Putting Performance Engineering into Model-Driven Engineering: Model-Driven Performance Engineering

Mathias Fritzsche[1] and Jendrik Johannes[2]

[1] SAP Research CEC Belfast, BT370QB Belfast, United Kingdom
mathias.fritzsche@sap.com
[2] Technische Universität Dresden, D-01062, Dresden, Germany
jendrik.johannes@tu-dresden.de

Abstract. Late identification of performance problems can lead to significant additional development costs. Hence, it is necessary to address performance in several development phases by performing a performance engineering process. We show that Model-Driven Engineering (MDE) specifics can be utilised for performance engineering. Therefore, we propose a process combining MDE and performance engineering called Model-Driven Performance Engineering (MDPE).
Additionally we present our first experiences in application of MDPE concepts.

1 Introduction

Usability of software applications is highly dependent on how the resulting software system performs and if the resulting software fulfils performance related requirements. A software's performance, which is defined as "the degree to which an application or component meets its objectives for timelines" [1], is however seldom analysed before the actual system is implemented and performance of the running system can be measured.

Examples like the development of the Information System for the German Police called "Inpol-Neu" [2], which was published in the mass-media, show that addressing performance in a late phase of software development can lead to significant additional development effort. "Inpol-Neu" was planned to go productive in spring 2001. End of 2001, performance tests for the integrated system indicated that the system was not able to fulfil performance-related requirements. A reduced version of "Inpol-Neu" went live at 18th August 2003. The example of "Inpol-Neu" indicates the importance of addressing the performance of an integrated application throughout the whole development lifecycle.

Earlier performance analysis can be done through *performance engineering*, which is not well integrated into most software development processes. This paper investigates how to improve the integration of performance engineering into the development process of complex software applications. It argues that by moving from *Code-Centric Development* (CCD) to *Model-Driven Engineering*

(MDE) many of the causes for poor integration of performance engineering can be prevented. Based on this observation, it proposes an extension of MDE that we call *Model-Driven Performance Engineering* (MDPE).

The paper is structured as follows: Section 2 reviews performance engineering techniques for CCD and delineates extended possibilities of performance engineering by utilisation of MDE. Following these observations, we propose a *Model-Driven Performance Engineering* (MDPE) process in Section 3. Section 4 summarises first experiences of applying MDPE and introduces a tool architecture for future implementations. Section 5 compares our approach with related concepts and highlights its unique properties.

2 Background

This section provides a comparison of extending both *Code-Centric Development* (CCD) and *Model-Driven Engineering* (MDE) with performance engineering.

2.1 Extending Code-Centric Development with Performance Engineering

Different established performance analysis techniques like Queuing Networks [3], Layered Queuing Networks [4], Queuing Petri Nets [5], and FMC-QE [6] use formal performance models for performance predictions. Other approaches for evaluation are simulation-based [7], where models of a system are executed in a simulation.

It is evident that performance prediction at design time requires analysis models defining the structure and behaviour of the analysed system, resource requirements, branch probabilities, and details about *factors due to contention of resources*. Semi-formal models are commonly used in CCD, but mainly to support human understanding, communication, and documentation of a system. Therefore CCD requires models only to be sufficient to support human interpretation. They are not used as development artefacts for forward engineering from models to executable code.

In more detail, a semi-formal specification of models makes tool support in terms of transformations, consistency checks, and repository functionality difficult. Therefore, models in CCD are not kept in sync after the code is implemented or the system is refined and are thrown away. Analysis models for performance, design models, and code have to be developed separately using specific knowledge about performance modelling concepts like the Queuing Theory. Therefore these approaches are rarely used in industrial projects.

In terms of performance, a "fix it later approach" is commonly used for CCD as described in [1] and [8]. This can lead to significant correction costs as seen on the example of "Inpol-Neu". This is supported by our experience with SAP's development lifecycle called *Product Innovation Lifecycle* (PIL) [9]. SAP's PIL mainly addresses performance in early phases of requirement specification and

after integration of already implemented components. Performance predictions in intermediate stages of the development process are not performed.

We claim that this is because performance engineering solutions for CCD are not minimally intrusive in terms of missing automation and of missing formalisation: it is necessary to explicitly define models for performance engineering. This requires additional investment and performance engineering expertise that industry is not willing to provide. The next section argues that the same problems can be avoided by utilisation of MDE.

2.2 Extending Model-Driven Engineering with Performance Engineering

In MDE, a formalisation of modelling enables the usage of models as development artefacts in forward engineering from models to executable code. Modelling languages for *development models*[3] are formally defined with the help of meta modelling languages such as MOF [10]. This makes models machine-readable, which enables broad tool support as provided by the Eclipse Modelling Framework (EMF) [11] or SAP's Modelling Infrastructure MOIN [12].

