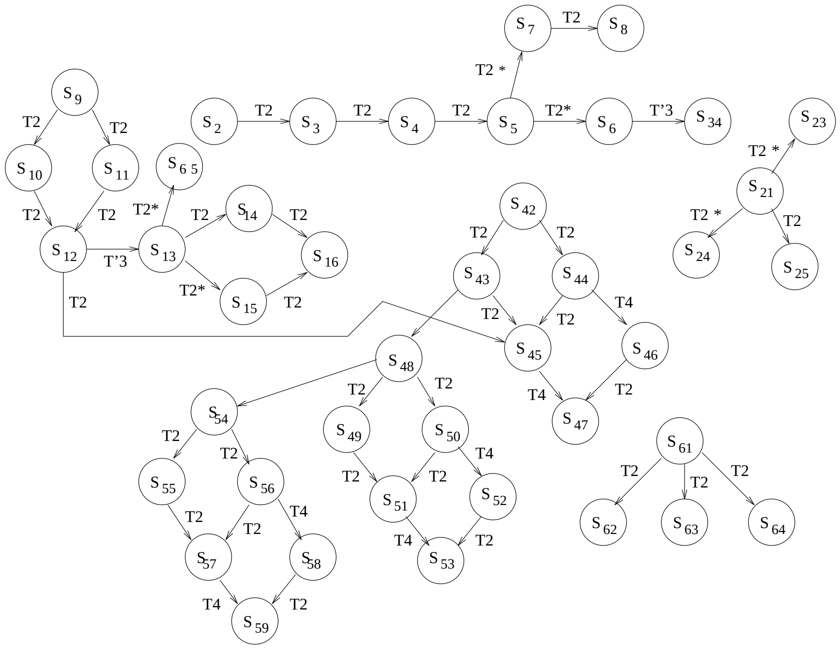
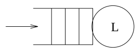


FIGURE 5. TS of Intermediate Multiphase Compiler

$I = \{I_1, \ldots, I_{16}\}$.
where the set P' differs from the set P in PHASE 1 for the A elements. A contains the processing elements representing the exogenous customers and the environment, i.e. Texts and ENV. PaC is empty as in PHASE 1 and I contains the sets I_i shown in table 2.
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]);

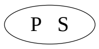


[T L]

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

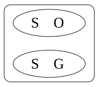
[L P]

- 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]

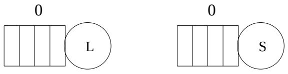
- from I_5 the complex server SC(O,G) and the partial path CP([O, G]);

[O G]

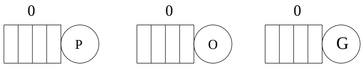
- from I_6 the passing of resulting data from G to the environment.

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

- 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 unitary capacity $CdS_1(P)$, $CdS_1(G)$ and $CdS_1(O)$, while the set $CONC$ become $CONC$ = $CONC \cup$ { {P,G},{L,G} {L,S},{P,S},{P,O},{L,O} }.



Others interaction sets build copy of the elements above, while the set $CONC$ has the following structure: $CONC$ = {{L,P},{L,S},{L,O},{L,G},{P,S},{P,O},{P,G}, {S,O},{S,G}}.

Now analyze each CdS_1 in order to single out those actually need.

- Delete CdS_1(L), there aren't any concurrent processing elements that communicate with i, i.e. no pairs in $CONC$ hold the processing element L as second pair's element.
- CdS_1(P), delete SC(L,P) and build L $\longrightarrow$ P in fact {L,P} $\in CONC$.
- CdS_1(S), delete SC(P,S)and build P $\longrightarrow$ S in fact {P,S} $\in CONC$.
- CdS_1(O), delete SM((S,O),(S,G)) and build S $\longrightarrow$ O and SC(S,G) in fact {S,O} $\in CONC$
- CdS_1(G), delete SC(S,G) and SC(O,G) and build S $\longrightarrow$ G and O $\longrightarrow$ G in fact {S,G},{O,G} $\in CONC$.

The resulting model is shows in figure 6, its topology is five service centers, the first one has exegenous arrivals and infinite capacity buffer, while the others have zero capacity buffer.

