Experimenting different software architectures performance techniques: a case study

Simonetta Balsamo Moreno Marzolla
Dipartimento di Informatica
Università Ca' Foscari di Venezia
via Torino 155 30153 Mestre, ITALY
{balsamo,marzolla}@dsi.unive.it

Antinisca Di Marco Paola Inverardi
Dipartimento di Informatica
Università dell'Aquila
via Vetoio (Coppito 1), 67010 Coppito di L'Aquila, ITALY
{adimarco,inverard}@di.univaq.it

Abstract

In this paper we describe our experience in performance analysis of the software architecture of the NICE case study. Naval Integrated Communication Environment (NICE) is responsible for managing and monitoring the heterogeneous equipments composing a naval communication system, by providing several secure communication facilities. The first technique we applied is based on stochastic process algebras with the support of the TwoTowers toolset. For some scenarios of interest this approach suffers of the state space explosion problem, therefore we experimented another technique based on simulation. We describe the application of the two software performance approaches to the case study in order to derive some performance indices at the software architectural level. The case study analysis allows us to point out the relative merit of the techniques, including the performance model derivation, the type of analysis and performance results that we can carry out, and the feedback at the design level, e.g. performance results interpretation that we obtain. Finally, we discuss how to take advantage of the integration of different techniques in software architecture performance analysis.

1 Introduction

In recent years, many approaches to performance analysis of software systems at the software architecture level have been defined. However, their application to real and complex case study is still inadequate although this would help understanding their capabilities, their complexity and their limits.

Moreover, different approaches highlight different figure of merits in terms of performance modeling, analysis and indices that can be evaluated, and of feedbacks at the design level that can derive from the performance results interpretation. In this scenario, the use of different techniques can provide to the software designer a more precise and comprehensive picture on the software architecture. Thus, we believe that the concurrent/complementary application of several analysis techniques can overcome the problems of the single techniques and conduct to more faithful analysis results.

In this work we present the application of two different approaches to software architecture performance analysis to the Naval Integrated Communication Environment (NICE) system. NICE is a system developed by Marconi Selenia which operates in an heterogeneous environment. It must satisfy strict performance constraints on a set of scenarios describing its critical activities.

In [9], we describe the application of a methodology based on process algebra to the NICE software architecture. We used the Æmilia Architectural Description Language (ADL) that is based on stochastic process algebra notation. The Æmilia models were derived from the UML sequence diagrams describing the behavior of the system. Since Æmilia is supported by a toolset allowing the model evaluation, we conducted a performance analysis on the resulting models. However, for some scenario of interest, the approach suffers of the state space explosion problem.

To overcome this problem we experimented another technique based on simulation. We derived a simulation model from annotated UML specifications of the NICE architecture. Annotations provide parameters to the simulator (such as the execution times of each action). The UML description is translated into a process-oriented simulation model, whose execution computes steady-state performance values such as the mean execution time of different scenarios.

In this paper we report on our experimentation of these two software architecture performance approaches on the NICE case study. The goal here is to discuss the advantages and the disadvantages of the applied techniques and to compare these two complementary approaches by highlighting their capabilities, limits, and applicability. Finally, we discuss how to take advantage of the integration of different methodologies in software architecture performance analysis.

The paper is organized as follows: in the next section we introduce the NICE case study. In Section 3 we briefly present the approach based on process algebra, while we more deeply detail the application of the simulative technique on the case study. In Section 5 we compare the two techniques discussing the advantages of their integration in the analysis. Finally, in Section 6 we report some final remarks.

2 The NICE Case Study

The Naval Integrated Communication Environment (NICE) is a project developed by Marconi Selenia. It provides communications for voice, data and video in a naval communication environment. It also provides remote control and monitoring in order to detect equipment failures in the transmission/reception radio chain. It manages the system elements and data distribution services, enables system aided message preparation, processing, storage, retrieval distribution and purging, and it implements radio frequency transmission and reception, variable power control and modulation and communications security techniques. The system involves several operational consoles that manage the heterogeneous system equipment including the ATM based Communication Transfer System (CTS) through blind Proxy computers.

