A Model Transformation Tool for Performance Simulation of Complex UML Models

Olivier Constant
Verimag
Centre Equation, 2 av de Vignate
38610 Gières, France
olivier.constant@imag.fr

Wei Monin
France Télécom R&D
28 chemin du Vieux Chêne
38243 Meylan, France
wei.monin@orange-ftgroup.com

Susanne Graf
Verimag
Centre Equation, 2 av de Vignate
38610 Gières, France
susanne.graf@imag.fr

ABSTRACT
Telecommunication operators need to continuously deploy innovative, functionally complex services that are by nature sensitive to performance issues. Performance analysis of complex services is a costly activity that suffers from the decoupling between design and performance modeling. To help overcome this difficulty, we propose a tool that automatically transforms UML2 models with complex behavior to performance models for a commercial simulator. The tool is extendible to other front-end design languages and other back-end performance tools, thanks to an intermediate metamodel with clear operational semantics.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – *elicitation methods, languages, methodologies, tools.*

General Terms
Performance, Design, Languages.

Keywords
Performance simulation, UML, Model transformation.

1. INTRODUCTION
The ever growing weight of software in the telecommunication industry is illustrated by the strong competition of telecommunication operators for the provision of innovative services which become increasingly functionally complex.

These services are by nature sensitive to performance issues for at least two reasons. First, they are implemented on top of distributed, heterogeneous deployment infrastructures. Processing nodes and network links are resources that can represent bottlenecks for the overall system's performance. And second, the services are subject to an uncontrolled number of concurrent requests from clients. Request arrival patterns, that can be arbitrary, have a huge influence on the system's overall performance.

Performance analysis is thus a key factor to the success of the design and development of these systems. Performance analysis of complex systems can be achieved by simulation. It consists in executing a performance model of the system, generally expressed in a specialized language based on Queueing Networks or Petri nets. At the early stages of design, performance simulation provides end-to-end response time and throughput estimations that allow to detect problematic architectural decisions [14]. At later stages, it allows to determine an acceptable tradeoff between performance and resource cost.

Nonetheless, performance simulation is rarely applied in real-life projects. Building a performance model of a complex system is a costly task that requires expertise. Moreover, "functional" design modeling and performance modeling differ in terms of terminology, languages and tools. When design and performance models of the same system are created, there is usually no guarantee that they are consistent with each other.

To reduce the gap between the roles of system designer and performance expert, a solution consists in enriching design models with performance information before transforming them to a performance model. We propose a model transformation tool that performs automatic translation from UML2 design models to models for a commercial performance simulator, HyPerformix Workbench [7]. The main strength of our tool, which is a result of the Persiform project [12], is its support for complex behaviors.

The tool helps reduce the cost of producing a performance model, and provides guarantees of consistency between a design model and its corresponding performance model. The tool is extendible to other front-end design languages and other back-end performance tools, thanks to a dedicated intermediate metamodel with clear operational semantics.

This article is structured as follows. The originality of the tool regarding related work is stressed in Section 2. How the tool is used and what features it supports are presented in Section 3. Section 4 gives an overview of the tool's design principles and finally Section 5 describes how the tool was validated against case studies.

2. RELATED WORK
A number of research works, e.g. surveyed in [2], have investigated approaches where early design models are transformed into performance analysis models. More recent works tend to apply Model-Driven Engineering (MDE) technologies for the same purpose [1, 6, 9, 13, 15, 16].

Most of these approaches share a common focus on the early stages of design and the application of analytical methods rather than simulation. The behavioral aspects of models are therefore simpler than what simulation allows. In addition, to our knowledge, no approach connects to a commercial simulator. The quality of the analysis tool, however, is a factor for the effectiveness of performance analysis.

Also, few of these approaches mention the issue of the systematic treatment of models and subsequent semantic difficulties. Formal semantics is rarely invoked, with the notable exception of [9].

Some of the aforementioned works are based on OMG profiles for UML, in particular the SPT profile [11]. Future works relying on the new MARTE profile [10] are expected. Our tool is also based on a UML profile, but it is independent of MARTE or SPT. While these OMG profiles provide a generic syntax based on a unified domain model, transforming models for simulation requires precise operational semantics, which can only be given to a restricted, focused syntax. Nonetheless, our profile could profitably be based on a subset of MARTE. It is not because MARTE was not available early enough for the Persiform project.

3. TOOL USAGE

From the user's perspective, the front-end side of our tool is a UML2 profile for the Eclipse-based Rational Software Modeler (RSM) design tool.

This profile pursues two objectives: (1) constrain the system designer to use UML in a way that is consistent with the methodology, and (2) allow the performance expert to complete the UML design model with performance information. This information can be edited via the design tool's GUI.

The profile focuses on Use Case Diagrams, Deployment Diagrams and Activity Diagrams (Figure 1). All other UML diagrams are allowed, but they are ignored by the transformation tool.

3.1 Structure Design

Use Case Diagrams allow the designer to identify the main services provided by the system and pinpoint related actors, which are classes of clients, in the environment. The performance expert may associate request arrival patterns with actors.

