Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 97 (2004) 219–239

www.elsevier.com/locate/entcs

Three Performance Models at Work: A Software Designer Perspective[1]

Vittorio Cortellessa[2]

Dipartimento di Informatica
Università di L'Aquila
L'Aquila, Italy

Antinisca Di Marco[3]

Dipartimento di Informatica
Università di L'Aquila
L'Aquila, Italy

Paola Inverardi [4]

Dipartimento di Informatica
Università di L'Aquila
L'Aquila, Italy

Abstract

The validation of software performance since the very early phases of the lifecycle is a crucial issue in complex software system design. Nowadays in the software development practice, the percentage of time and effort allocated to this task is still too small to avoid performance bugs, which are late to discover and hard to fix. This is due both to the short time to market and to the special skills needed (and often lacking in the development team) to effectively accomplish early performance validation.
Software architecture represents a system abstraction that may support the validation and the predictive analysis of system performance. Different notations/languages are available for representing software architectures under a performance viewpoint. In this paper we focus on performance issues of software architectures and we analyze different performance model notations from a software designer perspective. Goal of the paper is to speculate, through a simple case study, on the descriptive power of three largely used performance model notations and their suitability to gain feedback at the architectural design level.

Keywords: Automated Performance Validation, Software Architectures, Queuing Networks, Generalized Stochastic Petri Nets, Stochastic Process Algebras.

1571-0661 © 2004 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2004.04.038

1 Introduction

With the growing complexity and size of modern distributed software systems the need of tools to support design decisions is becoming a critical issue. Independently of the software process, the early design phases may heavily affect the software development and the quality of the final software product. Therefore inaccurate decisions at early phases may imply an expensive rework, possibly involving the overall software system.

Software development teams often have to decide among different functionally equivalent design alternatives relying only on their own skills and experience. This choice is driven by non-functional factors such as performance, reliability, and topological/economical constraints. The criticality of these attributes is high even in software systems where non functional requirements are not explicitly expressed. In fact, in a component based, distributed software system the attributes such as performance, dependability, maintainability determine the quality of the product and the success of a software development. On one side, the problem of assessing the quality of single components is an active research area, on the other side even if it could be possible to assume "good quality" of components the quality of the assembled software system would not always be guaranteed. Hence it is mandatory to pursue further investigation, especially in the early development phases, on how components interact each other and with the environment.

The interest in applying performance methodologies and tools to the software development is very high. The information needed to build accurate models for performance evaluation of software systems, e.g. the hardware platform details, may become available quite late in the software lifecycle. On the other end, the increasing complexity of modern software systems moves the performance focus earlier in the lifecycle. Software Architectures (SA) are the natural candidate for such kind of analysis, and recently much work appeared aiming at carrying out quantitative analysis of software architectures [19,20,21,5,11,3,4]. From a wider perspective, the need of integration of non-functional and functional requirements analysis at architectural level has been recently highlighted in [16].

In the software practice, it is generally acknowledged that the lack of performance requirement validation is mostly due to the knowledge gap between software engineers/architects and quality assurance experts rather than to

[1] This work has been partially supported by MIUR projects "SAHARA" and "Progetto Società dell'Informazione SP4".
[2] Email: `cortelle@di.univaq.it`
[3] Email: `adimarco@di.univaq.it`
[4] Email: `inverard@di.univaq.it`

foundational issues. Moreover, short time to market requirements make this situation even more critical. In this scenario software modeling notations and tools may play a crucial role to fill this gap as well as to shorten the performance validation time.

Nowadays to deal with performance analysis of software systems, several model notations can be used. These model notations fall into two main categories: notations like Queueing Networks that were initially proposed to represent performance features of actual systems, typically hardware or manufacturing systems [14]; notations like (Stochastic) Petri Nets or Process Algebras that were first proposed in the software specification field and then exported in the whole performance domain [1,9,7].

Early in the lifecycle, the choice of the performance model notation is still open. From the software designer perspective, there can be a relevant difference between the above choices. For example, while Queueing Networks are apparently quite far from the software developer knowledge, Process Algebras (or Petri Nets) seem to be closer to the developer viewpoint. Nevertheless, it can be observed that, in the last few years, Queueing Networks constitute the favorite target for performance assessment [4], even in the early lifecycle phases where the software model is still based on abstract interacting components. Moreover, "Queuing Network modeling is a top-down process. The underlying philosophy is to begin by identifying the principal *components* of the system and the ways they *interact*, then supply any details that prove to be necessary" (quoting from [14]). This suggests a very intuitive and natural mapping of Queueing Network with early in the software lifecycle artifacts, like software architectures descriptions.

On the other hand PA and PN have the advantage of importing performance analysis almost for free in their modeling, thus making the performance model construction straightforward at the expense of the behavioral model construction.

Based on these observations, the study described in this paper originates from our interest to investigate the impact that a performance model notation may have on the software development. The lack of studies in this direction has been outlined very recently in [17]. To this extent we consider the three major notations at work in a simple case study, namely Queuing Networks (QN), Generalized Stochastic Petri Nets (GSPN), Stochastic Process Algebras (SPA). The question we would like to answer is: which notation may be more acceptable for a software designer? and under which assumptions on the designer skill and on the software development environment is this true?