On a gross grain the Software Architecture is composed of the NICE Management subsystem (NICE MS), CTS and EQUIP subsystems, as highlighted in Fig. 1. The WORKSTATION subsystem represents the management entity, while the PROXY AGENT and the CTS PROXY AGENT subsystems represent the interface to control the EQUIP and the CTS subsystems, respectively.

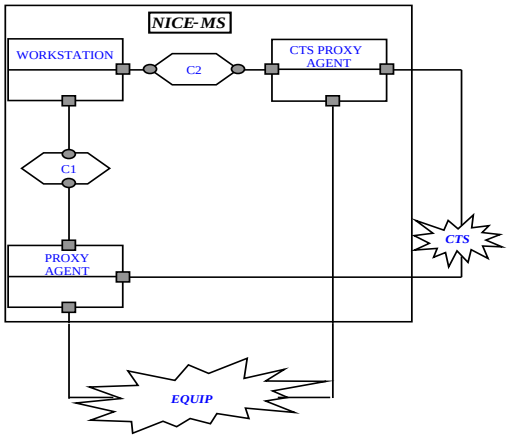


Figure 1: NICE static software description

Actually, the real configuration of the software system will be composed by one instance of WORKSTATION subsystem, two instances of CTS PROXY AGENT subsystem, ten instances of PROXY AGENT subsystem and at least twenty EQUIP subsystem instances. In general, a PROXY AGENT instance manages at least two EQUIP instances.

The more critical component is the NICE MS subsystem. It controls both internal and external communications and it satisfies the following class of requirements: fault and damage management, system configuration, security management, traffic accounting and performance management. All these class of requirements must satisfy some particular performance constraints. For the sake of the presentation, in this work we focus on two scenarios representing two crucial activities of the NICE system: *Equip Configuration* and *Recovery* activities. The two scenarios are described in Fig. 2. The estimated execution times of the various actions are exponentially distributed random variables; the mean values were provided by the system developers and are reported in Fig. 3.

Equip Configuration Scenario

The configuration scenario is activated by an operator when new parameter setting of one or more equipments is required. The system configuration activity consists in a set of actions having the aim the reconfiguration of any equipment(see Fig. 2(a)).

The performance constraint for this activity, as required by the system developers, is: *"The equipment configuration has to be performed within 5 seconds (with a tolerance of 1 second)".*

Recovery Scenario

The activity of system recovery belongs to the class of Fault and damage management requirements. This activity reacts to the failure of a remote controlled equipment. The recovery consists in a set of actions, part of which are executed on the equipment in fault and the others are executed on the CTS subsystem.

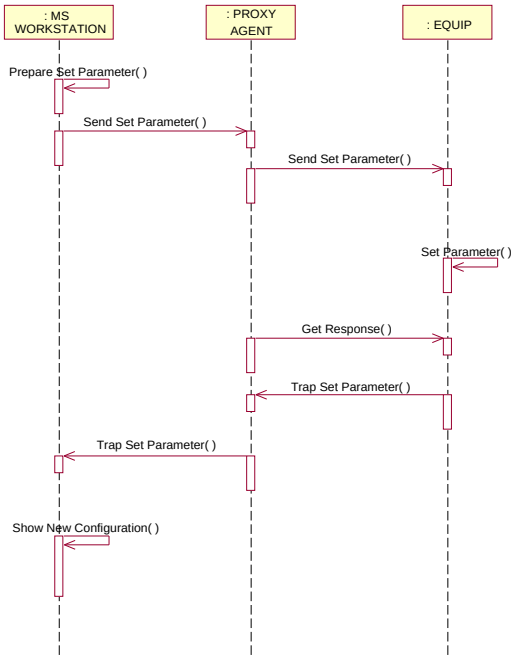
The performance requirement for this activity, as required by the system developers, is: *"A recovery has to be performed within 5 seconds (with a tolerance of 1 second), when a failure occurs".*

For the sake of the modeling, as shown in Fig. 2(b),the WORKSTATION subsystem is decomposed in three main components: COORDINATOR, P1 and P2 where P1 and P2 are auxiliary components interacting with PROXY AGENT subsystem and CTS PROXY AGENT subsystem, respectively, while COORDINATOR represents the control logics of the WORKSTATION subsystem.

