

# Performance Assessment of Software Models In a Configurable Environment Simulator

Michael Barth,  
Institute of Computer Science, Ludwig-Maximilians-Universität,  
Oettingenstr.67, 80538 München, Germany

**Abstract** *Performance assessment of software from the early design phase through the implementation has been subject to a great variety of approaches in the past. This proposal tries to achieve performance assessment of software models by using only UML diagrams describing all essential parts. The developer should be able to use this technique without the introduction of difficult mathematical models. Dynamical properties of the software we lay down in activity diagrams. The environment that will execute the software later is modeled by class diagrams. Finally a simulating software configured by the environment description rates the software model. Activity and data models to be used in activity diagrams. Resource models to be used in environment descriptions. The elements are held in several packages. All packages are easily extendable. Model elements can grow as the views on the design are refined during the software development process. We give an overview over this modeling technique and the concepts of the simulator.*

**Keywords:** Performance Modeling, Activity Diagrams, Simulator, VirtualEnvironment, UML

## I. INTRODUCTION

Performance often is the crucial aspect in developing software. In most cases only small parts of a software's functionality have to comply to certain requirements. Modeling techniques, especially modeling with the UML in a software development process, are accepted by many developers. Every modeler should be able to express performance requirements in UML if this is the methodology is used to work in. In a top down approach we want to be able to refine all models and we will prefer not to change the formalisms during the process. Therefore we are forced to look for simple models.

We introduce a method to specify dynamical properties of software with the UML in a simple way that is intuitively understandable. We also want to model the set of resources that will execute the software later. Finally we show how to assess the model's performance properties in each phase of the project.

Many approaches for performance modeling have been proposed in the past such as stochastic timed petri nets, stochastic process algebras, queuing network models, etc. These approaches are clearly summarized in [bal01, State of the Art]. Many proposals have been made to derive these more difficult models from UML models more or less automatically; see [bal03], [pet01], [bal02], e.g. However, the modeler is responsible for the usage of proper models. This decision often depends on a difficult context. We tried to to achieve a maximum of acceptance with UML diagrams as simple as possible.

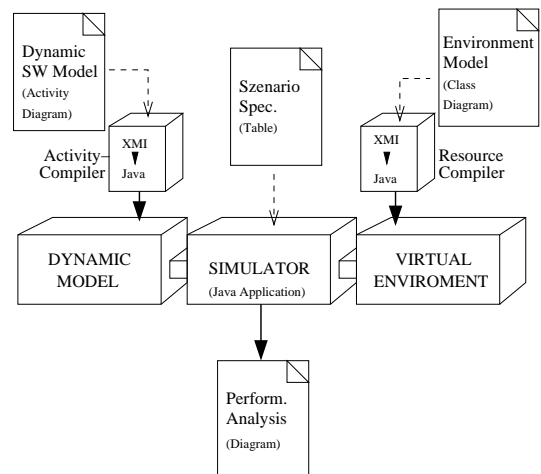


Fig. 1. Simulator

Dynamical aspects of software functions are modeled with activity diagrams. The control flow is modeled with activity states and transitions. The data flow is modeled also in activity diagrams using Object Flow States. These diagrams are translated into a dynamic model. The environment is described in a class diagram. All hardware components are objects connected with associations. These diagrams are translated to a virtual environment model. Finally, particular cases and scenarios can be defined in a scenario specification. This scenario can be assessed by a simulator cal-

culating the response time and the dynamical behavior. An overview of this simulating environment and its requirements is given in fig. 1.

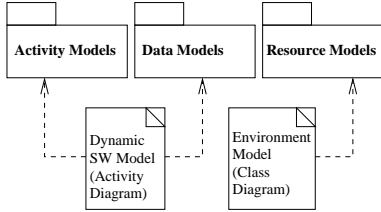


Fig. 2. Model packages

We need activity models and data models to create diagrams describing the dynamical behavior and resource models to create the environment description (fig. 2). In section II we explain these elements of the performance model.

In the top down process we also have to refine the environment description. We start with abstract system boundaries and proceed with real hardware resources at different levels. We must be able to construct new appropriate models during the development process. This is shown in section III.

Finally we want to give an overview over this methodology with a simple example in section IV.

## II. ELEMENTS OF THE PERFORMANCE MODEL

Time is consumed only during the execution of a job by a resource. We can assess time consumption only if there are sufficient informations about all resources (the environment), the control flow and the data flow (the dynamic model) of the software. In this section we introduce an overview over all necessary elements for the performance assessment of software models.