To address these questions,we discuss these model notations, by means of the modeling and the analysis of a simple case study, along two dimensions:

1. adequacy to embed and manage performance relevant aspects (e.g., workloads) at the design architectural level;
2. easiness to model, adjust and modify the architectural aspects (e.g., number and type of components) taking into account the possible feedback obtained by means of performance validation.

The team we set up to experimentally compare the considered notations is made of six people that did not have a deep knowledge of any of the notations but were fairly acquainted with software engineering principles. They used the three tools to model the case study, and they reported on the dimensions of our interest.

Far from being a formal classification of performance models, goal of this paper is to highlight the suitability of such notations from a software designer perspective basing on the case study modeling and analysis. Even from the analysis of a simple case study, relevant differences can be devised among performance models along the sketched dimensions.

The paper is organized as follows: Section 2 introduces the experimental framework that includes the architecture design process we propose and the case study we use along the paper, in Section 3 we model the case study by means of SPA, GSPN and QN, in Section 4 we apply the architecture design process to the three models and we comment the obtained results, in Section 5 we compare the three notations, whereas final remarks are presented in Section 6.

2 Experimental Framework

In this section we introduce the elements of our experimental framework, that are: (i) the architecture design process that we propose to take into account performance issues, and (ii) the case study on which we apply the process.

2.1 Architecture Design Process

In Figure 1 we show the design process we consider of a software architecture, enriched by the feedback coming out from the performance validation. At this level, performance is estimated with low knowledge of the hardware platform where the software system will be executed. Therefore, the expected performance feedback consists of the identification of "critical" software components/subsystems whose design needs to be revised.

The primary step consists of building, from an abstract description of the software system, a software architecture model that embeds performance aspects. The output of the performance assessment on the software architecture consists of a set of indices of interest (e.g., component throughput, mean queue length). From the output analysis, some issues may come out.

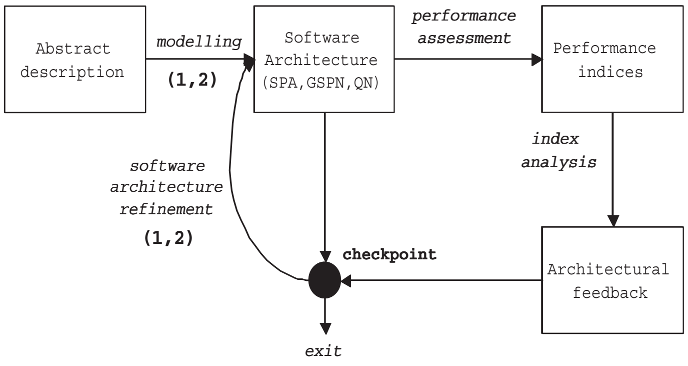

Fig. 1. Architecture Design Process

In response to these performance issues, a range of alternative solutions can be suggested, and these constitute the *architectural feedback* of Figure 1. Being at the architectural level, the techniques to produce alternatives may affect either the components or the communications between them. The techniques we are interested to are the ones closely affecting the components and their workload, which essentially may fall in three categories: splitting, merging and duplication.

- *Splitting* an overloaded component in two or more components means to distribute the set of services provided from the component over a set of newly introduce components. The way of splitting such component is driven by several criteria, including the operational profile. For example, let us suppose that the component provides three services, namely s_1, s_2 and s_3, and their operational profile[5] is expressed by the tuple (f_1, f_2, f_3), where $f_1 \cong f_2 + f_3$. In this scenario a natural alternative could be splitting the component in two new ones, one providing just the s_1 service, and the other providing s_2 and s_3 services.
- *Merging* means to distribute the set of services provided from an underloaded component over a set of existing components. For example, let us suppose that the utilization of each component[6] is among the indices analyzed in the performance assessment step, and let us suppose that one component utilization is under a certain threshold (e.g., 40%). In this scenario

[5] By operational profile, for this specific case, we mean the distribution of frequencies of service invocations.

[6] By utilization we mean the percentage of time the component is busy.

a natural alternative could be distributing the services of this component over other ones whose utilizations allow such an overhead.

- *Duplication* of a component trivially means to create one or more new occurrences of the same component. This type of technique may be used every time the component can not be split, for example because it is a minimal component (i.e. it provides only one basic service) or because of some design constraints that force the software structure.

At the checkpoint in Figure 1 the choice among alternatives is performed considering all software product/process requirements. In practice, the developer must make a tradeoff analysis in order to decide whether and how it is worth to refine the architecture according to the performance feedback.

The numerical labels on the edges of Figure 1 refer to the above introduced dimensions (i.e., adequacy and easiness) and indicate the steps we concentrate to observe and compare the three model notations. The remaining steps may be affected from those dimensions as well, but for the scope of this paper we are more interested in the designer viewpoint rather than to the performance evaluator one.

2.2 Case Study : a XML translator

In this section we present the simple system on which we have based our experimentation.

The software system we consider is called XML translator (XT). It automatically builds an XML document from a text document with respect to a given XML schema [24]. The text document has a fixed structure to allow the automatic identification of its specific parts that are then emphasized by using the XML tags defined in the given XML Schema.

The system reads a text document, and creates a new XML file with the information content of the text document suitably formatted with respect to the considered XML syntax [23] and the XML Schema. The system builds the new file by iterative steps in which identifies useful information and marks it up. Multiple users can concurrently connect to the system and request its services.