Deployment diagrams are a means to specify the allocation of software units (artifacts) onto the system's deployment infrastructure. The performance expert may enhance the definition of the infrastructure with resource information. A deployment node may be associated with resources such as CPUs, memories, network links or semaphores with specific characteristics.

3.2 Behavior Design

The critical point is the support for behavior. For simulation purposes models need to be executable, that is, they must have unambiguous operational (behavioral) semantics. To achieve this in UML, the syntax of UML behaviors must be restricted to an appropriate, consistent subset. This stands even if UML models are not executed directly but rather transformed to a simulation language.

A major strength of our tool is its full support for a significant subset of UML Activity Diagrams. It leaves the designer with the ability to specify models that are behaviorally complex yet elegant, while guaranteeing their adequacy to performance simulation. This subset is further extended to tackle semantic variation points and performance modeling concerns.

Activities in our profile describe the behavior of use cases, that is the overall processing of a client request by the system. An activity is a reusable, encapsulated graph of actions and control nodes. It may own datatyped variables and in/out parameters. An activity may (synchronously) call other activities by means of particular actions with parameter passing.

A control node can be a fork/join, a probabilistic or deterministic decision point, an initial node, or a final node of a certain kind. An action may compute data, block until some condition becomes true, or consume time. The details of actions are specified in C code, possibly involving predefined probability distribution functions.

In addition, actions may represent resource consumptions that are specified by the performance expert: CPU time consumption, memory allocation/release, semaphore take/release, data sending through a network link. A resource consumption is associated to a software unit, which links to actual resources via the deployment relationship.

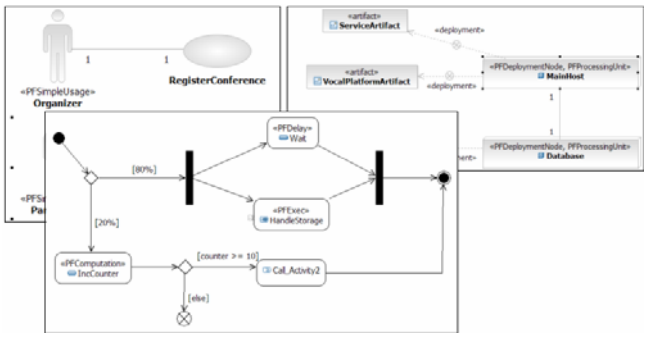


Figure 1. A simple UML model, consistent with our profile

3.2.1 Treatment of semantic ambiguities

The selected UML subset includes a number of semantically ambiguous elements. In some cases, it is possible to impose a precise semantics because it suits most designers. In other cases, we have to give the designer the flexibility to use a default semantics or choose an alternative one.

For this purpose, the profile defines some syntactic extensions for advanced designers. These extensions mostly relate to concurrency. An activity may have several concurrent "executions" due to multiple calls, and executions may in turn contain several concurrent "threads" according to fork/join nodes. Hence the following choices and extensions:

- Variables (attributes) of an activity are by default shared by all threads of all executions of the activity (they represent the system state). The designer may alternatively tag a variable as kind "flow", meaning that every thread has its own copy of the variable. This is useful for local data manipulation such as e.g. loop iterators.
- Join nodes are hard to interpret unless they occur in simple fork/join patterns in loop-free activities. To cover all usage cases, two kinds of synchronization and a notion of causality thread have been added to the syntax.
- To support the joint use of join nodes and flow variables, a syntax extension allows the designer to specify the value of flow variables of the thread leaving the node (Figure 2).
- UML defines two kinds of final nodes with different semantics. Two additional kinds have been added for a finer support of concurrency [3].

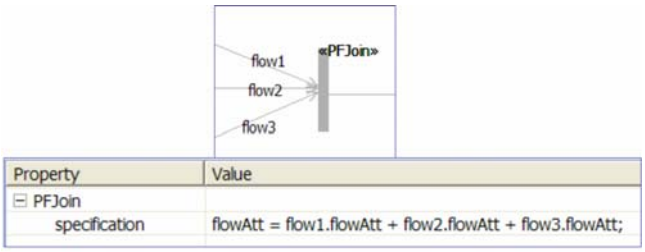


Figure 2. Specifying a flow variable (*flowAtt*) in a join node

This systematic treatment of semantic ambiguities is a prerequisite for guaranteeing the transformability of all consistent models.

3.3 Consistency Check and Transformation

The user can check that a model is syntactically consistent with our methodology at any time. Consistency is defined in the profile as a set of OCL constraints (rules of static semantics). These constraints can be tested against the model from within the design tool via its validation facilities. All constraint violations are notified.

A consistent model can be automatically transformed via the design tool's GUI by a transformation based on the ATL model transformation engine [8]. The output is a file that can be opened, browsed and exploited with the HyPerformix Workbench simulator.

4. TOOL DESIGN

The tool architecture can be described at two different levels of abstraction: concept transformation at a high level, then transformation artifacts at a technical level.

4.1 Architecture: Conceptual Level

Our tool uses an intermediate metamodel, which is called *PF*, to enhance maintainability and extendibility (Figure 3). The goal of *PF* is to make explicit all executable concepts and to define their operational semantics in terms of a formalism with a well-understood semantics.