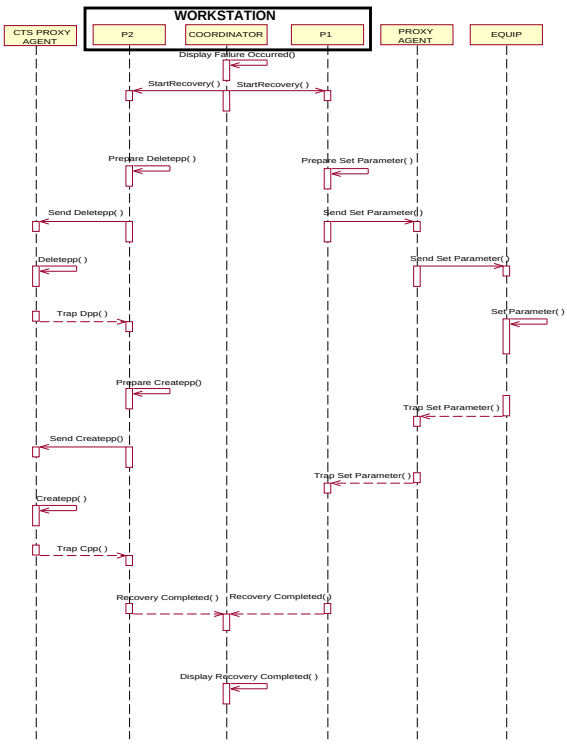
When a failure occurs, COORDINATOR actives two parallel executions (a fork takes place), the one through P1, PROXY AGENT and EQUIP and the other through P2 and CTS PROXY AGENT. When the recovery procedure is completed, COORDINATOR receives a notification from P1 and P2 (a join takes place).

3 Æmilia Modeling

In [9], we have analyzed the NICE system performance by using the tool TwoTowers [7] on an Æmilia model of the NICE system. Æmilia is an architectural description language (ADL) based on the Stochastic Process Algebra $EMPA_{gr}$ [8]. Æmilia aims at making a Stochastic Process Algebra [2, 10] notation suitable for software architectural descriptions. It provides a formal specification language for the compositional, graphical and hierarchical modeling of



(a) Configuration Scenario



(b) Recovery Scenario

Figure 2: Configuration and Recovery scenarios

software systems. Æmilia is supported by a helpful graphical notation based on flow graphs [12] that allows an easier modeling process. Æmilia specifications can be analyzed by means of the TwoTowers tool [7]. TwoTowers provides functional verifications and performance evaluation.

Action	Mean Execution time (sec.)
PrepareSetParameter	1.00
SendSetParameter	0.01
SetParameter	2.00
GetParameter	0.01
TrapSetParameter	0.01
ShowNewConfiguration	1.00

(a) Configuration Scenario

Action	Mean Execution time (sec.)
Display Failure Occurred	0.00
Start Recovery	0.00
PrepareSetParameter	1.00
SendSetParameter	0.01
SetParameter	2.00
GetParameter	0.01
TrapSetParameter	0.01
TrapCreatepp	0.01
TrapDeletepp	0.01
PrepareDeletepp	1.00
SendDeletepp	0.01
Deletepp	1.00
PrepareCreatepp	1.00
SendCreatepp	0.01
Createpp	2.00
RecoveryCompleted	0.00
DisplayRecoveryCompleted	0.50

(b) Recovery Scenario

Figure 3: Mean execution times

In [9], we have shown how to model the NICE behavior by means of Æmilia. Æmilia models have been derived starting from the UML sequence diagrams provided by Marconi Selenia. Performance information on the software components and the performance constraints to be satisfied have been extracted from the NICE technical documentation.

The process we used to derive the Æmilia specification proceeded in three steps:

- *synthesis*from the system scenarios of the statechart diagrams of all components modeling their partial behavior;
- *flow graph derivation* representing the architectural topology of the system;
- generation of the *Æmilia Textual Description* representing the target model. In this step system scenarios were also needed since they contain information on the internal actions executed by the components in the scenario evolution.

By means of TwoTowers, we conducted an analytic analysis of the performance of the models. However, as we already mentioned, for some scenario of interest the approach suffers of the state space explosion problem due to the huge

dimension of the underlying Markov Chain models that the tool builds.