From this first description of the system we can identify two distinct software components:

- a *StructureBuilder*, that preprocesses the text file to create its XML related content (i.e. XML special characters) conform to the XML syntax rules. The output of this step is a new text file semantically equivalent to the former, but syntactically different. It also creates an empty XML file according with the established XML Schema, i.e. the file contains only the

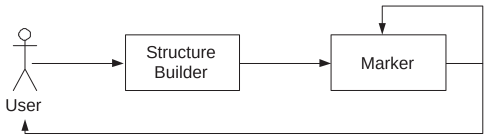

Fig. 2. Static Description of XT system

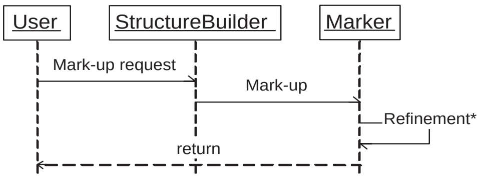

Fig. 3. Behavioral Description of XT system

structure of the document.

- a *Marker* that, by using a heuristic approach, localizes useful information in the text document, singles out it by significant tags from the XML Schema and inserts this chunk of information in the XML file. This component works iteratively on the XML version of the document for an unknown number of times until it is not acceptable (i.e., it does emphasize most of the useful information under certain heuristic conditions).

A static description of XT system is shown in Figure 2, whereas its behavioral aspects are defined by means of the UML Sequence Diagram [22] in Figure 3, which shows that all the interactions are asynchronous.

3 The XT system modeling

In this section we present the SPA, GSPN and QN models of the case study (XT) introduced in Section 2.2. At the end of each model description we report specific considerations about the tool used as well as general thoughts about the notation.

SPA and GSPN are extensions of, respectively, Process Algebra (PA [15]) and Petri Net (PN [18]), that introduce features to model timing and probabilistic aspects of software systems. PA and PN are well-known notations used to model behavior of different types of systems. In the following we assume readers familiar with PA, PN and QN basics [15,18,14].

Across the three models we use a set of common parameters, that are introduced to characterize performance aspects. In particular:

- *lambda* represents the inverse of the user average thinking time, that is the average interval of time between a system response and the following user

```
specification System

behaviour

        (User|||User|||User)|[enq1,arrival]|(Queue1(0,3)|[deq1]|
        StructureBuilder|[enq3]|Queue2(0,3)|[deq2,enq2]|Marker)
where
        process User := (work,lambda); enq1; arrival; User endproc

        process Queue1(n,k) := [n>0] -> (deq1; Queue1(n-1,k)) []
                                [n<k] -> (enq1;Queue1(n+1,k)) endproc

        process Queue2(n,k) := [n>0] -> (deq2;Queue2(n-1,k)) []
                                [n<k] -> (enq3;Queue2(n+1,k)) []
                                [n<k] -> (enq2;Queue2(n+1,k)) endproc

        process StructureBuilder := deq1; (processing,mu1); enq3; StructureBuilder     endproc

        process Marker := deq2; (markup,mu2); (((refinement,p); enq2; Marker) []
                                                  ((backtousers,100000-p);  arrival;  Marker))
endproc
endspec
```

Fig. 4. Stochastic Process Algebra model of XT system.

request;

- $mu1, mu2$ are the service rates of the *StructureBuilder* and *Marker* components, respectively;
- p models the probability of document refinement (namely the heuristic condition in Section 2.2).

Note that $mu1$ and $mu2$ are intrinsic parameters of the software system (i.e. they depend on the internal design of the software components), whereas $lambda$ models the types of users and p the types of documents to be processed.

All of them assume the same meaning, independently of the notation adopted to model the system. So, they will be used as reference values to configure our experiments in Section 4.

3.1 Stochastic Process Algebras

Stochastic Process Algebras (SPA [12,2,10]) permit to label a process action with a rate which may represent either a time value (i.e., estimated time for action execution) or a probability (i.e., frequency of action execution).

We modeled our case study by using the *TIPP* stochastic process algebra [9], which fits our modeling requirements and is supported by a stable design and evaluation tool (namely *TIPPtool* [13]). TIPPtool is a free downloadable software, which allows to edit the model and perform qualitative and quantitative analysis. These tasks are supported by a user-friendly GUI.

The TIPP specification of the XML translator system is shown in Figure 4. It includes two processes, one for each software component identified in Section 2.2, i.e. the *StructureBuilder* process and the *Marker* process.

To properly model the whole system, we introduce two additional processes, *Queue1* and *Queue2*, that represent buffers to store asynchronous service requests addressed respectively to *StructureBuilder* and to *Marker*. An external user is modeled by a special process, *User*, that generates text formatting requests for the system.

The internal behavior of each process is modeled using a standard process algebra semantics. Process actions are nondeterministically composed by the *[]* operator, and each action may be guarded with a boolean expression (e.g., *[n>0]*); action sequencing is expressed by the semicolon operator.

We have simple actions (e.g., *enq1*) and rated actions, whose rates can be used as measures of either their execution times (e.g., *(markup,mu2)*) or their relative execution frequencies (e.g., *(refinement,p)*). The execution time is straightforwardly obtained from inverting the rate value (e.g., *markup* execution time is given by $1/mu2$). The same rate may be also used to transform a nondeterministic choice among actions into a stochastic one. For example, in Figure 4, *refinement* and *backtousers* are rated actions, but since they are placed as heads of two nondeterministic alternatives, their rates also give (besides the standard time meaning) the relative frequency of each alternative. In other words, the *refinement* alternative will be selected with a $p/(p+(100000-p))$ frequency (while its execution time will be $1/p$), and the *backtousers* alternative will be selected with a $(100000-p)/(p+(100000-p))$ frequency (while its execution time will be $1/100000-p$).

