A Simulation-Based Approach to Software Performance Modeling*

Simonetta Balsamo
Dipartimento di Informatica
Universitá Ca' Foscari di Venezia
via Torino 155, Mestre, Italy
balsamo@dsi.unive.it

Moreno Marzolla
Dipartimento di Informatica
Universitá Ca' Foscari di Venezia
via Torino 155, Mestre, Italy
marzolla@dsi.unive.it

ABSTRACT

Quantitative performance analysis of software systems should be integrated in the early stages of the development process. We propose a simulation-based performance modeling of software architectures specified in UML. We propose an algorithm for deriving a simulation model from annotated UML software architectures. We introduce the annotation for some UML diagrams, i.e., Use Case, Activity and Deployment diagrams, to describe system performance parameters. Then we show how to derive a process-oriented simulation model by automatically extracting information from the UML diagrams. Simulation provides performance results that are reported into the UML diagrams as tagged values. The proposed methodology has been implemented into a prototype tool called UML-Ψ (UML Performance SImulator). The proposed methodology will be illustrated on a simple case study.

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

Unified Modeling Language (UML), Simulation, Performance Modeling

*Work partially supported by MURST Research Project Sahara and by MIUR Research Project FIRB "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools"

1. INTRODUCTION

Software performance is the process of predicting and evaluating whether the software system satisfies performance goals defined by the user [3]. Early identification of unsatisfactory performance of Software Architecture (SA) can greatly reduce the cost of design change [13, 14]. The reason is that correcting a design flaw is more expensive the later the change is applied during the software development process.

The Software Performance Engineering methodology [13, 14] has been the first comprehensive approach to the integration of performance analysis into the software development process. Since then, various classes of performance models of software systems have been considered in the literature [3], including models based on simulation [1, 2, 5].

In this paper we consider the derivation of a simulation model from a SA specification. The approach we describe can be easily extended to model complex architectural patterns involving, for example, simultaneous resource possession and arbitrary types of blocking behaviors. We develop an algorithm for translating annotated UML specifications of software systems into simulation models. We consider SA expressed in term of Unified Modeling Language (UML) [8] Use Case, Activity and Deployment diagrams. The diagrams are annotated according to a subset of the UML Profile for Schedulability, Performance and Time Specification [9] (referred as *UML performance profile*). Annotations provide parameters to the simulation model.

We consider UML representations of SA in term of Use Case, Activity and Deployment diagrams. As recognized in [11], Use Case diagrams provide information about the workloads driving the system, Activity diagrams show the internal dynamics of the system, and Deployment diagrams show the mapping between software entities (processing steps) and hardware entities (processors on which the software executes). We define a process-oriented [7] simulation model of an UML software specification introducing an almost one-to-one correspondence between behaviors expressed in the UML model and the entities or processes in the simulation model. This correspondence between system and the model helps the feedback process to report simulation results back into the original SA. Previous simulation-based performance modeling approaches [2, 5] were developed before the UML performance profile was defined. Thus, they introduced their special extensions to UML or introduced non-standard annotations to express quantitative information useful for deriving the model.

The paper is organized as follows. In Sect. 2 we illustrate

the proposed methodology for generating simulation models from UML specifications. In Sect. 3 we apply the methodology to a simple case study, and conclusions and future works are discussed in Sect. 4.

2. THE METHODOLOGY

The proposed methodology of simulation-base software performance of SA is illustrated in Fig. 1. We consider the SAspecification as a set of annotated UML Use Case, Activity and Deployment diagrams. Annotations are expressed according to a subset of the UML Performance Profile [9]. We derive simulation model parameters from the tagged values extracted from the UML diagrams. Once the process structure of the simulation model is defined, we build its implementation into a simulation program, which is eventually executed. Simulation results are inserted into the UML diagrams as tagged values and can be used to provide a feedback at the software design level. The modeling cycle can be iterated to compare design alternatives and to identify a SA that satisfies given performance requirements.

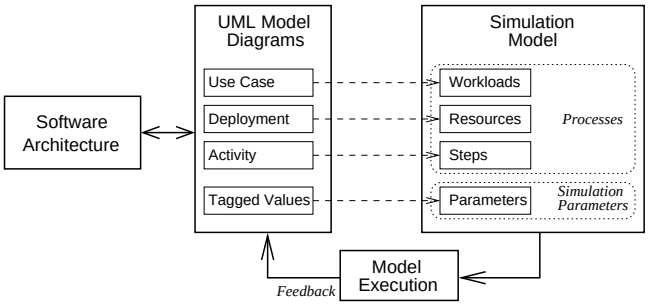

Figure 1: Methodology for simulation modeling of software systems

The proposed methodology has been implemented in the prototype simulation-based software performance evaluation tool UML-Ψ (UML Performance SImulator), which processes an XMI [10] description of UML Use Case, Activity and Deployment diagrams. The UML SA has to be annotated using a subset of the UML Profile for Schedulability, Performance and Time Specification [9]. The simulation results include the mean execution time of each Use Case and each Activity in the Activity diagrams. We also evaluate the utilization and throughput of the computational resources (processors) on which the activities are executed. Fig. 2 illustrates the structure of the performance simulation model derived from the UML diagrams.