For that purpose, *PF* is built upon concepts as they exist in two well-known frameworks: Petri nets and Queueing Networks. Petri nets are appropriate for expressing concurrency and synchronization, while QNs represent resource consumptions in a high-level manner. In PF, resource consumptions occur when certain Petri net transitions are fired. In addition, PF encompasses time, data, probabilities and parameterized modules (Figure 4).

The first step of the transformation process (*UML2PF*) consists in giving UML models a clear operational semantics in terms of a PF model. The second step (*PF2WB*) consists in translating PF models into a Workbench model that has an equivalent semantics. PF concepts that do not exist *per se* in Workbench are translated to C code.

Thanks to the clear operational semantics of PF, our tool can be extended to other front-end design languages, and to other back-end performance simulation languages of equivalent expressive power. A successful experiment has been carried out in the Persiform project with the introduction of annotated Message Sequence Charts as another front-end [4, 12].

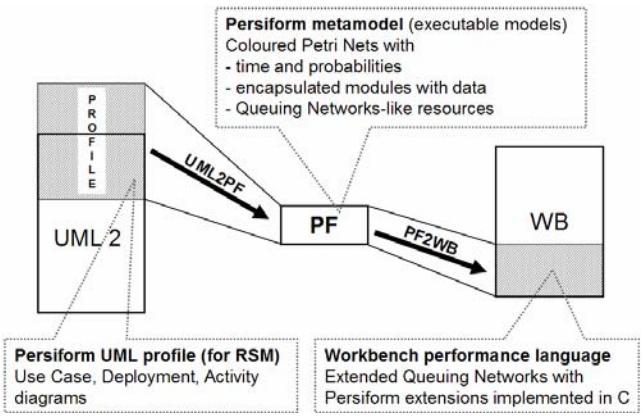


Figure 3. Conceptual view of the transformation process

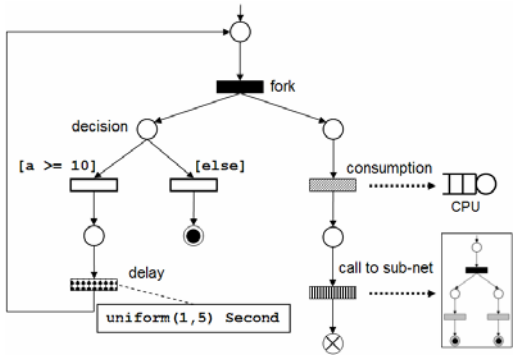


Figure 4. Example of a PF model (arbitrary representation)

4.2 Architecture: Technical Level

For the purpose of generating a concrete performance simulation file, more is needed than transformation of concepts (Figure 5). Two last transformation steps take care of generating: (1) a file that conforms to the simulator's XML format, and (2) a graphical layout so that the simulator is able to display a model which is readable by the performance expert. The last step makes use of the Graphviz tool [5].

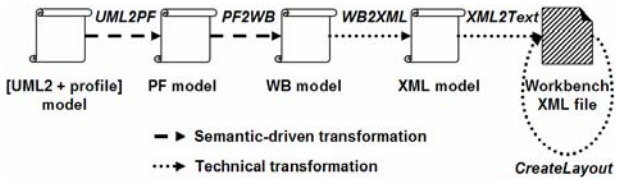


Figure 5. Technical view of the transformation process

Almost all transformation steps have been implemented in ATL [8] in a fully declarative style. This results in high-level, readable code whose maintainability and extendibility have already been tested with success.

5. TOOL EVALUATION

The tool has been experimented on three case studies provided by industrial partners of the Persiform project. The first one is a DSL registration system. Its non-trivial size (17 Activity Diagrams) and complex behavioral patterns allowed testing the tool's robustness.

The second case study is a military command and information system. It enabled us to cover a significant subset of the concepts supported by the tool. Performance simulation showed that the design of the system as it was initially modeled was inappropriate; indeed, the management of user sessions was too permissive (Figure 6).

The third case study, a service integration platform, allowed validating the semantics. Since there existed a legacy Workbench model of the system built "from scratch", its simulation results were compared to results obtained from a UML model of the system via our tool. Both outputs turned out to be equivalent, as expected. Of course, additional experiments would increase confidence.

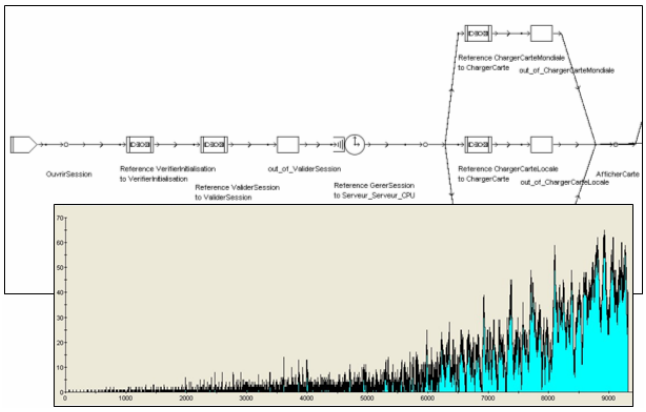


Figure 6. A Workbench model produced by the tool and corresponding performance results showing an explosion of the number of pending CPU consumption requests.