Finally, the whole system behavior is specified in the topmost part of Figure 4, where the basic system processes are composed by using the parallel operator $|||$ and the synchronization actions (e.g., *[enq1,arrival]*). For the sake of the example, we show in Figure 4 a system configuration with 3 users.

TIPP tool specific considerations

The particular choice of rates p and $100000-p$ is due to the need of introducing relative frequencies over two alternatives introducing almost no further delays to their execution times. In fact, by varying the interval of values for p from 10000 to 90000 by 10000, we are able to model different stochastic distributions with negligible delays. This artifice is strictly related to the semantics underlying the TIPP Process Algebra. It can be overcome by using a different algebra/tool. For example, the $EMPA_{gr}$ Process Algebra [7] permits to associate priorities and execution frequencies to immediate actions (corresponding to the simple ones of the TIPP algebra).

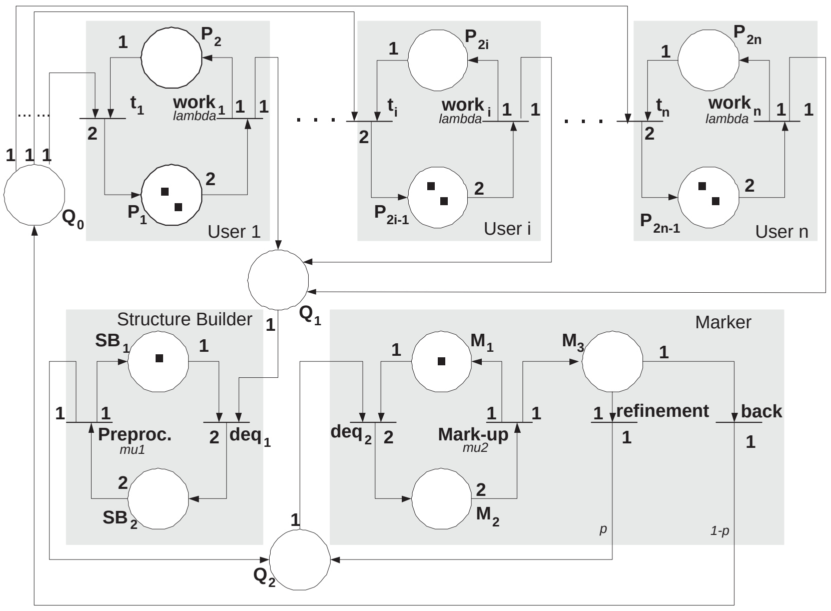

Fig. 5. Petri Net model of XMLtranslator system

Process Algebras general thoughts

PA allow a natural mapping between processes and architectural components. This helps the software designer to describe the software architecture. However, in order to quantify the component behavior the PA specification requires more details on the internal behavior of the components (in terms of the actions each process performs) often not available in the early stages of a development process. With regard to component interactions, PA allow one to easily specify synchronous interactions. In order to introduce asynchronism in communication some additional structures (e.g. processes) are needed, one example is processes that model waiting queues and scheduling policies over queues. On the positive side, SPA allow the specification of a performance model with a notation that is not distant from the one used for software specification, hence attaining the feature of easiness to use from software designers.

3.2 Generalized Stochastic Petri Nets

Generalized Stochastic Petri Nets (GSPN [1]) extend the PN model notation to introduce several features among which timed transitions and stochastic distributions on nondeterministic behaviors.

We modeled the XT system by using a GSPN, and we used for performance analysis the HiQPN tool [6]. HiQPN is a free downloadable software, which (like *TIPPtool* does for SPA) permits the model definition and certain types of analysis, also supported by a user-friendly GUI.

Our GSPN model of the XT system is shown in Figure 5. The lower shaded areas highlight the sub-nets modeling the *StructureBuilder* and the *Marker* components. The higher shaded areas represent the system users that provide text formatting requests to the system.

The generic $i-th$ user is made up of two places: P_{2i-1} represents a busy user (formulating a request), and P_{2i} represents an idle user (waiting for a reply). With the user busy two tokens appear in P_{2i-1}; upon a $work_i$ transition firing, one token goes into P_{2i} to move the user in a waiting state, and one token enqueues to Q_1. Q_1 models the waiting queue of the *StructureBuilder* component.

The same logic applies to the *StructureBuilder* (*Marker*) component, since a token into SB_1 (M_1) represents the idle state of the component, whereas a pair of tokens into SB_2 (M_2) models its busy state. Service requests processed by *StructureBuilder* enqueue to Q_2, which models the waiting queue of the *Marker* component. Service requests processed by *Marker* may be either refined from the same component (enqueued in Q_2) or sent back to the users as replies of accomplished service (enqueued in Q_0).

We assigned a time attribute to every transition modeling the service execution of a component. The timed transitions of the model are: $work_i$, the $i-th$ user is formatting a text; *preproc*, the *StructureBuilder* component is processing a request; *markup*, the *Marker* component is processing a request. All the remaining transitions are immediate. A probabilistic selection rule is applied to the transitions outgoing M_3, in order to model the relative frequency of the $refinement$ and *back* alternatives.