### A. Dynamic models

We apply UML activity diagrams to describe dynamic properties of software models. These diagrams are created using a standard graphic tool for modeling and we store it in a file in XMI format. This file is read by the first prototype of an activity compiler and translated into a Java activity model.

The activity compiler supports Java classes UML activity graph elements, like activity states, forks, joins, transitions, object flow states and so on. Activity states and state transitions model the control flow we want to examine in activity diagrams.

1) *Activity / job classes*: Every activity state of our model represents a job that is to be executed by a resource. In general, only activities that consume execution time or space will be modeled. To specify the type of job that is to be executed, the developer defines

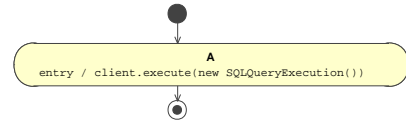


Fig. 3. Execution of an activity

the /ENTRY action of the activity state as shown in the example in fig. 3. We provide an extensible package of models of executable jobs. The developer chooses an appropriate model out of this package and defines the appropriate resource to execute an instance of this job.

The job models generally contain information that is essential to calculate the processing time. This information, e.g., is about the number of operations to be executed by the processing unit, memory that is consumed, etc. In the example in fig. 3 the resource CLIENT is about to execute an instance of a job of the type SQLQUERYEXECUTION.

Moreover, the example implies that required memory is only allocated during the execution of the job and is released immediately after the process terminates. Persistent consumption of memory is illustrated in the examples shown in fig. 14 below.

2) *Data classes*: OBJECTFLOWSTATES model the data flow in activity diagrams. These states describe data objects, created by activities, or consumed by using them as parameters.

The data package elements can classify OBJECTFLOWSTATES. We deliberately abstract from the data contents of objects, but rather concentrate on the memory relevant parameters.

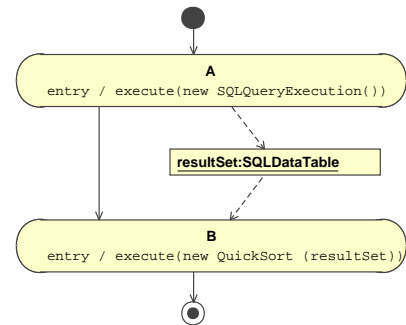


Fig. 4. Data flow modeling

The example in fig. 4 shows transitions connected to the data flow state leaving one activity state and entering another. This implies the time for processing both jobs specified by the activity depends on the size of the data object.

Furthermore, the data object consumes memory

only during its lifetime. The lifetime starts immediately after termination of A's /ENTRY action and it ends when B's /ENTRY action starts. The consumption of memory causes effects during the execution of A and B, however, this is modeled by the specified activity/job classes, SQLQUERYEXECUTION and QUICK-SORT. Both classes have to model additional load. Persistent data objects are modeled by introduction of the stereotype PERSISTENT as described shortly.

### B. Resource classes

We have to describe the environment that will execute the software. The environment is a set of resources mostly hardware. Types, sizes, speed and structure of the resources influence the performance properties. The environment description is laid down in an UML class diagram.

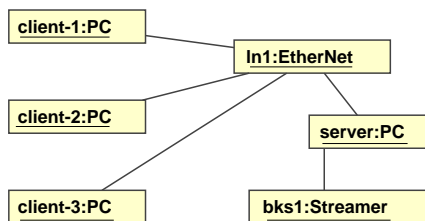


Fig. 5. Simple example of an environment

Each object in this diagram describes one resource. The resource model package shown in fig. 2 contains resource model classes. (The package can be loaded into the modeling tool.) Every object is classified by the appropriate model class. If resources are connected together this can be shown by an association. Both models must own matching interfaces. This is shown in an example in fig. 5. If no appropriate class can be found in the resource package new models can be developed or composed. The extension of the package is described in section III.

The class diagram is stored in XMI format. For each resource model an implementation is provided to calculate the consumption of capacity while the execution of a job. The resource compiler reads the XMI file and creates an environment model as Java objects. This model is used as an "arithmetic unit" by the simulator (fig. 1).

The simulator interprets the software model like a real resource would execute the software. However, the result of the simulator's computation is the performance analysis.

### III. EXTENSION AND CALIBRATION OF RESOURCES

The environment can be modeled as described in section II-B. Only prepared classes were used here. In

some cases none of the prepared classes will fit. These might be, e.g., use cases that are executed in an abstract system, special database operations (high level), or special interrupt calls (low level). The modeler needs to extend the package or develop new resource models.