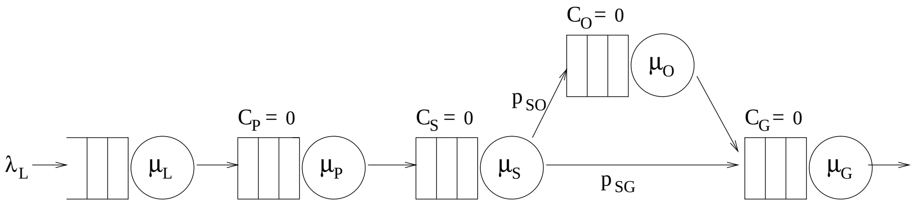


FIGURE 6. QNM for intermediate architecture

5.3. **The model for the concurrent architecture.** The model for the multiphase concurrent compiler architecture is obtained similarly to the previous case, and we only show the resulting model. We assume a FIFO discipline at each service center.

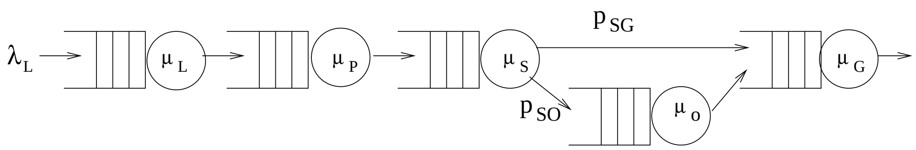


FIGURE 7. QNM for the multiphase compiler architecture with repository

The QNM is formed by five service centers and its topology is shown in Figure 7. From the algorithm we obtain the two tuples [T, L, P, S, O, G] and [T, L, P, S, G] that characterize two type of services as in the sequential architecture. The resulting model is an open single class network.

6. PARAMETERIZATION

In order to characterize the workload we have to define the parameters for the arrival process of the system and for the mean service time for both supplied service. The mean service time for both supplied service is used to define the mean service time of the model according to the weighting average defined in the algorithm. Since we are at a very high level of abstraction we consider symbolic parameters. Such parameters will assume a relative meaning if we compare various competing software architecture and we analyze the models in symbolic way, or they are instantiated. Thus for the model corresponding to the sequential architecture 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 whole model, where a 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 $j \in \{L, P, S, G\}$ of a customer.
- p_{SO} and p_{SG} the routing probabilities that a customer goes from service center S to service center O and from service center S to service center G, respectively.

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

7. EVALUATION

7.1. **Symbolic Evaluation.** Consider the models where each node has infinite capacity queue. In this case we can obtain a product-form QNM, by assuming exponential arrival and service time distribution, FCFS service discipline and independent arrivals and services. Hence the QNM has product-form and we can

analyze each node separately. We assume that the stationary condition holds and we derive the steady-state solution. Let us compare the sequential and concurrent architecture of the multiphase compiler. We start analyzing the QNM for the sequential architecture shown in Figure 4. The network is stable if the unique node is stable, i.e. if the arrival rate is smaller than the service rate, i.e. $\lambda < \mu$. Therefore we impose this condition and proceed with the analysis.
Let us introduce the following notation:

- n, the random variable number of customers in the model;
- $\rho = \frac{\lambda}{\mu}$, traffic intensity of the model; it is 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 as an M/M/1 queueing system whose simple closed-form solution is well-known in the literature [7, 8] and for which we can immediately obtain the performance indices which characterize the model under study. By analyzing the model shown in Figure 4 we derive the following performance indices:

- mean number of customers in the system:
$$N = \frac{\rho}{1-\rho}$$
- system throughput:
$$X = \lambda$$
- mean response time of the system:
$$R = \frac{N}{\lambda} = \frac{1}{\mu * (1-\rho)}$$
- system utilization:
$$U = \text{Prob}\{n > 0\} = \rho$$

It is worth noticing that the assumptions made to define the model are satisfied by the software system, i.e. the model is representative. Moreover the model solution is exact. For the concurrent architecture we obtain the model shown in Figure 7 which is a Jackson QNM with product-form solution and can be easily analyzed by decomposition into five M/M/1 systems with appropriate parameters. The network model is stable if all the nodes are stable, i.e. if the following constraint holds:

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

If this stationarity condition is satisfied then we analyze the network model by the M/M/1 solution for each node, similarly to the analysis of the model of the sequential architecture. We summarize the results as follows.
Let us define:

- n_j, the number of customers at node $j \in \{\mathrm{L}, \mathrm{P}, \mathrm{S}, \mathrm{O}, \mathrm{G}\}$;
- $\rho_j = \frac{\lambda_L}{\mu_j}$, traffic intensity at service center $j \in \{$ L, P, S, G$\}$; it is defined as the ratio between the mean arrival rate and the mean service rate at the service center j.
- $\rho_O = p_{SO}\frac{\lambda_L}{\mu_O}$, traffic intensity at service center O;