HiQPN tool specific considerations

Since our intent is to consider basic modeling notations, the model of Figure 5 has been built using a minimal Petri Net notation that allows the modeling of performance related features, that are timed transitions and stochastic distributions on nondeterministic behaviors. However, the HiQPN tool permits to build models in extended Petri Net notations, such as Colored GSPN and Hierarchical Queueing PN [6].

The complexity of our model would be lower by using extended notations, but this would mean also a higher PN skill in the software designer that we want instead to keep minimal. However, the choice of adopting such a powerful PN tool (i.e., HiQPN) leaves open the possibility, in future, of considering more complex and demanding models.

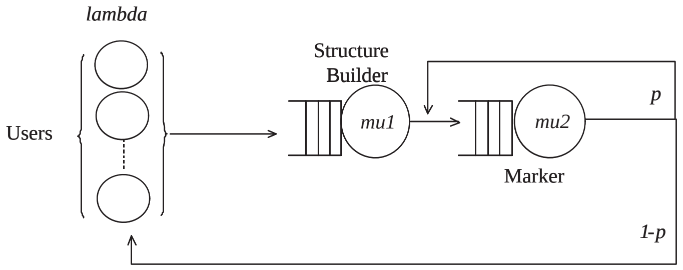

Fig. 6. The XT Queueing Network Model

Petri Nets general thoughts

With basic Petri Nets (PN) the system is modeled from a functional viewpoint, making it difficult to identify components within the model. Indeed there is no direct mapping between PN facilities (places, transitions and tokens) and software components, rather a software component may correspond to a Petri sub-net. The PN notation was originally created to model concurrent systems, so it is especially suited for modeling systems with several loosely coupled components. In cases of highly interacting components, synchronous interactions are obviously modeled, whereas asynchronous ones may require (as for Process Algebras) additional structures. For example a simple priority based scheduling on a waiting queue requires the usage of an extended PN notation, such as Colored Petri Nets. As for PA, out of the above limitations, extensions of PN (such as GSPN) allow to specify a performance model with a notation that is not distant from the one used for software specification.

3.3 Queueing Networks

Queueing Networks are a well-known notation for modeling system performance [14].

To solve the QN model of the XT system we used the Mean Value Algorithm (MVA [14]), since the model is in product form. Our MVA implementation takes as input a text file containing all the parameters needed to the computation (such as number of users, service rates and the QN topology) and gives as output four text files, each containing values of a performance index: utilization, throughput, mean queue length and response time, respectively.

In Figure 6 we show the Queueing Network Model of XT system. It reflects very closely the description in Figure 2. Each component is modeled as a queued service center, while the group of users is modeled as an *Infinite Servers* center. Timing attributes are assigned in a straightforward manner to the service centers. A probabilistic selection rule is applied to the paths outgoing the *Marker* service center to model the relative frequency of the $refinement$

and $back - to - users$ alternatives.

MVA specific considerations

We just like to remark that the possibility of evaluating the QN model by means of the MVA algorithm is due to the simple nature of our case study, which results in a product form model.

Queuing Networks general thoughts

Queuing Networks embed an intuitive mapping between components and service centers. For software modeling at the architectural level, they also provide an immediate way to connect components, that is by means of connections among service centers. Of course, communications among service centers are all asynchronous, based on the queues associated to service centers. Being queues explicitly modeled, different scheduling policies are easy to introduce. Limitations arise in QN when synchronous interactions have to be modeled. In these cases, QN modeling has to add atypical features such as null length queues, and their evaluation may become much more complex. Besides, QN are not well suited to describe details of internal behavior of the components in terms of the actions each component performs. Therefore, in late lifecycle phases, when the software modeling requires more details, QN may not be powerful enough to support software design, so resulting far from common software notations.

4 Models at work: results and comments

In this section we apply the architecture design process (shown in Figure 1) to the XT system, described in Section 2.2 and modeled in Section 3. This experiment is aimed at comparing the ability of the three considered notations to embed feedback coming from performance validation.

We have used two different configurations of model parameters, that in this section we identify respectively as[7]:

- **Fast StructureBuilder:** $lambda = 1.5$, $mu1 = 1.0$, $mu2 = 0.5$;
- **Fast Marker:** $lambda = 0.5$, $mu1 = 1.0$, $mu2 = 1.5$.

For both configurations we assume that other software systems represents the users of the XT system, and this assumption allows such low values for

[7] All the parameter values are expressed in $documents/msec$.

lambda. Each configuration has been evaluated with the probability of document refinement assuming the following values: $p \in \{0.1, 0.5, 0.9\}$.

The comparison is carried on two performance indices that are the mean queue length (i.e. average number of documents waiting to be processed) and the throughput (i.e. average number of documents processed per time unit) of each software component building up the XT system. We study the index trends while growing the number of XT users.

We like to remark that the complexities of the model evaluation processes may sensibly differ from each other, and they may also introduce some approximation errors in the index values. It is out of the scope of this paper to compare the model notations along this dimension, because many other factors would enter into the picture (e.g. product forms, solution tool features).

Due to the low complexity of the XT system, full convergence has been experienced over the performance index values obtained for the considered notations. The result values are shown in the following sections. Observe that the full convergence of the results validates the three models of XT system and assesses their semantic equivalence.

Fast StructureBuilder

Figure 7 shows the performance indices for the Fast StructureBuilder configuration. The results analysis in this case brings the straightforward consideration that the workload of the Marker component is too high. In fact, overall the document types (modeled by p), the Marker throughput saturates for a number of users that goes from 2 to 5.