6. CONCLUSION

We have presented a tool that is able to transform complex UML models of service-oriented systems to exploitable performance simulation models. The tool takes advantage of the capabilities of performance simulation thanks to a rich intermediate metamodel with a clear operational semantics. We have tested that the combined use of a modular architecture and high-level model-driven engineering technologies give the tool an interesting level of maintainability and extendibility.

Current limitations are related to the consistency checking of UML models. Although a set of consistency constraints has been defined in the profile, it is difficult to define an exhaustive one. Also, the tool does not parse and check textual expressions of UML models (C conditions and actions on variables). They are kept as strings throughout the transformation process and only checked and compiled by Workbench.

Perspectives include the redefinition of the profile with a relevant subset of the MARTE profile. This will allow users to access a standard syntax for performance information. In terms of application domains, the tool is planned to be extended in order to be applied on avionic systems.

7. REFERENCES

[1] D'Ambroglio, A. 2005 A Model Transformation Framework for the Automated Building of Performance Models from UML Models. Proc. 5th ACM Workshop on Software and Performance (Palma, Illes Balears, Spain, July 12 - 14, 2005).

[2] Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M. 2004 Model-based performance prediction in software development: a survey. IEEE Trans. on Software Eng., vol.30 (5), May 2004, pp. 295-310.

[3] Bozga, M., Combes, P., Graf, S., Monin, W., Moteau, N. 2006 Qualification d'architectures fonctionnelles. Proc. NOTERE Conf. (Toulouse, France, June 2006).

[4] Constant, O., Hélouët, L., Jard, C. 2006 Traduction des Diagrammes d'Activités et des Message Sequence Charts vers le formalisme intermédiaire des Réseaux de Petri Colorés Stochastiques. Persiform Deliverable D3.1.

[5] Graphviz web page. http://www.graphviz.org

[6] Grassi, V., Mirandola, R., Sabetta, A. 2007 A model-driven approach to performability analysis of dynamically reconfigurable component-based systems. Proc. 6th ACM Workshop on Software and Performance (Buenos Aires, Argentina, February 5-8, 2007).

[7] HyPerformix web page. http://www.hyperformix.com

[8] Jouault, F., Kurtev, I. 2005 Transforming Models with ATL. Proc. Model Transformations in Practice Workshop at MoDELS (Montego Bay, Jamaica, 2005).

[9] Lopez-Grao, J.P., Merseguer, J., Campos, J. 2004 From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. Proc. 4th ACM Workshop on Software and Performance.

[10] Object Management Group. 2007 A UML Profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems), Beta 1. Document ptc/07-08-04.

[11] Object Management Group. 2005 UML Profile for Schedulability, Performance, and Time Specification v.1.1. Document formal/05-01-02.

[12] Persiform: French RNRT national project. http://www-persiform.imag.fr/

[13] Petriu, D.C., Woodside, C.M., Petriu, D.B., Xu, J., Israr, T., Georg, G., France, R., Bieman, J., Houmb, S.H., Jürjens, J. 2007 Performance Analysis of Security Aspects in UML Models. Proc. 6th ACM Workshop on Software and Performance (Buenos Aires, Argentina, February 5-8, 2007).

[14] Smith, C.U., Williams, L. 2001 Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley.

[15] Woodside, M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J. 2005 Performance by unified model analysis (PUMA). Proc. 5th ACM Workshop on Software and Performance (Palma, Illes Balears, Spain, July 12-14, 2005).

[16] Zhu, L., Liu, Y., Bui, N.B., Gorton, I. 2007 Revel8or: Model Driven Capacity Planning Tool Suite. Proc. 29th Intl. Conference on Software Engineering (Minneapolis, USA, May 20-26, 2007).

8. APPENDIX A: DEMONSTRATION

The demonstration will be organized as follows.

8.1 Introduction *(Slides and Demo)*

8.1.1 Context: Service-oriented systems (Slide)

What are the characteristics of these systems and why performance is an issue.

8.1.2 Existing techniques for performance evaluation (Slide)

Complementary techniques with their own advantages and drawbacks: analytical methods, simulation, prototyping.

8.1.3 Modeling with HyPerformix Workbench (Demo)

Main concerns in performance modeling: resource contention, causal flows, time consumption, resource modeling, environmental conditions. Tradeoff: realism of the model vs. simulation cost.

Expected outputs and how to use them.

8.1.4 Relationship Between Design and Performance Modeling (Slide)

Common concerns: causal flows (behaviors), functional delays, synchronizations, deployment.

Differences:

- Performance-specific concerns (environment, resource consumptions, time-based abstractions).
- Not all levels of abstraction in design models are relevant w.r.t. performance modeling.

8.2 Tool overview *(Demo)*

What the tool does.

How it integrates with Rational Software Modeler on the Eclipse platform.

Its architecture and its design rationale.