This experience showed us that it is easy to model a system with Æmilia from the software design perspective. This is due to the high expressiveness of the Æmilia language and to its software architecture concepts that make it closer to software developers. Æmilia also presents high scalability feature: whenever the first Æmilia model has been defined, the subsequent changes to the software architecture, in terms of number of component instances, can be simply reflected on the Æmilia model. Its disadvantage as a performance modeling notation is that it requires detailed information on the software component internal behavior. This, in general, might prevent its use at the software architecture level because of the potential lack of detailed knowledge. Another drawback in the performance indices specification is that the TwoTowers input format for indices requires a specific knowledge on the underlying reward theory. Due to this also feedback results at the software architecture level might be difficult to obtain, since there could not be a direct mapping between the analysis results and the performance indices. For example, to evaluate the system response time it is necessary to apply both reward theory and specific laws of queuing theory. This makes the mapping of the performance analysis results on the software architecture difficult.

4 Simulation modeling

An alternative approach for performance modeling and analysis of the software architecture is based on simulation. This modeling approach allows general system representation of arbitrarily complex real-world situations, which can be too complex or even impossible to represent by analytical models. We analyze the NICE system performance by using the tool UML-Ψ described in [4, 3], based on a process oriented discrete-event simulation. We apply the UML-Ψ tool to derive the simulation model from annotated UML use case, deployment and activity diagrams specification, as follows.

Use case diagrams are used to model workloads applied to the system. Actors correspond to open or closed workloads, a workload being a stream of users accessing the system. Each user executes one of the use cases associated with the corresponding actor. Use cases $U_{i,j}$ associated with Actor A_i is given probability $p_{i,j}$ to be chosen. The sum of the probabilities of use cases associated to the same actor must be 1, that is $\sum_{j=0}^{k} p_{i,j} = 1$, for each i. Deployment diagrams are used to describe the physical resources (processors) which are available. Finally, activity diagrams show which computations are performed on the resources. There must be at least one activity diagram associated to each use case. Each action state represents a computation, that is, a request of service from one active resource (processor).

In order to carry on performance analysis in UML-Ψ we add quantitative informations to UML specification, by using stereotyped and tagged values corresponding to a subset of those described in the UML Performance Profile [13].

≪OpenWorkload≫ or ≪ClosedWorkload≫ actors respectively denote unlimited (open) and finite (closed) number of users accessing the system. For actors representing open workloads we specify the interarrival pattern of users with the `RTarrivalPattern` tagged value. For closed workloads we specify the following tagged values:

PApopulation Number of users of the system;

PAextDelay External delay experienced by users.

Each step of an activity diagram is stereotyped as ≪PAstep≫, and can be furtherly annotated with the following tagged values:

PArep the number of times this step has to be repeated;

PAdelay an additional delay in the execution of this step, for example to model a user interaction;

PAinterval the time between repetitions of this step, if it has to be repeated multiple times;

PAdemand the processing demand of the step;

PAhost The name of the host (deployment diagram node instance) to which the service is requested.

Resources are modeled as node instances in Deployment diagrams. Active resources (processors) correspond to nodes stereotyped as ≪PAhost≫. Each node instance can be tagged with the following attributes:

PAschedPolicy (recognized values are "FIFO", "LIFO" and "PS") The scheduling policy of the processor, either first-come-first-served (FIFO), last-come-first-served (LIFO) or processor sharing (PS).

PActxSwT the context switch time.

PArate the processing rate of the host, with respect to a reference processor. Thus, a `PArate` of 2.0 means that the host is twice as fast as the reference processor.