Note that the Marker component is minimal, in the sense introduced in Section 2.1. Therefore the only suitable feedback alternative consists of duplicating the component itself. At the checkpoint of the architecture design process (see Figure 1), we suppose that the developer opts to refine the architecture by duplicating the Marker component.

The refinement implementation obviously depends on the model notation (i.e. SPA, GSPN, QN). The modifications required to duplicate the Marker component are shown in the following, and all of them require the duplication of the Marker waiting queue as well.

SPA - No new process definition is introduced. The only modification concerns the *behaviour* part of the specification shown in Figure 4, where the subsystem composed by the *Queue2* and *Marker* processes has been duplicated. The new instance of this subsystem runs in parallel to the existing one, and it also synchronizes with the remaining part of the XT system by means of the *enq3* action. The service requests sent to the subsystem are now routed to each instance with a probability of 0.5.

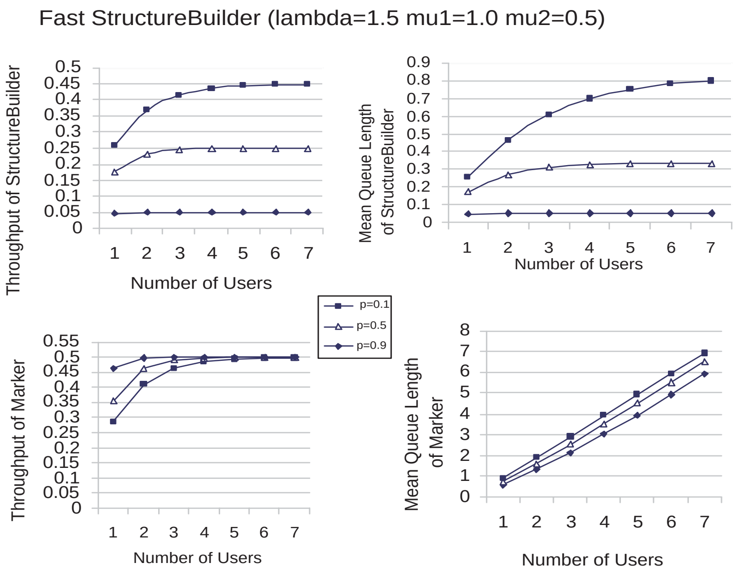

Fig. 7. Fast StructureBuilder performance indices.

GSPN - To model the new XT architecture, the subsystem composed by the Q_2 place and the Marker sub-net (i.e. the one in the Marker shaded area in Figure 5) must be duplicated. In order to connect these subsystems with the remaining part of the system, a new place (namely P) and two new transitions (namely T_1 and T_2) have to be introduced in the GSPN model. The *Preproc* transition now goes into the new place P instead of going in the Q_2 place. T_1 and T_2 outgo the place P, and each one enters an instance of the Q_2 place. T_1 and T_2 are immediate transitions and we associate a 0.5 relative frequency to each one in order to model the same workload for each subsystem instance.

QN - The Marker service center with its waiting queue has to be duplicated. The paths outgoing the StructureBuilder service center enter with 0.5 probability each Marker instance.

New results (shown in Figure 8) are obtained from evaluating the new models. From a quick analysis we observe that the throughput of any Marker instance has decreased and the performance of the whole system has improved, because no evident bottleneck appears over the range of values considered for the number of users. The StructureBuilder component shows a quite high throughput which is still far from saturation.

We judge these results satisfactory for the software designer, thus exiting the process at the checkpoint.

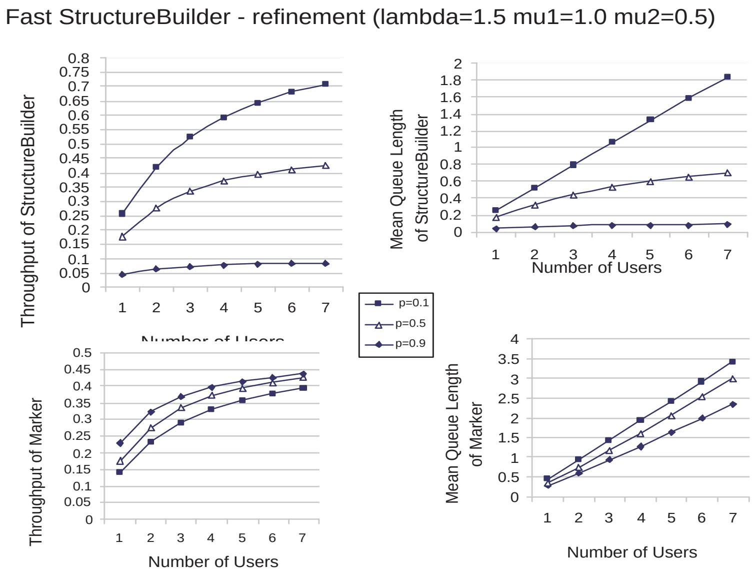

Fig. 8. Performance indices of Fast StructureBuilder refinement.

Fast Marker

Figure 9 shows the performance indices for the Fast Marker configuration. Even here the StructureBuilder and Marker components experience some saturation phenomenon for extreme values of p (i.e., $p = 0.1$ for the former component and $p = 0.9$ for the latter one). The designer may consider acceptable, in this case, the XT behavior since for all the intermediate p values the system seems to perform sufficiently well, thus he/she exits the architecture design process.