8.3 Case Study: The Military Command and Information (MCI) System *(Demo)*

8.3.1 Informal presentation of the system

Who is the client, what are the main functional and non-functional requirements.

8.3.2 Specification: Use Case Diagram

How and why Use Case Diagrams are used. How the performance expert completes the model.

8.3.3 Specification: Deployment Diagram

Similar to Use Case Diagram.

8.3.4 Specification: Activity Diagrams

Decomposing the behavior into activities.

Identifying and designing reusable activities.

Specifying the content of activities. Concepts used:

- Sub-activity calls with parameter passing.
- “Shared” and “flow” variables, data computations.
- Deterministic and probabilistic choices.
- Forks and loops.
- Specific final nodes.
- Delays and resource consumptions: CPU time and memory.

8.3.5 Tool application, simulation, results

Transformation of the UML model by our tool.

Viewing the generated model with Workbench.

Configuration of the model for simulation experiments.

Graphical animation.

Outputs obtained, conclusions on the modeled system.

8.4 Case Study: Traditional Performance Modeling vs. Model Transformation Approach *(Demo)*

8.4.1 Informal presentation of the IOS-W system

8.4.2 Comparison between the legacy “from scratch” performance model and the “UML-based” model

8.4.3 Checking semantic equivalence

Comparison of simulation outputs obtained from both models.

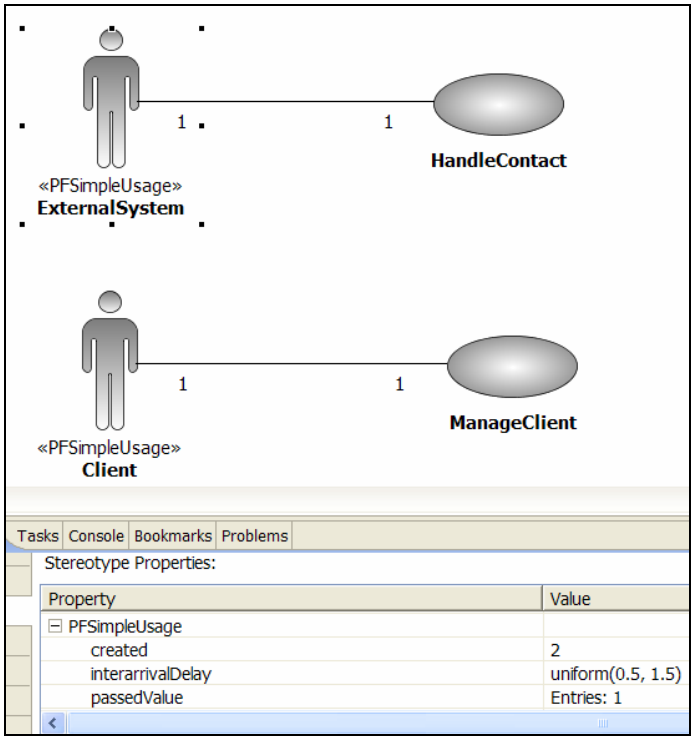
8.4.4 Conclusion

- Traditional “from scratch” approach: smaller model, clever use of the simulator’s modeling concepts, better efficiency. But the relationship with the design model is unclear.
- Model transformation approach using our tool: performance model is less elegant but semantically equivalent (same simulation results). This approach saves time and reduces costs, and guarantees that the performance model is consistent with the design model.

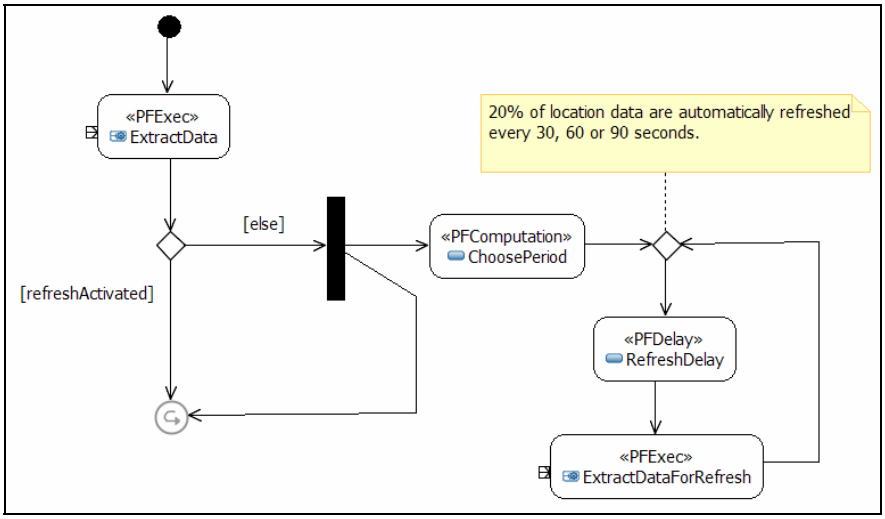
8.5 Advanced Features of the Tool *(Demo)*

- Distinguishing causal flows in activities.
- Weak and strong synchronization in join nodes.
- Specification of flow variables in join nodes.
- Combining final nodes in complex model configurations.
- Using priorities in resources.