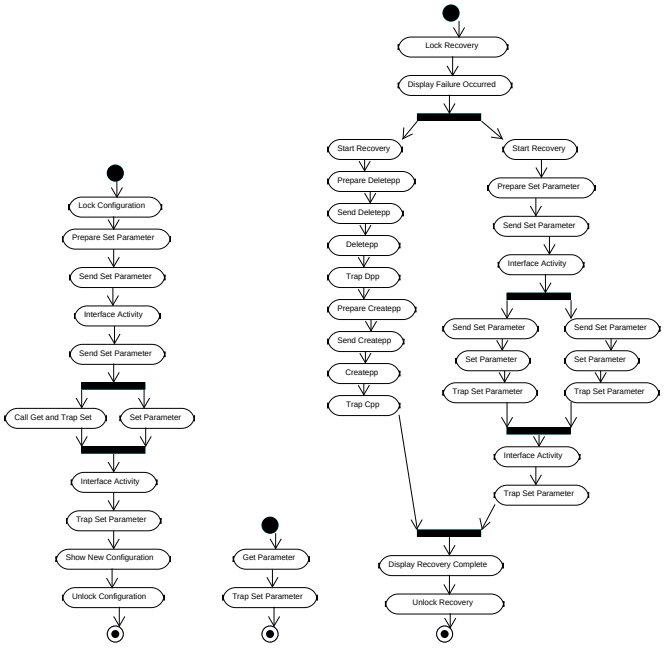
Passive resources correspond to nodes stereotyped as ≪PResource≫. Passive resources have a maximum capacity, expressed with the `PAcapacity` tag. Requests of a resource are done by actions stereotyped as ≪GRMacquire≫, while release of a resource is done by actions stereotyped as ≪GRMrelease≫. If the residual capacity of a resource is less than what requested, the requesting action is suspended until enough resource is available. Pending requests are served FIFO.

The UML-Ψ tool parses the XMI representation [14] of the annotated UML model. Currently the XMI variant used by the ArgoUML [1] tool is supported. From the annotated UML model, a process-oriented simulation model is automatically derived. UML elements are mapped directly into simulation processes in the following way. Actors are translated into processes generating the workload. Deployment node instances correspond to processes simulating the resource with the given scheduling policy, processing rate and context switch time. Finally, each action state in the activity diagrams is translated into a simulation process. When a workload user is activated, it chooses the use case to execute. The activity diagram associated with the selected use case is translated into a set of processes, one for each action state. The simulation process associated with the

starting activity is finally executed. Each step, once completed, starts the successor step until the end of the activity diagram is reached. At that point the workload user is resumed.

UML-Ψ computes the following steady-state performance measures: mean execution time of each action state, mean execution time of each use case, and mean utilization and throughput of the processing resources. These values are computed using the batch means method [5, Chap. 7]. Mean values are expressed in term of confidence intervals, where the confidence level can be specified by the user. The simulation is stopped when the desired accuracy is obtained, that is, when the relative confidence interval widths are less than a given threshold. For simulating the NICE system we set the confidence level to 95% and a 10% confidence interval relative width. Simulation provides estimated performance measures whose mean values are inserted into the original UML model as tagged values associated with the relevant UML elements.

In order to apply the simulation-based modeling technique, it is necessary to translate the sequence diagrams of Fig. 2 into activity diagrams. This can be done easily, resulting in the activity diagrams depicted in Fig. 4(a) (which corresponds to the Configuration scenario of Fig. 2(a)) and the activity diagram of Fig. 4(b) (corresponding to the Recovery scenario of Fig. 2(b)).



(a) Configuration Scenario (b) Recovery Scenario

Figure 4: Activity diagram for the Configuration and Activity Scenario. The Configuration diagram has a composite state ("Get and Trap Set") whose content is described by the small diagram of Fig. 4(a)

Note that the system we are simulating is synchronous, meaning that when an equipment is being configured (resp. repaired), then no other equipment can be configured (resp. repaired) at the same time, but must wait until the current corresponding operation has been completed. In order to simulate this behavior it is necessary to use two passive resources in order to simulate a lock on the scenarios. When executing a scenario it is first necessary to get the lock; if no configuration (recovery) operation is currently running, then the lock is granted immediately. If the lock is not available, the configuration (recovery) request is put on a queue. We compute the mean execution time of the scenarios, including the contention time spent waiting for another running scenario to complete. The service demand for each action state was set as in Fig. 3.