5 Summing up: the three notations at glance

In this section, we discuss the lessons learned from the experiment. In Table 1 we show the results that come out from the general thoughts on the considered notations presented in Section 3, and from the experiment report. Of course, the interpretation of the results also take into account the limitations derived from the case study we used. In fact, the case study presents some peculiar aspects, such as all asynchronous communications, small architectural size (i.e., limited number of components), and lack of external sources/sinks of requests (i.e., it is a closed system), that might promote a notation versus the other ones.

In Section 1 we devised the capability of each modeling notation as a

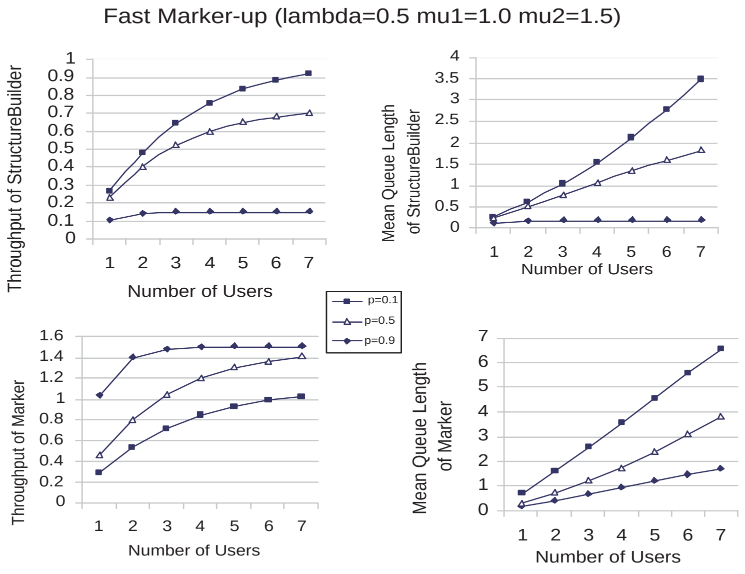

Fig. 9. Fast Marker performance indices.

Notation	Easiness		Adequacy
	To model	To resize	
SPA	Medium	High	Medium
GSPN	Medium	Low	Medium
QN	Medium	High	High

Table 1
Classification of considered notations

combination of ability to describe and refine typical architectural aspects and adequacy to embed and manage performance relevant aspects. In Table 1 these two macro-dimensions are identified as *easiness* and *adequacy*. Easiness divides into *easiness to model* which considers the difficulty to provide the initial model and its refined versions (in terms of their topology), and *easiness to resize* which considers the difficulty to change the system configuration, i.e. to change the number of instances of a component. Adequacy refers to the capability of the model to embed and manage performance aspects, e.g. to express service times. We use a coarse grain numerical scale for these dimensions, with only three ordered values: low, medium and high.

Easiness to model holds medium for QN because although they are quite distant from commonly used design notations, in the early software lifecycle phases there is a natural correspondence with architectural concepts.[8] Easiness to model holds medium also for SPA and GSPN even though they may be considered notations familiar to software designers. Their drawback is that as soon as the system architecture becomes more complicated the complexity of the models sensibly increases.

Generalized Stochastic Petri Nets result difficult to resize. Let us consider, for example, the users issue. In order to modify the number of considered users the sub-net corresponding to the user has to be singled out, duplicated and suitably connected to the network. Stochastic Process Algebras and Queueing Networks are instead easy to resize. In SPA it is sufficient to compose new user (process) instances in parallel, and in QN only an input parameter needs to be changed.

Adequacy is high for QN where performance indices, input parameters and routing probabilities are explicitly considered and managed. GSPN and SPA instead provide performance information less directly. For example, in both cases in order to represent the routing probability, notational tricks needed to be adopted: in the PA two extra actions were introduced, while in GSPN an extra place and two immediate transitions were introduced.

In summary QN seemed to behave better with respect to all the considered dimensions despite their performance analysis aptitude. This should not surprise, as sketched at the beginning of this section, since we are using the QN notation at the architectural level, where behavioral details can be hidden. The limitation of QN lays in their potential distance from the behavioral model. The more behavioral details (possibly internal to components) the software model requires, the more lack of expressiveness the QN notation suffers.

5.1 Considered dimensions vs refinement techniques

The feedback obtained from a performance analysis consists in several suggestions of architectural refinements. In Section 2.1 we have devised three categories of architectural refinements, i.e. *splitting*, *merging* and *duplication*. From a practical viewpoint the characteristics of the adopted notation (i.e., easiness and adequacy) affect the complexity of the implementation of the architecture refinements suggested from the performance results.

Let us separately consider the three refinement techniques. Each component splitting changes the architecture topology, therefore a notation with

[8] This value is not set as high in order to mitigate the particular suitability of our case study to be modeled with QN, due to its asynchronous nature.

a high value of *easiness to model* would be suited to this goal. A similar consideration can be made for merging operations, whereas it is evident that component duplications are better supported from notations with high values of *easiness to resize*. Besides, the application of any refinement technique leads to changes in performance aspects such as workload distribution and routing probabilities among components. Therefore the *adequacy* of the notation is better being high in any case.

Of course these considerations cannot affect our classification of the considered notations. In fact, before starting the architecture design process (Figure 1) it is virtually unattainable to predict what types of refinements will be suggested from the performance results, so the choice of the modeling notation cannot be affected from these considerations.