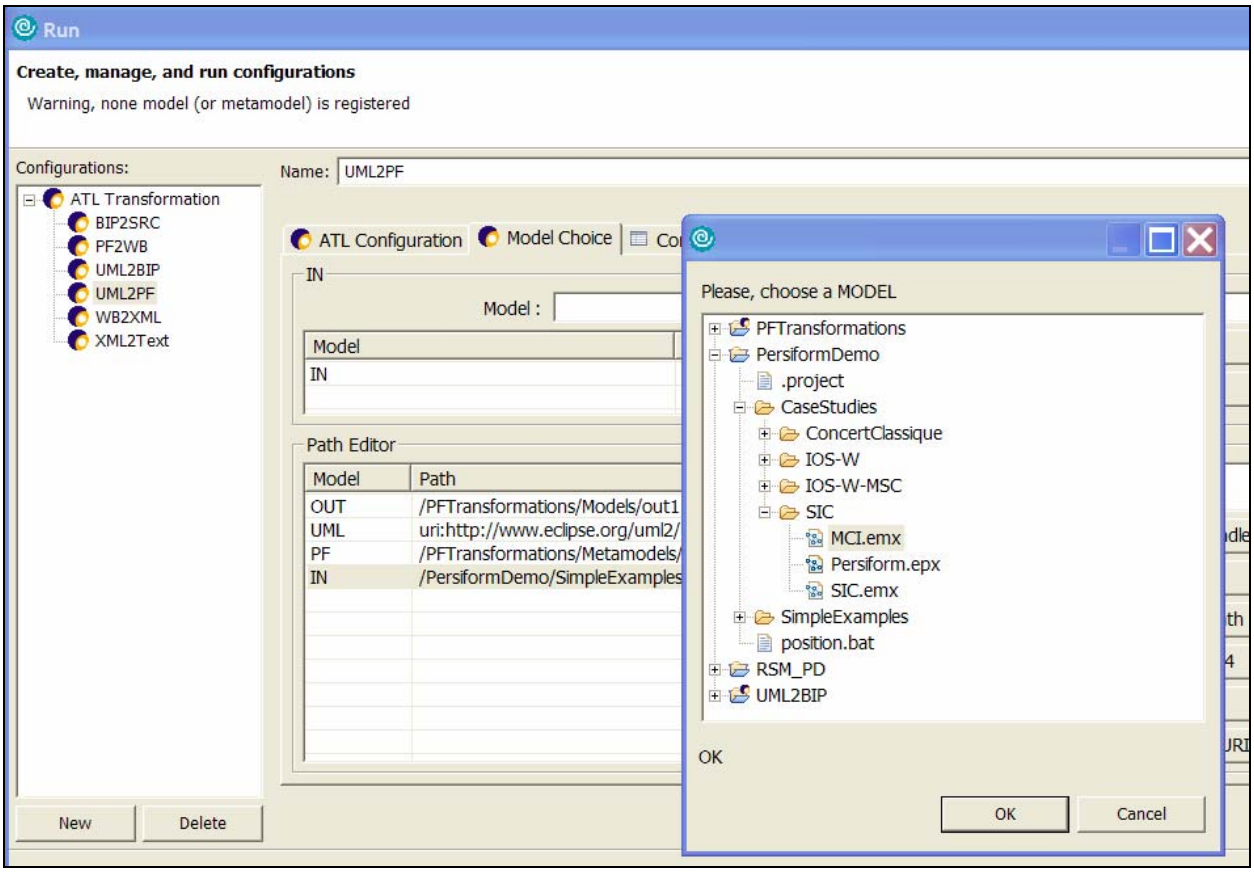
APPENDIX B: SCREEN DUMPS



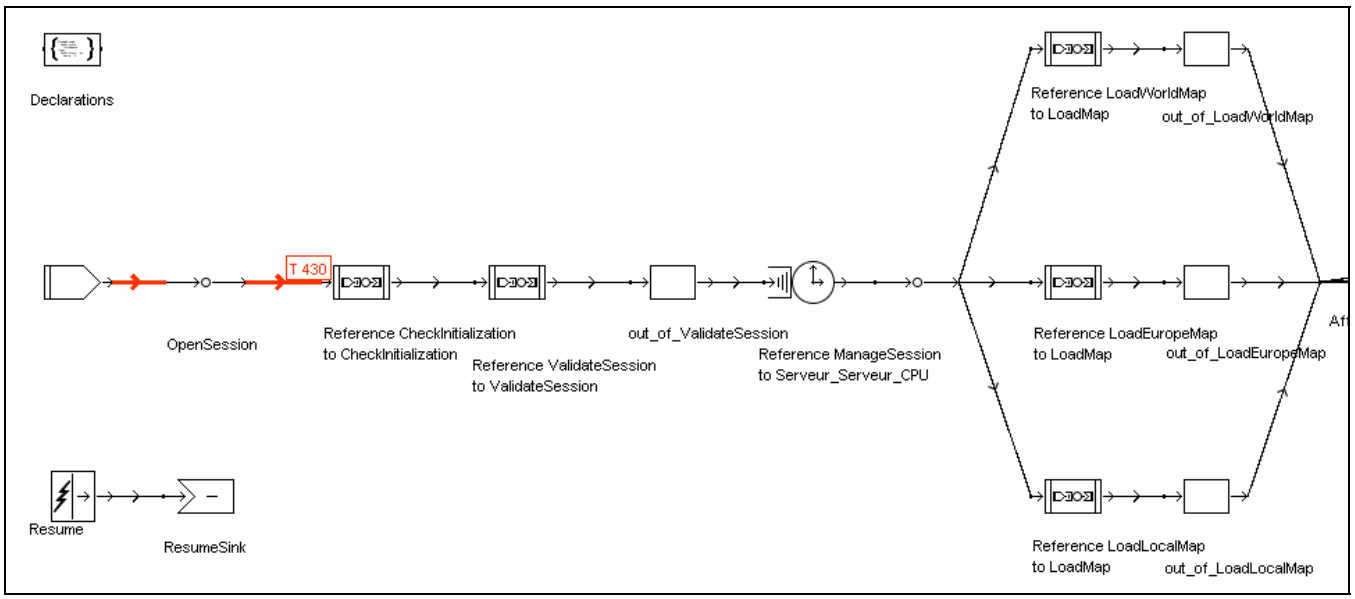
8.3.2: Use Case Diagram of the MCI System with our profile's performance extensions.