Hence we obtain the following local performance indices:

Performance Index	**for node** $j \in \{\mathrm{L,P,S,O,G}\}$
Mean number of customers	$N_j = E[n_j] = \frac{\rho_j}{1-\rho_j}$
Throughput	$X_j = \lambda_L$ and $X_O = p_{SO}\lambda_L$
Mean response time	$R_j = \frac{N_j}{\lambda_L} = \frac{1}{\mu_j(1-\rho_j)}$
Utilization	$U_j = \mathrm{Prob}\{n_j > 0\} = \rho_j$

and the overall system performance indices as follows:

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

7.1.1. *Comparing the two Software Architectures.* In this section we carry on the performance comparison between the two software architectures in terms of the performance index introduced above. To this aim we have to consider the same workload for the two QNMs, i.e. the inputs to these models have to be equally specified. More precisely, these amounts at imposing identities between the load measures, arrival rates of customers and total service demands of the two models. 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;
- $\frac{1}{\mu^c_j}$ mean service time of a customer in center $j \in \{L,P,S,O,G\}$ in the concurrent model;

The relation between the corresponding arrival rates of customers is obtained by imposing $\lambda^c = \lambda^s = \lambda$. 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. This is given by $\sum_{j\in\{L,P,S,G\}} \frac{1}{\mu_j}$, for a customer that requires service type N, and $\sum_{j\in\{L,P,S,O,G\}} \frac{1}{\mu_j}$, for service type O. Therefore the relation between the service demands of the two models is the following:

$$(1) \qquad \frac{1}{\mu^s} = a(\frac{1}{\mu^c_L} + \frac{1}{\mu^c_P} + \frac{1}{\mu^c_S} + \frac{1}{\mu^c_O} + \frac{1}{\mu^c_G}) + (1-a)(\frac{1}{\mu^c_L} + \frac{1}{\mu^c_P} + \frac{1}{\mu^c_S} + \frac{1}{\mu^c_G}).$$

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 perform comparable elaborations.

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 the situation in which 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.

Now we can proceed by comparing 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 workload assumptions above, it is meaningless to simply compare the resulting measures. Indeed, this index directly derives from the mean arrival rates at the models that are assumed identical in the two models. Therefore the throughput for the two models is the same. However, it is meaningful to compare the two models by considering the maximum throughput that they can support, provided that the stationarity 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 $\lambda^c < min(\{\mu_j \mid j \in \{\mathrm{L,P,S,G}\}\}, \frac{\mu_O}{p_{SO}})$ in the concurrent model. Thus we obtain the maximum throughput as follows:

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

in the sequential model and

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

in the concurrent model. Hence we immediately obtain by equation (1) that $X_{seq} < X_{con}$, that is the maximum throughput of the sequential model is less than the throughput in the concurrent model. The performance comparison of the two software architectures in terms of utilization is meaningful by considering the service utilization of the concurrent model and the single service center of the sequential model. In order to compare the two models in terms of mean number of jobs, note that this is a global performance index in the sequential model, while we can derive both the local and the global index in the concurrent model. Hence a meaningful comparison of the two software architectures in terms of this performance index is based on the global mean number of jobs.

Finally, the average response time of the two models can be compared by applying the Little's theorem. Since the two models have the same throughput, as discussed above, their comparison in terms of mean response time provides the same results as for the average number of jobs. Let now consider us 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 equation (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:

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

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 δ_{jO} is equal to p_{SO} if $j = O$ and to 1 if $j \neq O$.
On the other hand for the sequential model we can write: $U_{seq} = \frac{\lambda}{\mu^s} = \frac{(a+4)\lambda}{\nu}$.
By comparing utilization U_{seq} with any U^j_{con} for $j \in \{L, P, S, O, G\}$, we can write:

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

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

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

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

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

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

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

which is a non negative function, due to the stability condition in the sequential model. To understand the meaning of these formulas let us suppose that service types N and O are equally required, that is $a \simeq 0.5$. Under this hypothesis we can say that the maximum throughput in the concurrent model is 4.5-times bigger than that of the sequential one. Consistently, the utilization of a center in the concurrent model is about $\frac{2}{9}$ the utilization of the unique center in the sequential model. Hence the workload that lead the sequential model to its saturation is 4.5 times smaller than the workload that make the concurrent model saturated. Similarly, if 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. Viceversa 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 studying the *Rapp* function we evaluate the ratio of the mean number of customers in the two models, by varying parameters λ and ν. In particular by setting this function to be equal to 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:

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

$$\nu, \lambda > 0 \tag{5}$$

$$\nu > \lambda \tag{6}$$

We observe that $k \geq 0$ because of the system stability conditions, 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. Equation (4) provides the relationship between the system parameters arrival rate λ and service rate ν in order to observe the average number of jobs in the sequential model k-times greater than that of the concurrent model. By summarizing for this scenario we can quantitatively compare the two software architecture by observing that:

- the maximum throughput in the concurrent model is 5-times greater than that of the sequential model, if service O dominates (4-times if service N dominates and 4.5-time if two services are equally required),
- the saturation workload for the sequential model is five times smaller than that of the concurrent model (four time smaller if service N dominates and 4.5 time smaller if two services are equally required),
- relationship (4) 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:

$$\frac{1}{\mu^s} \simeq \frac{1}{\mu^c_L} \tag{7}$$

$$\frac{1}{\mu^c_j} \ll \frac{1}{\mu^c_L} \quad j \in \{\mathrm{L, P, S, O, G}\} \tag{8}$$

Hence we obtain $X_{con} = \mu^c_L$ and $X_{seq} = \mu^s$. By formula (7) we 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 (7) we obtain identity. Hence we conclude that the two models for each class have the same critical workload. We observe that also the mean number of customers in a given center, L, is greater than that in the other centers of the model. By condition (8) 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 (7) yield nearly the same value. Hence the two models have nearly 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 nearly 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:

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

where

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

By using formula (3) this leads to $X_{con} = l$.
Substituting relation (1) in formula (9) we have:

(10) $$\frac{a+4}{u} \leq \frac{1}{\mu^s} \leq \frac{a+4}{l}$$

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 constrains:

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

For the sequential model utilization is derived by formula (10) and by U_{seq} 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 constraint:

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

On the other hand for the mean number of customers in the sequential model, from formula (10) 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} we obtain

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

From the formulas above 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 $(a+4) \in [4,5]$.

The latter relationship provide an interval for the ratio of the mean number of jobs in the two models as a function of system parameters (arrival and service rates). This ratio denotes 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 two models in terms of mean queue length and mean response time is expressed by formula (11) that provides an interval for the ratio of the mean number of jobs in the two models as a function of system parameters (arrival and service rates). This ratio interval can be used to evaluate the relative performance improvement of the concurrent model with respect to these two performance indices. In the development process of the software project this relation can be applied to determine the appropriate workload and its allocation (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.

7.2. **Evaluation of network with blocking and synchronization.** When we represent the software architecture synchronization by a QNM with finite capacity queues and blocking, the evaluation of performance measures is more expensive and complex than product-form networks. Indeed we cannot apply the efficient algorithms for separable network with infinite capacity queues, but we usually have to make an approximate analysis of models by an heuristic method. Therefore, due to the major complexity of the evaluation process we cannot perform symbolic analysis because the approximation methods have an iterative structure to compute some parameters. In the case that under study we must instantiate model parameters for evaluating and comparing the derived models. Evaluation can be carried on with simulative methods or with tools for the approximate analysis of queueing network. To this aim we can derive a range of values for each model parameter and implementation scenario. For each scenario we evaluate a set of performance indices. Such values can be interpolated to obtain a function that describe the index trend within the defined ranges. For example we can assume a scenario in which all the architecture components have comparable mean service time. Then the the mean service time of each service centers of the network is of the same degree of magnitude. Another example is a scenario in which there is a bottleneck component, then either the mean service time of the corresponding service center should be greater than that of other service centers or the workload to that center has to higher than that to the other ones. Moreover if the model has joint service centers, i.e. service centers with a mean service time equal to a weighting average of some mean service times, then we can define two cases: (i) the weights are equally distributed, (ii) there is a weight more relevant with respect to the other ones. Software performance evaluation has to be carried on for each case, as described above. This corresponds to consider the architecture behavior when there is a uniform requirement of services (case (i)), and when a given service is more required (case (ii)). Although this numerical analysis can be expensive when we have to consider a wide set of model parameters in the performance analysis, however note that the direct symbolic evaluation can applied only in particular cases.

8. Conclusions

In this paper we have presented an approach to evaluate the performance of software architecture specified in CHAM. The method allows the automatic derivation of a QNM that can then be efficiently evaluated. In carrying on this work we have made two main choices, let us briefly discuss them in the following. 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. Future work concerns the construction of a comprehensive system in which both the specification and the evaluation of a software architecture can be carried out. So far, we have actually used separate tools to perform the various steps. From our (limited) experience we have found this approach encouraging and stimulating both in terms of design and analysis support.

