Experimenting different software architectures performance techniques: a case study *

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

Antinisca Di Marco Paola Inverardi
Dipartimento di Informatica
Universitá dell'Aquila
via Vetoio 1, 67010 Coppito, 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 which is responsible for providing several secure communications in a naval communication system. We applied two complementary techniques, one based on stochastic process algebras and one based on simulation, 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 considered 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.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling Methodologies*

General Terms

Performance, Design

Keywords

Simulation, Process Algebra, Performance Modeling

1. INTRODUCTION

Recently, many approaches to performance analysis of software systems at the software architecture level have been defined. Their application to real and complex case study helps 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 [8]. 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, where we used the Æmilia Architectural Description Language (ADL) based on stochastic process algebra. The Æmilia models were derived from the UML sequence diagrams describing the behavior of the system, and performance analysis on the resulting models was conducted. However, for some scenarios of interest, the approach suffered the state space explosion problem.

To overcome this problem we experimented another technique that is 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 of interest.

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.

2. THE NICE CASE STUDY

The Naval Integrated Communication Environment (NICE) is a project developed by Marconi Selenia. It pro-

*This work has been partially supported by MIUR projects SAHARA and Progetto Società dell'Informazione SP4.

vides 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; it implements radio frequency transmission and reception, variable power control and modulation, 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. In NICE-MS the WORKSTATION subsystem represents the management entity, while the PROXY AGENT and the CTS PROXY AGENT subsystems represent the interface to control the equipments (EQUIP subsystem) and the CTS subsystem, respectively.

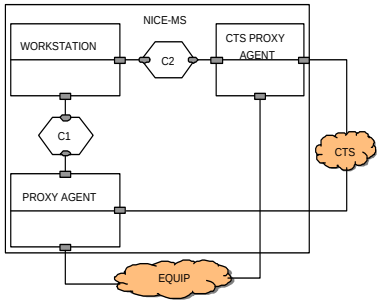


Figure 1: NICE static software description

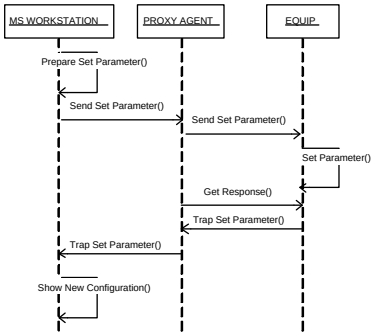
The configuration of the software system will be composed by one instance of WORKSTATION, two instances of CTS PROXY AGENT, ten instances of PROXY AGENT and at least twenty EQUIP instances. In general, a PROXY AGENT instance manages at least two EQUIP instances.

The more critical component is the NICE MS subsystem. It controls both internal and external communications and it satisfies the following classes of requirements: fault and damage management, system configuration, security management, traffic accounting and performance management. All these classes of requirements must satisfy some particular performance constraints.

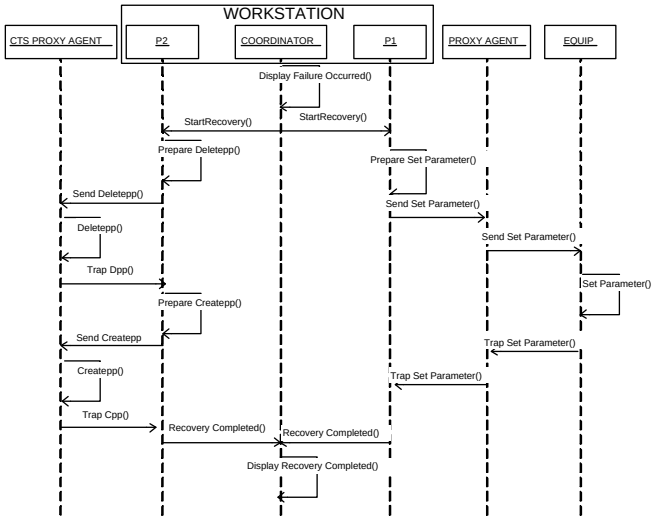
In this work we focus on two scenarios representing two crucial activities of the NICE system: the *Equip Configuration* activity, belonging to the System Configuration requirements class, and the *Recovery* activity belonging to the fault and damage management requirements class. The two scenarios are described in Fig. 2 where the component communications are all synchronous and the WORKSTATION component waits for the completion of the tasks that it triggers. 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



(a) Configuration Scenario



(b) Recovery Scenario

Figure 2: Configuration and Recovery scenarios

of actions to reconfigure the equipments (see Fig. 2(a)). The performance constraint for this activity is: *"The mean execution time of the equipment configuration has to be lower than 5 seconds (with a variance of at most 1 second)".*

`Recovery` *Scenario*

System recovery 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 is: *"The mean execution time of the recovery has to be lower than 5 seconds (with a variance of at most 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 the PROXY AGENT and the CTS PROXY AGENT, respectively, while COORDINATOR represents the control logics of the WORKSTATION.