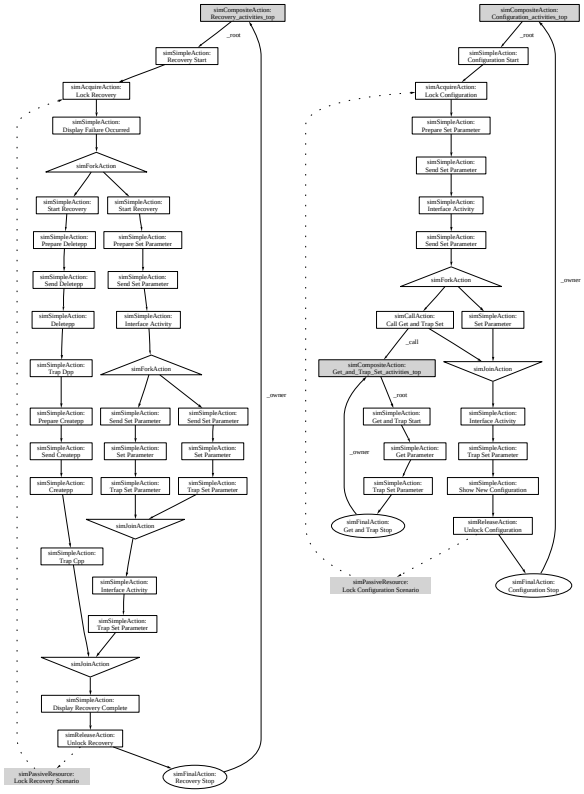


Figure 5: Structure of the simulation model

A simplified representation of the resulting simulation model is given in Fig. 5 where each node represents a simulation process and arrows indicate the process activation order. This graph shows the correspondence between the activity diagrams and the simulation processes. The dotted arrows indicate requests or releases of passive resources, which in our example are denoted as borderless gray boxes labeled as "Lock Configuration Scenario" and "Lock Recovery Scenario". As explained above, these resources are used to guarantee that only a single configuration and recovery scenario is executed at the same time.

We simulate the system considering an increasing number N of equipments, for $N = 1 \ldots 6$. We assume that the time between successive configuration or recovery operations on the same equipment are exponentially distributed with mean $15s$. Simulation results in terms of average execution times of the Configuration and Recovery scenarios are shown in Table 1, which are also plotted in Fig. 6 as a function of N. The table displays also the total execution time of the simulations on a Linux/Intel machine running at 900Mhz, with 256MB or RAM.

N	Configuration Scenario		Recovery Scenario		Simulation
	Mean Exec. Time (s)	Requirement Satisfied?	Mean Exec. Time (s)	Requirement Satisfied?	**Exec. Time**
1	4.02	yes	6.61	no	$56s$
2	4.68	yes	7.64	no	$1m09s$
3	5.50	yes	10.26	no	$1m37s$
4	6.87	no	13.70	no	$1m29s$
5	8.95	no	17.66	no	$2m44s$
6	10.94	no	23.97	no	$2m51s$

Table 1: Computed mean execution times for the Configuration and Recovery scenarios, for different number N of equipments. The last column on the left reports the execution time of the simulation program

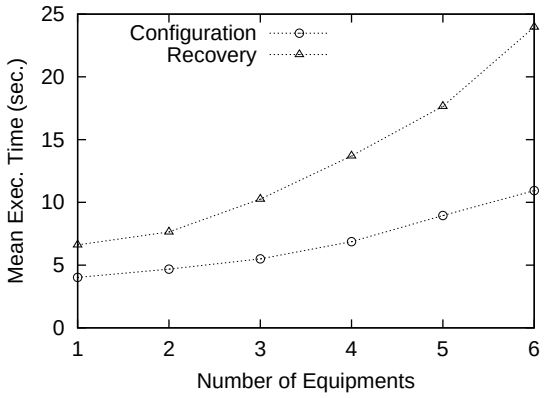


Figure 6: Graph of the execution times from Table 1

5 Comparing the approaches

We now compare the two approaches according to different criteria.

Performance Model Derivation The simulation-based performance model can be easily derived from the UML specification, as there is an almost one-to-one mapping between UML elements and simulation processes [3].
The methodology based on Æmilia SPA-based ADL has nice and useful features inherited both from the process algebra notation and from the ADL. Æmilia expressiveness is high thanks to its ADL nature. From a software architectural specification it is quite simple to derive an Æmilia textual description since it reflects the SA structure. The only drawback of Æmilia in order to carry on a performance analysis at SA level, is strictly related to its process algebra aspects. In fact, we need pieces of information on the internal behavior of the components. This drawback is not evident in our case study since this information is contained in the scenario. In fact, the scenario contains component interactions and method invocations internal to the components with respect to the actions that consume time. Hence, every time we have this information, the Æmilia textual description is easily and automatically derivable.