Why Queueing Network Models ? Queueing Network Models have been a popular tool in the performance evaluation world since they achieve a favorable balance between accuracy and efficiency. From our perspective of software architectures, QNM can be defined, parameterized and evaluated at a low cost and their level of abstraction easily allows a faithful modeling. Other approaches to performance evaluation of software system specification have been proposed, which include stochastic process algebra and stochastic Petri nets, e.g. [1, 3]. Most of the approaches are based on Markovian processes, which although very general have the drawback of not guaranteeing efficient solutions.

Why CHAM specification? The approach we have presented is not committed to the use of the CHAM formalism. The derivation of the QNM starts from a labeled transition system which represents the dynamic behaviour of the system. Actually, the algorithms presented in Section 4 needs also some auxiliary information (the INPUT sets) which are directly derived from the algebraic specification. In principle any formalism that has a transition system semantics suitably enriched with some auxiliary information on the nature of the interconnection mechanisms can be used.

The work presented in this paper represents a first step towards the construction of a flexible environment for the evaluation of software architectures specified in CHAM. At present we can automatically generate the transition system out of a CHAM description. Future work is the realization of a user-friendly environment to drive the completion of the QNM definition, with the possibility of creating several different scenarios in which evaluate the models.

References

1. G. Balbo, G. Conte and M. A. Marsan. Performance Models of Multiprocessor Systems. Series in Computer Systems, The MIT Press, (1986).
2. 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.
3. M. Bernardo and R. Gorrieri "Extend Markovian Process Algebra" In Proc. CONCUR '96, LNCS (Springer-Verlag) No. 1119, (1996) 315-330.
4. G. Bckle, H. Hellwagner, R. Lepold, G. Sandweg, B. Schallenberger, R. Thudt, S. Wallstab "Structured Evaluation of Computer System" IEEE Computer Society, 45-51, June 1996.
5. D. Compare, P. Inverardi and A.L. Wolf "Uncovering Architectural Mismatch in Dynamic Behavior" Technical Report CU-CS-828-97, Department of Computer Science, University of Colorado, Boulder, Colorado, February 1997, to appear in Science of Computer Programming, January 1999.

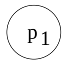
6. P. Inverardi and A. L. Wolf "Formal Specification and Analysis of Software Architectures Using the Chemical Abstract Machine Model" IEEE Transactions on Software Engineering, Vol. 21, No. 4, 373-386, April 1995.
7. K. Kant. Introduction to Computer System Performance Evaluation. McGraw-Hill, 1992.
8. L. Kleinrock. Queueing Systems. Vol. 1: Theory. (Wiley 1975).
9. 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).
10. E. D. Lazowska, J. Zahorjan and M. Vernon "A Comparison of Performance Petri Nets and Queueing Network Models" Modelling Techniques and Performance Evaluation, edit by S. Fdida and G. Pujolle, Elsevier Science Publishers B. V. (North-Holland), 191-202, (1987).
11. F. Aquilani "Valutazione delle Prestazioni di Architetture Software" Thesis in Computer Science, University of L'Aquila, December 1999.
12. R. O. Onvural "Survey of Closed Queueing Networks with Blocking" ACM Computing Surveys, Vol. 22, No. 2, 83-121, June 1990.
13. G. Pasquini "Uno strumento per la generazione di automi da specifiche CHAM" Thesis in Computer Science, University of L'Aquila, December 1999.
14. 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.
15. H. G. Perros, "Queueing Network with Blocking: Exact and Approximate Solutions" Oxford University Press, 1994.
16. M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prent. Hall, (1996).
17. C. U. Smith. Performance Engineering of Software Systems. Addison-Wesley Publishing Company, (1990).
18. 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.
19. L. Bass, P. Clements and R. Kazman. "Analyzing Development Qualities at the Architectural Level". Software Architecture in Practice. SEI Series in Software Engineering, Addison-Wesley 1998.
20. C. Hofmeister, R. Nord and D.Soni, "Applied Software Architecture" Addison-Wesley, Object Technology Series, October 1999.
21. IEEE Proceedings of the First International Workshop on Software and Performance, Santa Fe', October 1998.

Appendix

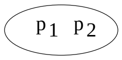
In order to identify the model components that will be incrementally built by the algorithm, we have introduced a set of definitions with their graphical representations.

DEFINITIONS

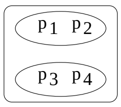
- Simple Server $SS(p_1)$, it provides a service performed by the processing element p_1.