The basic object of the simulation model is a `PerformanceContext` which contains the other elements of the model, namely Workloads, Scenarios and processing Resources. Workloads represent requests to the system arriving from external users. The population of users may be infinite (open workload) or with limited, fixed size (closed workload). Closed workloads are characterized by the two following attributes: `PApopulation`, representing the total number of users in the workload, and `PAextDelay` representing the delay between the end of a scenario execution and the beginning of the next request. Open workloads are characterized by the `PAccurrence` attribute, representing the pattern of interarrival times of consecutive requests.

Each workload drives one or more scenarios. Each time a new user belonging to a workload requests service to the system, one of the associated scenarios is selected. Selection is done randomly, according to the probability associated to each scenario.

A scenario is a set of abstract scenario steps, represented by the `AbsStep` class. All kinds steps are characterized by the following attributes:

PAprob Probability to execute this step

PArep Number of times this step has to be repeated

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

PAinterval Time between repetitions of this step, if it has to be repeated multiple times

PAdemand Processing demand of this step

PArespTime Computed execution time of this step.

Abstract scenario steps can either be composite steps (described by the `PScenario` class), or atomic steps of different kinds (`PStep_fork` for fork nodes, `PStep_join` for join nodes and `PStep` for normal atomic steps). Scenarios are collections of steps; exactly one of these steps is marked as the root step (starting step) of the scenario.

Active resources (processors) are modeled as node instances in Deployment diagrams. Such nodes correspond to objects of type `PRhost`. The actual scheduling policy of the processor is implemented by an object derived from class `PRhost_impl`. Each node instance can be tagged with the following attributes:

PAschedPolicy Scheduling policy of the processor, either first-come-first-served ("FIFO"), last-come-first-served ("LIFO") or processor sharing ("PS").

PActxSwT Context switch time.

PArate Processing rate of the host, with respect to a reference processor.

PAthroughput Computed throughput of the resource.

PAutilization Computed utilization of the resource.

Note that the performance model shown in Fig. 2 is slightly different from the one described in the UML Performance Profile [9]. We choose a different structure for the processing steps and active resources class hierarchy in order to simplify the derivation of the simulation model. The UML Profile defines a `PScenario` class from which a `PStep` class is derived, thus every step is a scenario. This is because each step, at a deeper level of detail, could be modeled as a whole scenario, that is, a sequence of multiple sub-steps. We choose a different structure to model the `PStep` and `PScenario` hierarchy to keep atomic steps and scenarios as separate entities. We apply the Composite Pattern [6] to reflect the hierarchical nature of the processing steps. This choice makes the construction of the simulation model easier, because there are different kinds of step (e.g., `PStep`, `PStep_fork`, `PStep_join`) which are modeled as different simulation object types. The class hierarchy representing active resources has also been expanded. We apply the Command Pattern [6] to model hosts with different scheduling policies.

The proposed approach to derive the simulation model to evaluate SA performance from UML Use Case, Activity and Use Case diagrams is defined as the following algorithm:

{Process Use Case diagrams}
for all Use Case diagram U **do**

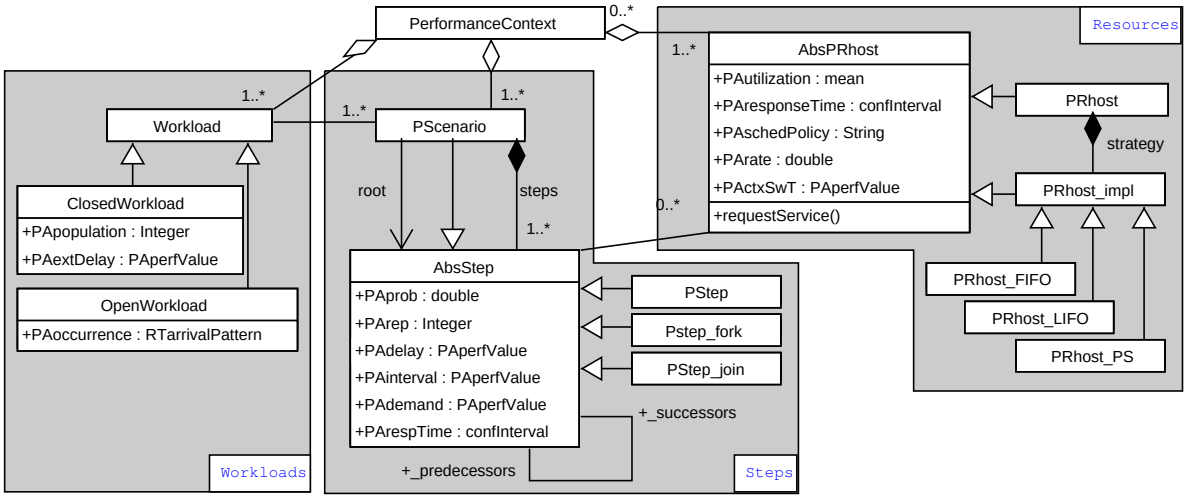

Figure 2: Structure of the simulation performance model

for all Actor $a \in U$ **do**
  if a is tagged as `OpenWorkload` **then**
    $Ac \leftarrow$ Make new `OpenWorkload` object
  else if a is tagged as `ClosedWorkload` **then**
    $Ac \leftarrow$ Make new `ClosedWorkload` object
  for all Use Case u associated with a **do**
    $Sc \leftarrow$Make new `PScenario` object
    Link Sc to Ac
    $A \leftarrow$Activity diagram associated with u
    {Process Activity diagram}
    for all Activity $s \in A$ **do**
      if s is an atomic step **then**
        $P[a] \leftarrow$ new `PStep` object
      else if a is a fork step **then**
        $P[a] \leftarrow$ new `PStep_fork` object
      else if a is a join step **then**
        $P[a] \leftarrow$ new `PStep_join` object
      Add $P[a]$ to Sc
    {Link activities}
    for all $a, b \in A$ such that b is successor of a **do**
      Set $P[b]$ as a successor of $P[a]$
{Process Deployment diagrams}
for all Deployment diagram D **do**
  for all Node instance $n \in D$ **do**
    $p \leftarrow$New `PRhost`
    if n is tagged with `LIFO` scheduling policy **then**
      $p' \leftarrow$New `PRhost_LIFO`
    else if n is tagged with `PS` scheduling policy **then**
      $p' \leftarrow$New `PRhost_PS`
    else
      $p' \leftarrow$New `PRhost_FIFO`
    Link p with p'

The algorithm creates a new process for each relevant element in the UML model. The first step is to identify the Actors in Use Case diagrams. Each actor is translated into an `OpenWorkload` or `ClosedWorkload` simulation process, depending on its stereotype. Then, each Use Case associated with the Actor is examined and translated into a `PScenario` simulation process. To define the content (sequence of steps) of the scenario, the Activity diagram associated with the Use Case is examined. All the activities are translated into the appropriate kind of step (that is, `PStep`, `PStep_fork` or `PStep_join`) depending on the type of the UML element. Finally, each node instance in the Deployment diagrams is translated into a simulation process of the appropriate type, depending on the scheduling policy specified by the user in the tagged values. A more detailed description of the methodology and the tool is given in [4].

3. CASE STUDY

In this section we illustrate with an application example of the proposed methodology described in the previous section. The example is very similar to the one appearing in Sec. 8 of [9]. It involves a web-based video streaming application. We assume a constant population of N users. Each user selects a video to view using a web browser and the browser contacts the remote web server for the video to be sent back through the Internet. The video server triggers the activation of a video player on the workstation of the user before sending the stream of frames. The UML Use Case, Deployment and Activity diagrams are depicted in Fig. 3. Model elements are tagged according to the notation described in Section 2. Note that an analytical model of the system of Fig. 3 cannot be easily evaluated due to the fork/join component in the Activity diagram.

A numerical example of the performance results, computed as steady-state average delays are reported in Table 1. The results are obtained by setting a population of $N = 20$ users and the other model parameters are set from the tagged values as shown in Fig. 3. The simulation took less than 2 seconds on a PentiumIII PC running at 900Mhz using the batch mean method. The computed average execution time of the scenario is the interval $[65.3993, 65.7069]$ seconds. The computed simulation results are intervals obtained at 90% confidence level and with a maximum confidence interval width of 5%. Such results are inserted into the original UML diagrams as values of the `PArespTime` tag associated with each Activity, and the `PAutilization` and `PAthroughput` tags associated with each Node instance in the Deployment diagram. These tags (not shown in the