Software Model Annotation In order to obtain a performance model, it is necessary to provide quantitative, performance-oriented informations which can be used to build the performance model. UML-Ψ requires the UML model to be annotated according to a subset of the UML Performance Profile [13]. Annotations are expressed using standard UML mechanisms (stereotypes and tagged values) which are supported by default by most UML CASE tools. The modeler only needs to know the notation used to specify the values, which in our case is based on the Perl language.
On the other hand, the performance model generated in Æmilia is parametric, meaning that each parameter (such as activities durations) must be instantiated before the model is executed by modifying it.

Generality Both approaches can be applied to any application domain and architectural pattern (such as client/server, layered architectures and others). The approach based on simulation allows general software model, that is, without any constraints on the software architecture model in order to derive the performance model. For example, it is possible to simulate fork/join systems, simultaneous resource possession, general time distributions and arbitrary scheduling policies. Modeling those situations in Æmilia is also possible even if not always easy, as they must be represented indirectly.

Performance Indices The simulation-based approach can derive many different performance metrics. However, the results are expressed in term of confidence intervals. Special care has to be taken in order to apply the correct statistical techniques to remove the initialization bias and compute means from sequence of observations which are usually correlated, requiring many samples (and thus potentially longer execution times) to be taken [5, 11]. UML-Ψ implements the batch means method described in [6] to compute the mean values. The user only specifies the desired accuracy in term of confidence interval relative width and confidence level.
The Æmilia-based approach does not suffer these problems, as the model can be solved analytically by the TwoTowers tool, which computes an exact numerical result. However, this approach shows a drawback in the specification of the performance indices. Indeed the TwoTowers input format for indices requires a specific knowledge on the underlying reward theory.

Feedback The performance values computed by UML-Ψ are inserted back into the original UML model as tagged values associated to the relevant model elements. In this way the user may get an immediate feedback on the system performances.

Obtaining feedback is more difficult using the Æmilia approach. The user may be required to combine different performance measures in order to obtain information related to the software model elements.

Scalability The simulation-based approach is scalable, meaning that the complexity of the simulation model increases linearly with the number of UML elements to simulate. Also, the user may perform many different experiments by changing the UML model, performing a simulation run and modifying the UML model to get better performances before iterating the process again. It is then very easy to perform many "what-if" experiments, changing parameters or structure of the model to see what the result is. Unfortunately, the lack of parameterized results can be a limitation for that. Namely, it is not possible to get performance results as a function of an (unknown) parameter, or set of parameters. It is necessary to explore explicitly all the alternatives by running different simulations to collect the results.
The scalability of the Æmilia-based approach is somewhat limited by the state-space explosion problem. Even for software models of moderate size, the analytical model becomes too complex and overflows the resources available on the host platform. This problem reduces the applicability of the approach in a real context. There can be solutions to the state-space explosion problem, but these solutions require the user to hand-tune the generated performance model. This requires specific skills and expertise, and reduces the automation of the approach and make the integration with the software development process difficult.

Integration The simulation-based approach is at the moment only able to compute performance measures from software architectures. No other kind of analysis are performed on the UML model.
On the other hand, the TwoTowers tool is able to perform both functional and non functional analysis on Æmilia models. It is then possible to start from the same Æmilia specification of the software system to verify different kind of properties.

The comparison is summarized in Table 2.

6 Conclusions

Experimenting generative approaches for performance modeling/analysis on real case studies might help in the understanding of their capabilities, limits and complexity. Moreover, the experiments might give some insights on how to combine different techniques to obtain faithful analysis results.

In this work we have discussed and compared two different approaches to software performance analysis by applying them to the NICE case study from the telecommunication domain. The first technique allows analytical analysis of the performance of the Æmilia model derived from UML sequence (and statechart) diagrams. The second one obtains the performance indices of interest by executing a simulation model automatically generated from UML use case, activity and deployment diagrams.