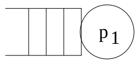
- Complex Server $SC(p_1, \ldots, p_n)$, it provides, as unique service, the set of service performed by the simple severs $SS(p_1), \ldots SS(p_n)$, in the given order. Its average service time results from the summation of the average service times of the simple servers.



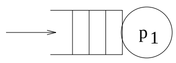
- Multiclass Server, $SM((p_1, p_2), (p_1, p_3))$, it provide different services (in this case two); a generic user can choose one of the services associated to the two complex servers. The pictured SM can be generalized with more two SC, each one with more two elements.



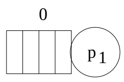
- Joint Simple Server, $SS_{med}(p)$. It enables to eliminate the multicalss servers, in order to obtain single class models.
- Service Center, $CdS(p_1)$, it provide the service offered by the processing element p_1; differently from the simple server it has an infinite buffer to queue users waiting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



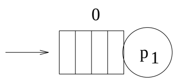
- Service Center with external arrival, $CdS_e(p_1)$, it is like $CdS(p_1)$, but it allows users from outside the system.



- Service Center with unit capacity, $CdS_1(p)$. It models the sinchronizzation; it has a zero capacity buffer.



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



- Compulsory Service Center with unit capacity, $CdS_{e1}(p)$. It models the broadcast communication.
- Connecting Service Center, $CdS_c(p)$. It indicate a connecting element; it has a infinite buffer and a neglect 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, while b can be a service center or a service center with exogenous arrivals.
- Temporary Right Transition, $p_1 \longrightarrow_{trd} p_2$. It indicates a transition of data from the processing element p_1 to the connecting element p_2 (right element of transition).
- Temporary Left Transition, $p_1 \longrightarrow_{trs} p_2$. It indicates a transition of data from the connecting element p_1 to the processing element p_2 (left element of transition).
- Partial Path, $CP(p_1, \ldots, p_n)$ it indicates the processing elements that can be traversed by a user, it indicates a possible history of the requested services to the system by a generic user.

$$[p_1, \ldots, p_n]$$

In the sequel we report the complete version of the algorithm and its complexity.

FIRST PHASE

INPUT:
  $P = \{$ Processing Elements $\}$
  $RR = \{$ Reaction Rules $\}$
  $EIR = \{$ Extern Interaction Rules $\}$
  $PaC = \{$ Passive Connecting Elements $\}$
  $TS =$ Transaction System

OUTPUT:
  $I = \{I_1, I_2, \ldots\}$ where $I_i = \{$ i-th Interaction Pairs Set $\}$

Begin
    If $EIR \neq \emptyset$ **Then**
      $P = P \bigcup \{ENV\}$;
    $S = \{S_1\}$; $i = 1$; $\sqsubset$ where S is defined as a list $\sqsupset$
    While $S \neq \emptyset$ **Do**
      s_c =Extract_First_Element (S); $\sqsubset$ Mark extract element $\sqsupset$
      If $\mid I_i \mid > 0$ **Then**
        $i = i + 1$;
      $AS =$ Adjacent_States (s_c);
      While $AS \neq \emptyset$ **Do**
        $s\_a =$ Extract_First_Element (AS);
        Insert(s_a,S); $\sqsubset$ If s_a not marked $\sqsupset$
        T =Label(s_c,s_a);
        If $T \in RR$ **Then**
          $<$ Extract the molecule pair involving in the reaction $>$;
          $<$ Extract processing or connecting elements P_1, P_2 $>$;
          $<$ Build the pair $IP = (P_1, P_2)$ or $IP = (P_2, P_1)$ $>$;
          $I_i = I_i \bigcup IP$;
          If Marked(T) **Then**
            $<$Mark the pair IP $>$;
        If $T \in EIR$ **Then**
          $<$ Extract the molecule at which the rule is applied $>$;
          $<$ Extract from the molecule the processing element P_1 $>$;
          $<$ Build the interaction pair $IP = (P_1, ENV)$ $>$;
          $I_i = I_i \bigcup IP$;

If Marked(T) **Then**
$<$ Mark the pair IP $>$;
End

The complexity of this phase is $O(nts\ m^2\ a)$ where nts stands for the number of nodes of the TS, m for the maximum number of molecules contained in a solution and a for the maximum number of adjacent nodes.