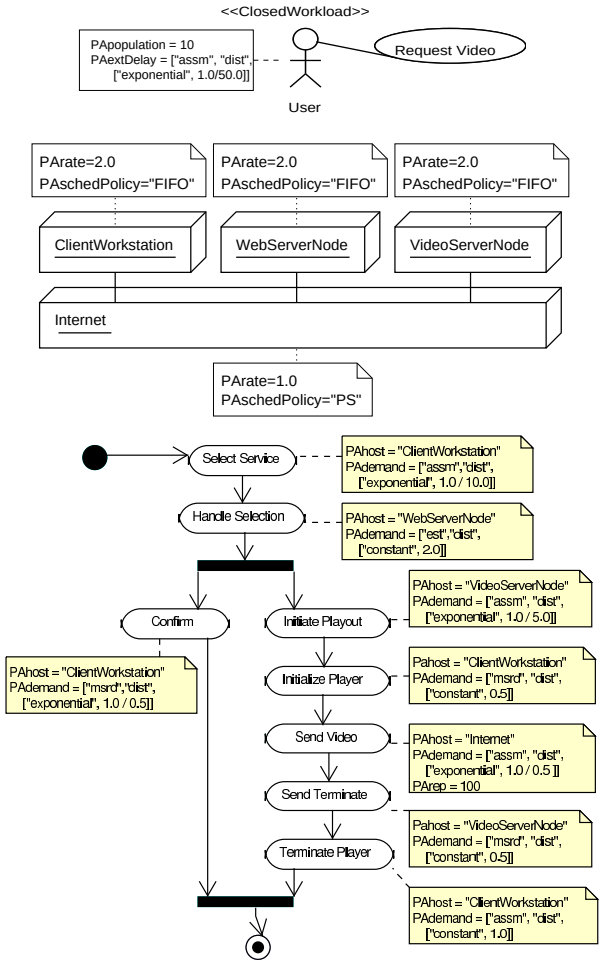

Figure 3: Simple Web video application.

figure) must be inserted, with any value; the value will be overwritten with the simulation results. The results show that the most utilized resource is the network. The software designer can now explore different scenarios by repeating the modeling and performance evaluation process with different parameters until the SA satisfies the performance requirements.

4. CONCLUSIONS AND OPEN PROBLEMS

We have proposed a simulation-based software performance modeling approach for UML models. Performance parameters are introduced in the specification model with an annotation which is a subset of the one defined in the UML Performance Profile [9]. The proposed algorithm derives the process-oriented simulation model from UML Use Case, Activity and Deployment diagrams. In the future we plan to include Sequence diagrams as an alternative to Activity diagrams for describing the dynamic behavior of the system [9]. The use of Deployment diagrams will be extended to include passive resource modeling, as well as active resources. Further research will be devoted to implement a more complete set of performance measures and to integrate the proposed methodology into a more general Software Performance Evaluation framework.

Table 1: Simulation results for the SA of Fig. 3.

Utilization	Throughput (requests/sec)
ClientWorkstation	
[0.183559, 0.183619]	[0.0611551, 0.0611941]
WebServerNode	
[0.0152827, 0.0153439]	[0.0152827, 0.0153439]
VideoServerNode	
[0.0432813, 0.0433357]	[0.030573, 0.0305853]
Internet	
[0.764314, 0.764321]	[1.5285, 1.52852]

5. REFERENCES

[1] L. B. Arief and N. A. Speirs. Automatic generation of distributed system simulations from UML. In *Proceedings of ESM '99*, pages 85–91, Warsaw, Poland, June 1999.

[2] L. B. Arief and N. A. Speirs. A UML tool for an automatic generation of simulation programs. In Proceedings of WOSP 2000 [12], pages 71–76.

[3] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Software performance: state of the art and perspectives. Tech. Rep. MIUR SAHARA Project TR SAH/04, Dec. 2002.

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

[5] M. De Miguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, and S. Piekarec. UML extensions for the specifications and evaluation of latency constraints in architectural models. In Proceedings of WOSP 2000 [12], pages 83–88.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of reusable Object-Oriented programming*. Addison–Wesley, 1995.

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

[8] O. M. G. (OMG). Unified modeling language (UML), version 1.4, Sept. 2001.

[9] O. M. G. (OMG). UML profile for schedulability, performance and time specification. Technical Report ptc/02-03-02, OMG, Mar. 2002.

[10] O. M. G. (OMG). XML Metadata Interchange (XMI) specification, version 1.2, Jan. 2002.

[11] R. J. Pooley and P. J. B. King. The Unified Modeling Language and performance engineering. In *IEE Proceedings – Software*, volume 146, pages 2–10, Feb. 1999.

[12] *Proc. of WOSP 2000*, Ottawa, Canada, Sept. 2000. ACM Press.

[13] C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, 1990.

[14] C. U. Smith and L. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison–Wesley, 2002.