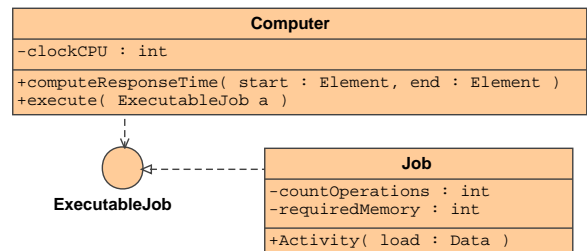


Fig. 6. Resources and their specific jobs

However, running a job requires specific resources (fig. 6). Appropriate pairs of resources and jobs must result from the combination of dynamic model and environment model. If new resources are developed possibly appropriate job models have to be developed as well. In this section we describe some extension mechanisms with a focus on resource models.

### A. Using package capabilities

New resources can be created only by using available classes and the capabilities of the resource package.

1) *Modifying resources:* The performance relevant attributes of a resource class will be initialized with predefined values during the compilation. The simplest way to create new resources is to change these values.

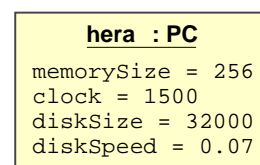


Fig. 7. A modified resource

All attributes that describe the capacity of the resource components can be explicitly initialized in class diagrams. For example CPU clock, memory sizes, etc. can be initialized with different values as shown in fig. 7. The resource compiler will create instances initialized with the values specified in the XMI files. The modeler can assess effects of faster CPU's, more memory, etc. on the performance of your software this way.

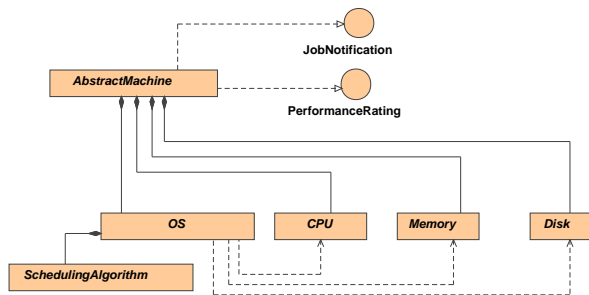


Fig. 8. Design patterns of a resource

2) *New resource combinations*: Some resources are modeled as composites of basic components or other resources. In fig.8 we show an example pattern of a computer. The computer in this example is not modeled by using a single class but by a composite of its basic components. All patterns are available as abstract classes. The modeler can combine appropriate components according to the pattern to new resources. The abstract classes are replaced by chosen subclasses. (The abstract classes in fig. 8 could be replaced with subclasses modeling a SD-RAM memory, a Pentium\_III microprocessor , or a Linux operating system, e.g.) This can be done in a separate class diagram or in the environment model itself. It is possible to redefine attributes as shown in section III-A.1 as well.

### B. Extending resource package classes

If no suitable subclasses can be found the modeler has to create new classes using inheritance. He will overwrite the required methods and implement the desired behavior their. The method calculating the response time is of dominating interest here. This can be realized in several ways.

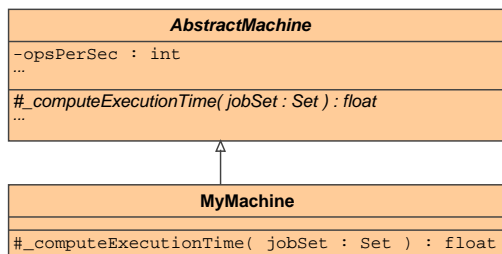


Fig. 9. Extending of abstract resources

1) *Implementing load functions*: In our example in fig.9 you have to implement the method `_COMPUTEEXECUTIONTIME`. The processing of an abstract job depends only on the number of (abstract) jobs to be executed in this example. A simple function can

be implemented, describing the relation between the number of jobs to be processed and the time elapsed while executing them. Any desired and also more sophisticated mathematical models may be chosen and implemented in this method.

This strategy is preferred in the early phases of a project, where no information of exact structure of the later implementation is available. Experts may estimate performance properties of use-cases (a simple linear dependency for example). The first architecture draft may be based on the estimations. (This strategy is also useful if the function of the dependency is known.)

The results of performance assessment with growing accuracy will influence the workflow of the project further on. New experiences in the proceeding project may improve the estimations. The modeler has to redefine and update the methods then. When parts of a project have been realized, measurements may replace estimations. (A special methodology may enhance common software development processes in future.)