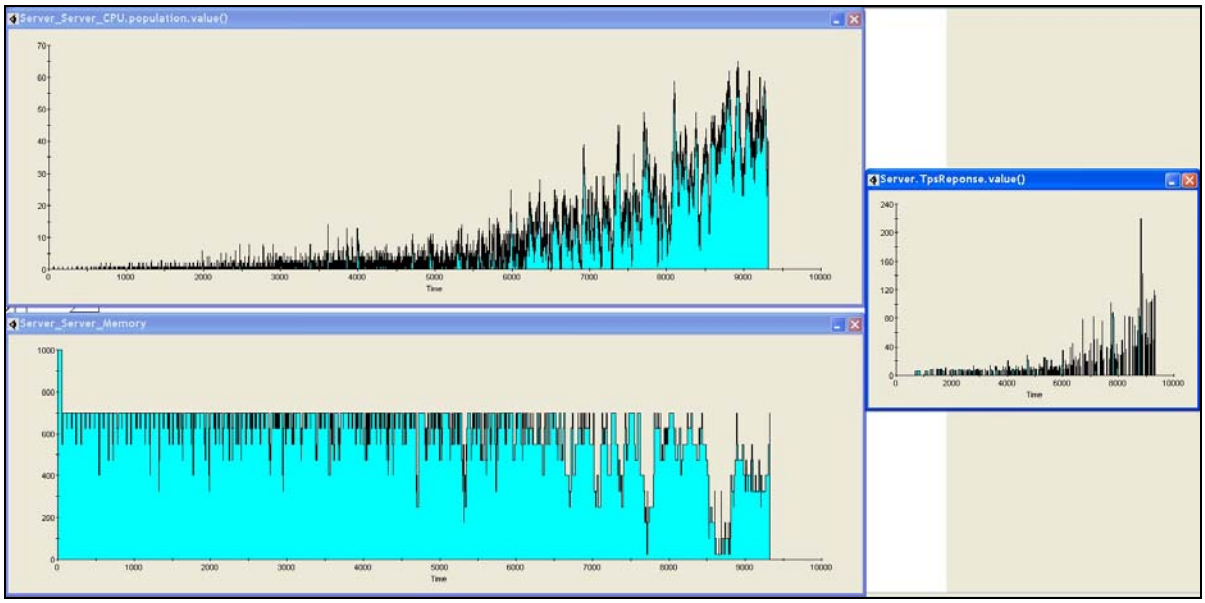
8.3.4: Activity Diagrams of the MCI System: the *HandlePeriodicExtraction* sub-activity.