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

3. Æmilia MODELING

In [9], we have analyzed the NICE system performance by using the tool TwoTowers [5] on an Æmilia model of the NICE system. TwoTowers provides functional verifications and performance evaluation, while Æmilia [7] is an architectural description language (ADL) based on the Stochastic Process Algebra $EMPA_{gr}$ [6]. Æmilia provides a formal specification language for the compositional, graphical and hierarchical modeling of software systems. Æmilia is supported by a helpful graphical notation based on flow graphs [11] that allows an easier modeling process.

Æmilia models of NICE system 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 a single scenario the approach derives the partial statecharts of each component involved in it, which describe the internal behavior of the component. A statechart starts from an initial state in which the component is waiting for taking part to the scenario execution and it contains one state transition for each message (sent, received, to itself) that occurs in the scenario lifeline. The last state transition ends in the initial state, to indicate that the component has finished its "work" in the scenario.

- *flow graph derivation*: from the sequence diagram and the derived statecharts, the flow graph related to the scenario can be automatically derived. From the SA configuration, one or more instances of each component are introduced; for each instance, a box is created in the flow graph. For each component, by looking at the transition in its statechart, a set of input and output interactions is identified. These local interactions are represented by black circles on the box line of the component instance. In relation to the interactions in the sequence diagram, the component instances are linked by arrows in the flow graph. Such arrows are the attachments in Æmilia syntax.

- *Æmilia Textual Description*: it is derived from the statecharts, which represent the behavior evolution of components, and from the flow graph, which contains information on the interactions between instances. Information on the software system configuration is needed to specify the architectural topology. Performance information (such as execution time) are included into the component actions.

By means of TwoTowers, we conducted an analytic analysis of the performance of the models, but for some scenario of interest the approach suffers the state space explosion problem due to the huge dimension of the underlying Markov Chain models that the tool builds.

4. SIMULATION MODELING

An alternative approach for performance modeling and analysis of the software architecture is based on simulation. This approach allows general system representation of complex real-world situations, which cannot be analyzed by analytical models. We analyze the NICE system performance by using the tool UML-Ψ (UML Performance SImulator) described in [1, 2], 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, as follows.

As discussed in [1, 2], use case diagrams are used to model workloads applied to the system (actors correspond to open or closed workloads), deployment diagrams are used to describe the physical resources (processors) which are available, and, finally, activity diagrams show which computations are performed on the resources. In order to carry on performance analysis in UML-Ψ we add quantitative information to UML specification, by using stereotyped and tagged values corresponding to a subset of those described in the UML Performance Profile [12].

Due to the particular characteristics of the NICE system we must introduce the passive resources management. We model passive resources by means of nodes stereotyped as ≪PResource≫ (by referring again to the UML Performance Profile [12]). 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 [13] of the annotated UML model, and 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 [3, Chap. 7]. Mean values are expressed in terms 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 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 of 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 easily done. 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 operation has been completed. In order to simulate this behavior it is necessary to use two passive resources representing 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 immediately granted. 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.

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

5. COMPARISON AND CONCLUSIONS

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 [1].
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 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 information 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 [12]. 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.
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 terms 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. This requires the simulation of many samples (and thus potentially longer execution times) [3, 10]. UML-Ψ implements the batch means method described in [4] to compute the mean values. The user only specifies the desired accuracy in terms 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 specifying indices in TwoTowers 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.

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

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 since it is not possible to get performance results as a function of an (unknown) parameter, or set of parameters. It is necessary to explicitly explore 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, 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. No other kind of analysis can be 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.

Summarizing, as 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 key 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.

6. REFERENCES

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

[2] S. Balsamo and M. Marzolla. Simulation modeling of UML software architectures. In D. Al-Dabass, editor, *Proc. of ESM'03, the 17th European Simulation Multiconference*, pages 562–567, Nottingham, UK, June 2003. SCS–European Publishing House.

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

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

[5] M. Bernardo. Twotowers 3.0: Enhancing usability. In *Proc. of MASCOT 2003*, pages 188–193, Orlando (FL), 2003. IEEE-CS Press.

[6] M. Bernardo and M. Bravetti. Performance measurement sensitive congruences for markovian process algebras. *Theoretical Computer Science*, 290:117–160, 2003.

[7] M. Bernardo, L. Donatiello, and P. Ciancarini. Stochastic process algebra: From an algebraic formalism to an architectural description language. *Performance Evaluation of Complex Systems: Techniques and Tools*, LNCS 2459:236–260, 2002.

[8] D. Compare, P. Inverardi, P. Pelliccione, and A. Sebastiani. Integrating model-checking architectural analysis and validation in a real software life-cycle. In *Proc. of FME 2003: Formal Methods, LNCS 2805*, pages 114–132, Pisa, Italy, 2003.

[9] D. Compare, A. D. Marco, A. D'Onofrio, and P. Inverardi. Our experience in the integration of process algebra based performance validation in an industrial context. Tech. Rep. TR SAH/047, MIUR Sahara Project, Oct. 2003.

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

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

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

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