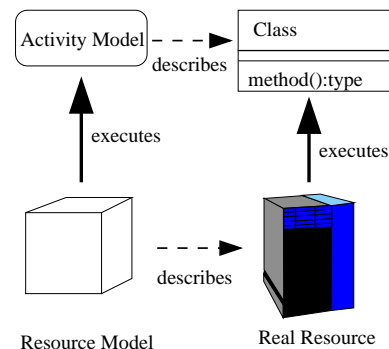


Fig. 10. Relations between model and reality

2) *Calibration of performance functions*: The most elegant methodology to implement the performance calculating methods, is generating code from performance measurements. We expect activity models (job models) to behave in our simulated environment regarding performance aspects analogous to real implemented jobs on real existing resources as shown in fig. 10. The modeler has to develop exemplary implementations of all types of jobs he is interested in. He has to develop also data objects, if they are needed as parameters. (Experimental this was mostly C++ code.) Java classes wrap these examples as native code. These examples were now used as probes to monitor a real resource by a software monitoring tool, specially developed for this purpose. An appropriate activity/job model has to be assigned. Ranges, numbers of parallel executions and the size of parameter loads must be defined for the measurement. The monitoring tool starts a testing thread, runs the probes on the resources. These tests are repeated within the defined

ranges and for the defined parameters. The resources net capacity is calculated by subtracting measured values of empty wrappers from measured values of charged wrappers.

If the survey was successful, the results are stored in a data file. Only a limited number of knots can be measured with this methodology of course. The knots are mathematical interpolated with linear functions or polynomial splines.

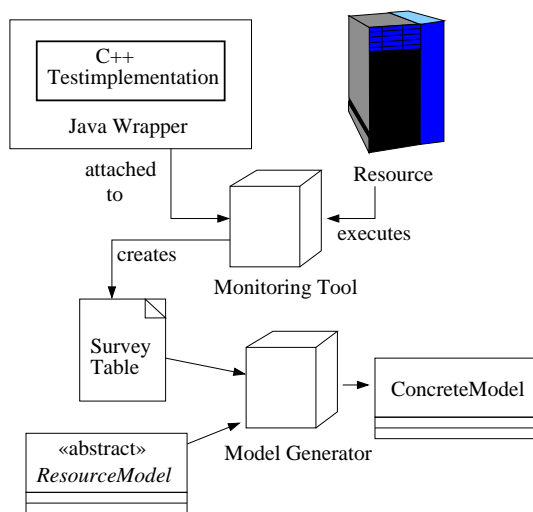


Fig. 11. Model generation by measurement of resources

Now the modeler can select an abstract resource model. A model generator creates Java code according to the chosen abstract resource or resource pattern. The model generator reads the survey table and generates code (a polynomial spline, e.g) for the performance calculating method from the parameters. The modeler can use the new resource model after compiling the code and adding the new model class to the resource package.

#### IV. EXAMPLES

In this Section two examples will illustrate aspects of the performance modeling in practice. The first common client/server example gives an overview that is intuitively understandable. The second example explains the particular problem of modeling persistent data.

##### A. Client / Server

In fig.12 a simple model of a client/server communication is shown. This model corresponds directly with the resource model shown in fig.5. Some of the activity models are taken from predefined classes, some are extensions of the activity package. Recall the detailed

semantics we already introduced: the objects are not persistent.

To calculate the time that is needed to execute these activities we must define some values. The size of DATAFORM and DOCUMENT is not specified in the diagram. Obviously we need this information. In a scenario specification table we may specify only the range of DATAFORM like: “DATAFORM.SIZE>0; DATAFORM.SIZE<20000;”. The printable document’s size might depend on DATAFORM like: “DOCUMENT.SIZE:=5000+(DATAFORM.SIZE\*120);”. With this exemplary specification the simulator is able to rate the response time of the execution of this activity diagram in the specified virtual environment within the range of DATAFORM’s size. A diagram of the performance analysis document may look like the example in fig. 13.