Both the approaches showed high generality in terms of performance aspects, software system application domains and software architectural patterns. Both of them require specific information on performance aspects of the system that are then used in the analysis. While the simulation technique allows their insertion through the UML activity diagram annotations, the other approach imposes the analyst to insert them directly in the performance model.

Moreover, the presented methodologies showed, as main drawback, the necessity to specify configuration parameters. The simulative technique requires the user to define confidence intervals and the duration of the simulation, while the Æmilia approach requires expertise for the specification of performance indices since the user must have a deep knowledge on the reward theory and queuing theory.

Differently to Æmilia approach, simulation provides a simple feedback mechanism that reports the obtained performance results on the software architecture level.

Finally, the methodology based on stochastic process algebra may require specific skills in software modeling since, in order to derive an Æmilia model, detailed information on the system behavior have to be provided. However, on the positive side Æmilia specification also allows for behavioral analysis.

As largely expected the two methodologies have pro and cons. However, in this work we have experimented the feasibility of a complementary approach at an affordable cost. A big element toward a combined use of the two approaches is the use of standard software artifacts as system initial documents. Obviously we are not considering the expertise in both methodologies required in order to apply the approach.

References

[1] ArgoUML – Object-oriented design tool with cognitive support. http://www.argouml.org/.

[2] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Software performance: state of the art and perspectives. Research Report CS-2003-1, Universitá Ca' Foscari di Venezia, Jan. 2002.

[3] S. Balsamo and M. Marzolla. A simulation-based approach to software performance modeling. Tech. Rep. TR SAH/44, MIUR Sahara Project, Mar. 2003.

[4] S. Balsamo and M. Marzolla. Simulation modeling of UML software architectures. pages 562–567, Nottingham, UK, June 2003. SCS–European Publishing House.

[5] J. Banks, editor. *Handbook of Simulation.* Wiley–Interscience, 1998.

[6] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation.* Prentice Hall, 3rd edition, 2000.

[7] M. Bernardo. *TwoTowers 2.0 User Manual.* http://www.sti.uniurb.it/bernardo/twotowers, 2002.

	Simulation/UML-Ψ	**Æmilia/TwoTowers**
Perf. Model Derivation	Easy. There is an almost direct mapping between UML elements and simulation processes.	Easy due to the ADL Æmilia notation which allows performance models which tightly reflect the software specification.
Annotations	Uses stereotypes and tagged values to specify performance-oriented parameters. Tag values are written in Perl following the specification for the UML Performance Profile [13]	Parameters should be instantiated by the modeler when the performance model is executed.
Generality	No constraint on the software model.	Modeling some particular aspects of software systems, even if possible, can be difficult.
Computed Performance Indices	Only approximate values are computed, which are given as confidence intervals. UML-Ψ implements a set of statistical functions which are automatically applied to output data analysis in order to compute sound results without any user guidance. For many problems (eg removing the initialization bias) there is no single agreed upon technique which can be applied to every situation.	Results are exact numerical values computed analytically.
Scalability	The size of the performance model increases linearly with the size of the UML model.	The size of the performance model can grow exponentially with the size of the software model, thus making the approach difficult if not impossible to apply but for small models.
Feedback	Performance results are inserted into the original UML model, as tagged values associated to the relevant model elements.	It is not easy to associate the computed performance measures to the software components to which they refer.
Integration	Only performance modeling is possible.	Both functional and non functional analysis can be performed on the same Æmilia representation of the software system.

Table 2: Summary of the comparison between the simulation-based and the analytical approach.

[8] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of cocurrent processe with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202(1–2):1–54, July 1998.

[9] D. Compare, A. D. Marco, A. D'Onifrio, and P. Inverardi. Our experience in the integration of process algebra based performance validation in an industrial context. Technical report, University of L'Aquila, 2003.

[10] H. Hermanns, U. Herzog, and J. P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1–2):43–87, Mar. 2002.

[11] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw–Hill, 3rd edition, 2000.

[12] R. Milner. *Communication and Concurrency*. Prentice-Hall International, 1989. International Series on Computer Science.

[13] Object Management Group (OMG). UML profile for schedulability, performance and time specification. Final Adopted Specification ptc/02-03-02, OMG, Mar. 2002.

[14] Object Management Group (OMG). XML Metadata Interchange (XMI) specification, version 1.2, Jan. 2002.