6 Conclusions

The motivations for the experiment presented in this paper come from the work in the field of software performance and software architectures we carried on in the last few years. The three performance model notations (and their variants) we presented, have been and are largely used. Normally the choice of one of them, as basis of a performance validation approach, is due to several factors which do not consider the user/software-designer perspective. The aim of our experiment was to look at these model notations in order to assess their suitability to support software designers. From the reported results we do not intend to induce general assessments on this field, due to the limitations of the case study and the experimental setting. We rather aim at setting a framework for a campaign of significant experiments in this direction.

From the software designer point of view, QN provide the most abstract / black-box notation, thus allowing easier feedback and model comprehension, especially in a component-based software development framework. For QN the problem remains to easily obtain the model from the behavioral descriptions, especially when a certain level of behavioral detail is required. This is not a problem in the other two models once the designers use the same notations for the behavioral descriptions. Therefore in cases where performance and behavioral analyses are both needed PA and PN notations evidently take advantage.

If we assume a standard development process, with standard software artifacts, like UML-based ones, the effort to produce the performance model from the behavioral description is comparable for all the three notations. In this context, it becomes relevant the existence of algorithms and tools that allow the creation of performance models from standard software artifacts at whatever level of detail. Several automated methodologies have been recently introduced for QN [4] but, to our knowledge, do not yet exist complete methodologies for GSPN and SPA. In order to make performance analysis widely used, future research must focus on the automatization and engineering of existing approaches which integrates standard behavioral modeling with performance model generation, and on the availability of user-friendly frameworks to carry on the analysis.

References

[1] M. Ajmone, G. Balbo, G. Conte, *A class of Generalised Stochastic Petri Nets for the performance evaluation of multiprocessor systems*, ACM Transactions on Computer Systems, 2:93–122 (1984).

[2] M. Ajmone, G. Balbo, G. Conte, "Performance Models of Multiprocessor Performance", The MIT Press (1986).

[3] F. Aquilani, S. Balsamo, P. Inverardi, *Performance Analysis at the software architecture design level*, Performance Evaluation, Vol. 45 n.4, pp. 205–221 (2001).

[4] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, "Software Performance: state of the art and perspectives", Technical Report, SAHARA project, submitted for pubblication (2002), URL: http://sahara.di.univaq.it/tech.php.

[5] L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice", SEI Series in Software Engineering, Addison-Wesley (1998).

[6] F. Bause, A. Klamann, "HiQPN User-s Guide", University of Dortmund (1996).

[7] M. Bernardo, "Theory and Application of Extended Markovian Algebra", PhD. Thesis, University of Bologna, (1999).

[8] J. Bosh, P. Molin, *Software Architecture Design: Evaluation and Transformation*, Proceedings of the 1999 IEEE Engineering of Computer Systems Symposium, (1999).

[9] H. Hermanns, V. Mertsiotakis, M. Rettelbach. *A Construction and Analysis Tool Based on the Stochastic Process Algebra TIPP*, Springer, LNCS 1055, pages 427-430 (1996).

[10] J. Hillston, R. Pooley, *Stochastic Process Algebras and their application to Performance Modelling*, Proc. of TOOLS'98 tutorials (1998).

[11] C. Hofmeister, R. Nord, D. Soni, "Applied Software Architecture", Addison-Wesley, Object Tecnology Series (1999).

[12] K. Kant, "Introduction to Computer System Performance Evaluation", McGraw-Hill (1992).

[13] U. Klehmet, V. Mertsiotakis, "TIPPtool: Timed Processes and Performability Evaluation - User's Guide", Technical Report 1/98, University of Erlangen (1998).

[14] E.D. Lazowska, J. Kahorjan, G. S. Graham, K. C. Sevcik. "Quantitative System Performance: Computer System Analysis Using Queueing Network Models", Prentice-Hall, Inc., Englewood Cliffs (1994).

[15] R. Milner, "Communication and Concurrency", Prentice Hall International, International Series on Conputer Science (1989).

[16] B. Paech, A.H. Dutoit, D. Kerkow, A. von Knethen. *Functional requirements, non-functional requirements, and architecture should not be separated - A position paper*, REFSQ2002, Essen, Germany, September 2002.

[17] D.B. Petriu, D. Amyot, M. Woodside. "Scenario-Based Performance Engineering with UCMNav", Technical Report, SCE-03-07, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada (2003), URL: http://www.sce.carleton.ca/faculty/woodside.

[18] W. Reising, "Petri Nets: an introduction", EATCS Monographs on Theoretical Computer Science, vol. 4 (1985).

[19] ACM Proceedings of Workshop on Software and Performance, WOSP'98, Santa Fe, New Mexico (1998).

[20] ACM Proceedings of Workshop on Software and Performance, WOSP2000, Ottawa, Canada (2000).

[21] ACM Proceedings of Workshop on Software and Performance, WOSP2002, Roma, Italy (2002).

[22] "Unified Modeling Language (UML), version 1.4", OMG Documentation. URL: http://www.omg.org/technology/documents/formal/uml.htm.

[23] "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation 6 October 2000. URL: http://www.w3.org/TR/2000/REC-xml-20001006.

[24] "XML Schema Part 1: Structures" (URL: http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/) and "XML Schema Part 2: Datatypes" (URL: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/), W3C Recommendation 2 May 2001.