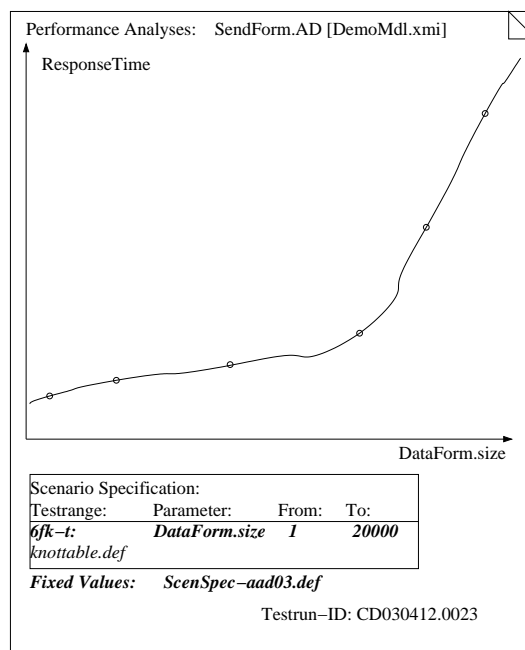


Fig. 13. Overview over the performance in a diagram

The simulator computes a limited number of knots and delivers a graph, that shows the response time depending on the size of DATAFORM.

##### B. Modeling of persistent data

To describe persistent data we introduce the stereotype PERSISTENT. This stereotype is used together with two other new stereotypes CREATE and CONSUME. The detailed semantics is as follows: the PERSISTENT data object is created by an activated transition of the CREATE stereotype. The object does exist until it is destroyed by the activation of an /ENTRY

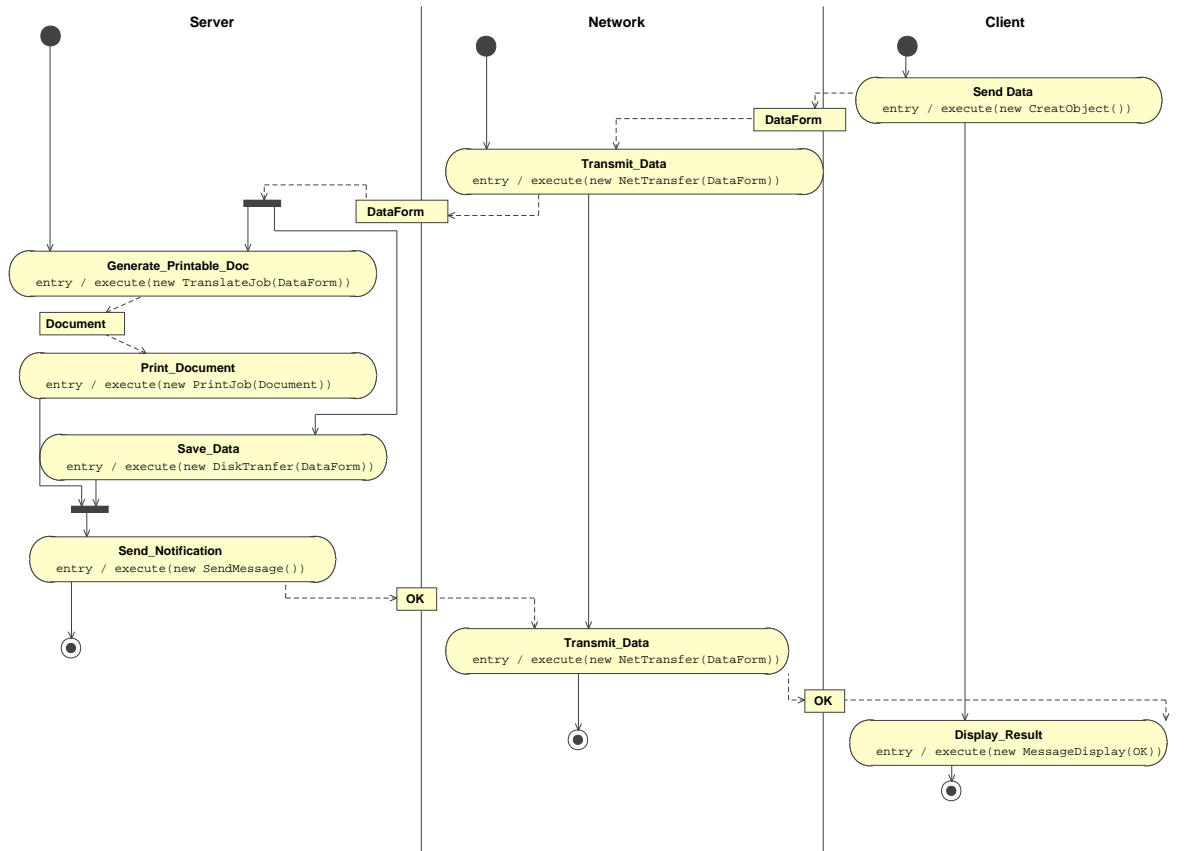


Fig. 12. Simple client/server example

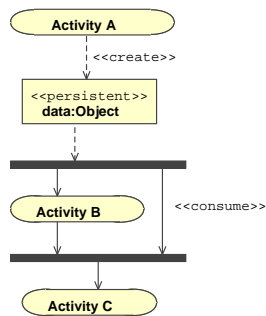


Fig. 14. Modeling of persistent data

action subsequent to a transition of the CONSUME stereotype.