SECOND PHASE

INPUT:
$A = \{$ Extern Processing Elements $\}$
$P' = \{$ Internal Processing Elements $\}$
$PaC = \{$ Passive Connecting Elements $\}$
$I = \{I_i \mid i = 1, 2, \dots\}$

OUTPUT:
$QNM =$ Queueing Network Model

Begin
For all i **Do**
For all $(p_1, p_2) \in I_i$ **do**
If $p_2 \in P'$ **Then**
If $p_1 \in A$ **Then**
Build $CdS_e(p_2)$;
Build $CP([p_1, p_2])$;
If $p_1 \in PaC$ **Then**
Build $CdS(p_2)$;
Build $p_1 \longrightarrow_{trs} p_2$;
If $p_1 \in P'$ **Then**
If Marked(p_1, p_2) and $\mid I_i \mid > 1$ **Then**
$PAIR = \{(p_1, p_2)\} \cup \{(p_3, p_4) \in I_i \mid p_1 = p_3 \wedge p_2 \neq p_4 \wedge p_4 \in P'$
$\wedge$Marked$(p_3, p_4)\}$;
If $\mid PAIR \mid > 1$ **Then**
$PAIR' = \{(p_5, p_6) \in I_i \mid p_1 = p_5 \wedge p_2 \neq p_6 \wedge \neg$Marked$(p_5, p_6)\}$;
If $\mid PAIR' \mid > 0$ **Then**
For all $(p_5, p_6) \in PAIR'$ **do**
If $p_6 \in P'$ **Then**
Build $CdS_1^o(p_6)$;
For all $(p_3, p_4) \in PAIR$ **o**
Build $CdS_1^o(p_4)$;
Build $CdS_1^o(p_1)$;
$CONC = CONC\{\{p_i, p_j\} \mid p_i \in (p_k, p_i) \wedge p_j \in (p_k, p_j) \wedge$
$p_i \neq p_j \wedge p_i, p_j \notin PaC \wedge (p_k, p_i) \in PAIR' \wedge (p_k, p_j) \in PAIR$
$\wedge p_i, p_j \in P'\}$;
Else
Build $SM(PAIR)$;
For all $(p_3, p_4) \in PAIR$ **do**
Build $CP([p_3, p_4])$;
Else
Build $SC(p_1, p_2)$;
Build $CP([p_1, p_2])$;
Else

Build $SC(p_1, p_2)$;
Build $CP([p_1, p_2])$;
If $\neg$Marked(p_1, p_2) and $\mid I_i \mid > 1$ **Then**
  $PAIR = \{(p_1, p_2)\} \cup \{(p_3, p_4) \in I_i \mid p_1 = p_3 \wedge p_2 \neq p_4$
   $\wedge \neg$Marked$(p_3, p_4)\}$;
  If $\mid PAIR \mid > 1$ **Then**
   Build $CdS_1^o(p_1)$;
   For all $(p_3, p_4) \in PAIR$ **do**
    If $p_4 \in P'$ **Then**
     Build $CdS_1^o(p_4)$;
   $CONC = CONC\{\{p_i, p_j\} \mid p_i \in (p_k, p_i) \wedge p_j \in (p_k, p_j) \wedge$
    $p_i \neq p_j \wedge (p_k, p_i), (p_k, p_j) \in PAIR \wedge p_i, p_j \in P'\}$;
  $PAIR = \{(p_1, p_2)\} \cup \{(p_3, p_4) \in I_i \mid p_1 \notin p_3 \wedge p_2 = p_4$
   $\wedge \neg$Marked$(p_3, p_4)\}$;
  If $\mid PAIR \mid > 1$ **Then**
   Build $CdS_1^o(p_2)$;
   For all $(p_3, p_4) \in PAIR$ **do**
    If $p_3 \in PaC$ **Then**
     Build $CdS_c(p_3)$;
   $CONC = CONC\{\{p_i, p_j\} \mid p_i \in (p_i, p_k) \wedge p_j \in (p_j, p_k) \wedge$
   $p_i \neq p_j \wedge (p_i, p_k), (p_j, p_k) \in PAIR \wedge p_i, p_j \in P'\}$;
If $p_2 \in A$ **Then**
 Build $p_1 \longrightarrow ENV$
If $p_2 \in PaC$ **Then**
 Build $p_1 \longrightarrow_{trd} p_2$
If $\mid I_i \mid > 1$ **Then**
 $PAIR = \{(p_1, p_2)\} \cup \{(p_3, p_4) \in I_i \mid p_3 \neq p_1 \wedge p_4 \neq p_2\}$;
  If $\mid PAIR \mid > 1$ **Then**
   For all $(p_1, p_2) \in PAIR$ **Do**
    If $p_2 \in P - A$ **Then**
     If $p_1 \notin PaC$ **Then**
      Build $CdS_1(p_1)$;
    For all $(p_3, p_4) \in PAIR$ **Do**
     If $(p_1, p_2) \neq (p_3, p_4)$ **Then**
      $CONC = CONC \cup \{\{p_i, p_j\} \mid i = 1, 2,\ j = 3, 4,$
      $p_i, p_j \in P - A\}$