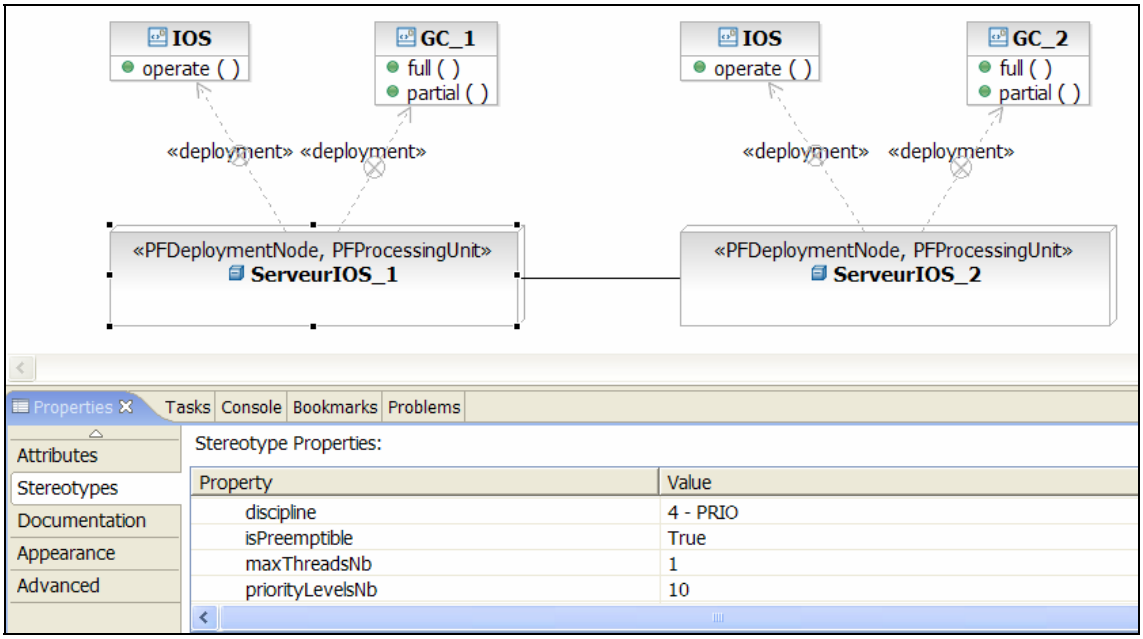
8.3.5: Running our tool on the MCI System model via the ATL GUI.



8.3.5: Visualizing and animating the generated Workbench model.



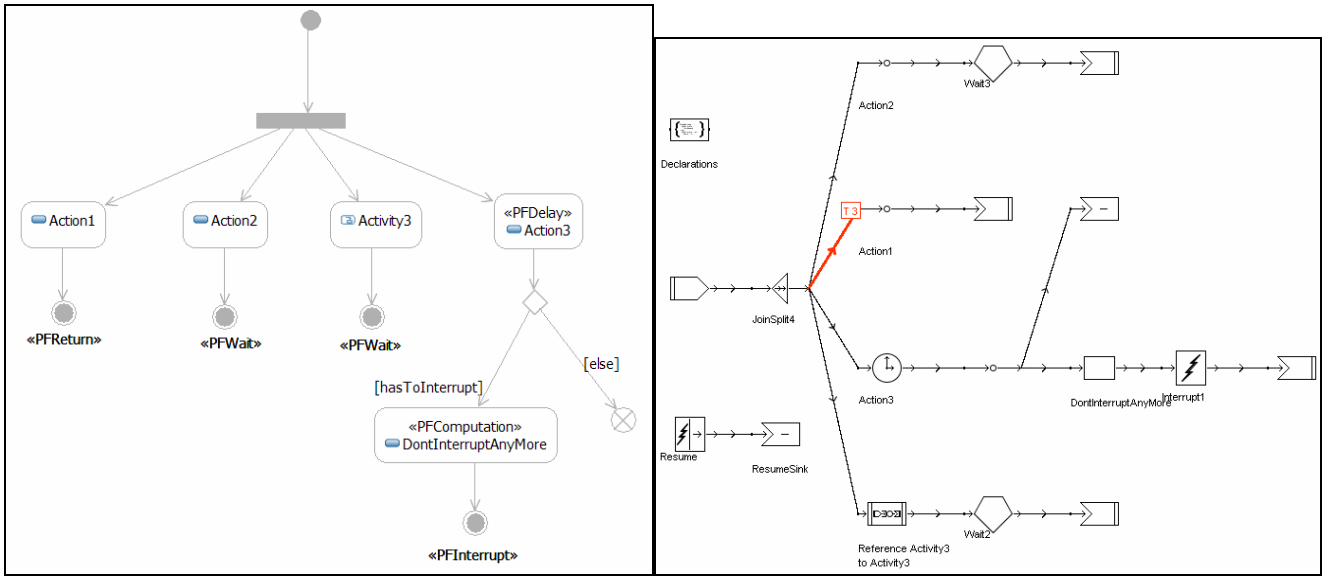
8.3.5: Measuring CPU load, memory usage and end-to-end response time during the simulation of the MCI System.



8.4.2: Deployment Diagram of the IOS-W system with performance extensions.

		(0;5]	(5;10]	(10;15]	(15;20]	(20;25]	(25;30]	(30;35]	(35
From scratch	**Nb requests**	0	7995960	3001730	202029	38919	24561	22713	
12500181	**In %**	0	63,966754	24,013492	1,6162086	0,3113475	0,1964852	0,1817014	0
From UML	**Nb requests**	0	7995960	3001730	202029	38919	24561	22713	
12500181	**In %**	0	63,966754	24,013492	1,6162086	0,3113475	0,1964852	0,1817014	0
Difference	**Absolute (%)**	0	0	0	0	0	0	0	
	Relative (%)		0	0	0	0	0	0	
		Min	Mean	Max	Count	Std. Dev.	CoV		
	From scratch	6,238	40,76	3657	12500181	135,3	3,32		
	From UML	6,238	40,76	3657	12500181	135,3	3,32		

8.4.3: Comparing simulation results of the traditional approach and the approach based on our tool.



8.5: Combining final nodes in a UML model and visualizing the execution of the generated Workbench model.