In example fig. 14 the data object is still existing during the execution of the activity B. If the job model defined for B also debits the resource with memory consumption by the amount of the data object, the size is counted twice. The modeler has to think about these effects in detail.

## V. FUTURE WORK

This paper is a first step in developing a workbench for performance modeling. Prototypes of the introduced software tools have been developed. These are to be extended or improved in several directions.

1) *Enhanced monitoring tool*: The monitoring tool can only be used by an expert. Special knowledge in hardware resources, platforms, wrapper technology, etc. is necessary. The modeler would desire a universal tool with graphical user interface.

2) *Physical models*: All actual resource models comply to simple physical and mathematical models. They have to become a lot more sophisticated to become more applicable and more easy to use.

3) *Models with States*: Resources in fact do have states. They can become inactive through former activities and may spontaneously invoke activities of the software. We do not have these types of resources yet. Therefore models have to be developed that introduce own activity models, when using them in diagrams.

4) *Activity Models*: We have to develop tool support for the extension of the activity package, analogous to the monitoring tool and the model generator.

5) *Methodology*: This frame work should be applicable in a big software development project. The development process should be controlled by the assessments of these tools. This should support the approach to the performance goals and lead to a suitable enhanced methodology.

## VI. CONCLUSION

We introduced a new method to model dynamical properties of software and abstract environments. Furthermore, we introduced a configurable environment simulator to assess the performance properties of the software model in various phases of a project. Several packages provide useful elements for performance modeling. Only few extensions of the UML had to be made. Performance properties can now be modeled in a conventional way. That should result in a high acceptance of this techniques. The provided mechanisms of extension and inheritance do allow one to proceed in a top down design manner.

Certainly this simple way of constructing “mechanical” models of an environment does not achieve absolutely accurate assessment of response time and dynamical behavior. But easily creating specialized models of data, jobs and resources allows one to set the focus on important parts of a project and “zoom” in. So vital performance aspects can participate in controlling a software development process, particularly if we want to follow a top down approach. Desired properties or achieved results can be found in the performance analysis documents of the simulator.

Finally this platform is widely open for extension. All conceivable performance models can be integrated, if they can be covered by a class or be expressed in a UML diagram.

### *Acknowledgments*

Many thanks to Martin Wirsing, Alexander Knapp and Matthias Hölzl for a lot of conversations, for many cups of coffee and their almost infinite patience.

## REFERENCES

- [UML1.4] OMG, Unified Modeling Language Specification, September 2001, Version 1.4
- [bal01] Software Performance: state of the art and perspectives, S.Balsamo, A.DiMarco, P.Inverardi, M.Simeoni, Universita del Aquila/Univ.d.Venezia, Technical Report CS-2002-13, Jan. 2003
- [OMGRFP] Response to the OMG RFP for Schedulability, Performance, and Time, June 2001 docnr. ad/2001-06-14
- [bal02] On transforming UML models into performance models, S.Balsamo M. Simeoni, Univ.d. Venezia, WTUML, ETAPS 2001
- [wil01] PASA-A Method for the Performance Assessment of Software Architectures, L.Williams, C.Smith, Workshop on SW Performance, Rome, 2002
- [Amm01] Modeling resources in a UML-based simulative environment, H.Ammar/V.Cortelessa/A.Ibrahim, Univ. West Virginia, ACS/IEEE, Beirut, Lebanon, 2001
- [bal03] Deriving Performance Models from Software Architecture Specifications, S.Balsamo/M.Simeoni, Univ. d. Venezia Italy, ESM, 2001
- [pet01] Software Performance Models from System Scenarios in Use Case Maps, D.Petriu/M.Woodside, Carleton Univ. Canada, Proc. Performance TOOLS 2002, London, 2002
- [wil02] PASA:An Architectural Approach to Fixing Software Performance Problems, L.Williams, C.Smith, Proc. CMG, Reno, 2002