For all $CdS_c(p)$ **Do**
 For all $p_1 \longrightarrow_{trd} p_2$ **Do**
  If $p = p_2$ **Then**
   Delete $p_1 \longrightarrow_{trd} p_2$;
   Build $p_1 \longrightarrow p$;
   Build $CP([p_1, p])$;
 For all $p_1 \longrightarrow_{trs} p_2$ **Do**
  If $p = p_1$ **Then**
   Delete $p_1 \longrightarrow_{trs} p_2$;
   Build $p \longrightarrow p_2$;
   Build $CP([p, p_2])$;
For all $CdS_1(p_1)$ **Do**
 need= $FALSE$;

```
    For all SM((p2,p3),...,(p2,pk)) Do
      For i=3 to k Do
        If p1 = pi Then
          If Marked(CdS1(p1)) ≠ o Then
            If need= FALSE Then
              If {p2,pi} ∈ CONC Then
                Build p2 ⟶ pi
                For j=3 to k Do
                  If i ≠ j Then
                    Build SC(p2,pj);
                Delete SM((p2,p3),...,(p2,pk));
                need= TRUE;
          Else
            Build p2 ⟶ pi
            Build CP([p2,pi])
            For j=3 to k Do
              If i ≠ j Then
                Build SC(p2,pj);
            Delete SM((p2,p3),...,(p2,pk));
        Else
          Build p2 ⟶ pi
          Build CP([p2,pi])
          For j=3 to k Do
            If i ≠ j Then
              Build SC(p2,pj);
          Delete SM((p2,p3),...,(p2,pk));
    For all SC(p2,p3) Do
      If p1 = p3 Then
        If Marked(CdS1(p1)) ≠ o Then
          If need= FALSE Then
            If {p2,p3} ∈ CONC Then
              Build p2 ⟶ p3;
              Delete SC(p2,p3);
              need= TRUE;
          Else
            Build p2 ⟶ p3;
            Delete SC(p2,p3);
        Else
          Build p2 ⟶ p3;
          Delete SC(p2,p3);
    If need= FALSE Then
      If Marked(CdS1(p1)) ≠ o Then
        Delete CdS1(p1);
  For all CdS1(p1) Do
    For all CdSe(p4) Do
      If p1 = p4 Then
        Delete CdSe(p1);
        Delete CdS1(p1);
        Build Cdse1(p1);
```

For all $p_1 \longrightarrow_{trd} p_2$ **Do**
For all $p_3 \longrightarrow_{trs} p_4$ **Do**
If $p_2 = p_3$ **then**
Delete $p_3 \longrightarrow_{trs} p_4$;
Build $p_1 \longrightarrow p_4$;
Build $CP([p_1, p_4])$;
Delete $p_1 \longrightarrow_{trd} p_2$;

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 i service centers.
Merge all compatible partial path.

For all $SM((p_1, \ldots, p_i), \ldots, (p_1, \ldots, p_j))$ **Do**
Build $SS_{med}(p)$;
$\mu(p) = a_1\mu(p_1, \ldots, p_i) + \cdots + a_k\mu(p_1, \ldots, p_j)$;
Delete $SM((p_1, p_2), \ldots, (p_1, p_k)$;

End.

The complexity of this phase is

$$O(n_i\ \{c_i^3 + nsc(ttd + tts) + s1(sm\ c_i + sc) + s1\ se + sm + ttd\ tts\})$$

where n_i is the number of sets I_i, c_i is the maximum number of pairs contained in a generic I_i, se the number of generated service center with exogenous arrival, $s1$ the number of generated CdS_1, sc the number of generated SC, sm the number of generated SM, tts and ttd the number of generated Temporary Left and right Transition.
It is worth noticing that we have the following upper bound:

$$c_i,\ sc,\ sm,\ nsc,\ ttd,\ tts \leq N(N-1)$$

$$s1, se \leq N$$

where N is the architecture component number. Thus all factors are polynomial, except for n_i factor that make the algorithm complexity exponential.

Dept. of Pure and Applied Math., University of L'Aquila, L'Aquila, Italy
E-mail address: inverard@univaq.it