Such tools offer automated transformation between models of different abstractions, between models specifying different view points, or from highly detailed models to platform specific code. Through this, different development artefacts can be kept in sync, saving effort and shifting effort from coding to modelling. By using transformations, performance models based on development models of different stages of refinement can be generated and performance engineering for each of these refinement stages becomes possible without much additional effort. This effort does not focus on modelling itself, but on annotating of separately defined performance information—like specification of resource requirements on already existing structural and behavioural models.

Based on these observations, the following section proposes an extended MDE process: *Model-Driven Performance Engineering* (MDPE).

3   Model-Driven Performance Engineering (MDPE)

The requirement of a short time to market in combination with an increasing complexity of software systems calls for a reasonable performance engineering process that utilise features of MDE as argued in Sect. 2. A solution that provides a high degree of automation is required to handle big and complex models of enterprise software. The flexibility in MDE wrt. different levels of abstraction and models of different domains, calls for a similar flexible solution for performance engineering. Earlier initial performance feedback with minimal effort should be supported as well as maximal performance feedback with extended (but still cost-efficient) effort. Such stepwise feedback should be given at any stage over the

---

[3] We use the term *development model* in this paper to distinguish between models as development artefacts and formal *performance models*.

whole development lifecycle. To meet these requirements we propose a solution which is based on the *Software Performance Engineering* (SPE) [1] approach and utilises MDE specifics.

SPE integrates different steps of performance engineering repeatedly at different stages of the development process. The idea is to generate early performance feedback; starting at the stage of requirement analysis. SPE is a stepwise process that distinguishes two kinds of performance models, a Software Execution model and a System Execution model.

The first one is represented in an execution graph [1]. This graph is a flow graph extended with resource demands and branching probabilities. It enables analysis in the absence of delays because of factors due to contention for resources like multiple users. Results of the analysis are the mean, best-case, and worst-case response time. The System Execution model is based on the Software Execution model extended with details about contention for resources (like deployment information and multi-user scenarios). The model can be represented as a Queuing Network [1]. The output of the analysis of this more complex performance model is the prediction of the utilisation of resources and detailed a performance prediction such as identification of bottle necks. Additionally, SPE includes the specification of patterns and anti-patterns for software architectures to minimise poor architectural decisions from a performance perspective [1, 13].

Fig. 1 delineates our idea of MDPE which is based on SPE but utilises specifics of MDE. The proposed process is an extension of MDE which enables SPE-like performance engineering by utilisation of MDE concepts as described in Sect. 2. In comparison to related approaches utilising model driven techniques [14–17], our approach enables stepwise performance feedback which can be repeated at different stages of refinement in a MDE process.

In MDPE, performance analysis is performed at each refinement step of the MDE process. Each time, performance feedback can be produced stepwise. Thus, we take two orthogonal dimensions of refinement in MDPE into account: One dimension to refine the performance models, and another dimension to refine the development models in a traditional MDE process.

Initial automated performance feedback is based on structural information from development models only (cf. *Static Model Analysis* in Fig. 1). This feedback will include the identification of performance anti-patterns and identification of model-based performance metrics to enable comparability of models from the performance perspective. An example of a simple model-based performance metric is the number of actions in a UML activity diagram.

The next analysis steps are based on performance analysis models. These can be derived from development models by enriching them with additional performance related information which requires involvement of the user. Following the concept of SPE, we distinguish between two kinds of performance models:

The *Initial Performance Analysis Model* is derived from development models enriched with the resource demands of actions and probabilities of branches in behavioural models. The computation of this model is simple. Only resource demands under consideration of branch probabilities have to be added. This sim-
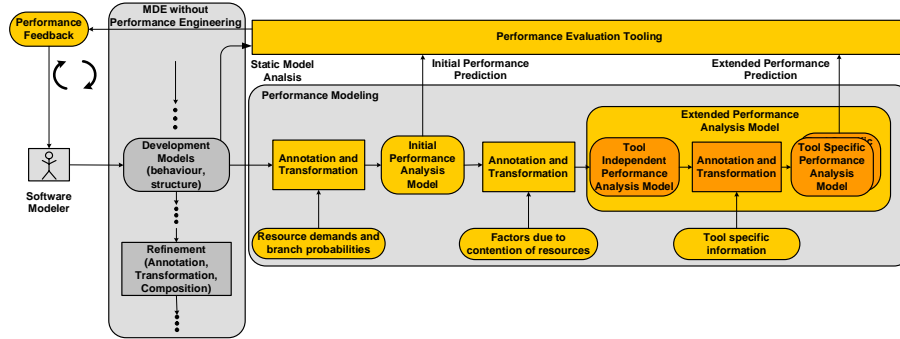
Fig. 1. MDPE concept as Block Diagram [18]

ple analysis enables *Initial Performance Prediction* which includes declaration of the mean, best-case, and worst-case response time. Initial Performance Analysis Models are conceptually the same as Software Execution Models in SPE.

The *Extended Performance Analysis Model* is comparable to a System Execution Model in SPE. It is a further enriched Initial Performance Analysis Model, annotated with information about contention for resources. The *Extended Performance Prediction* based on this model enables more precise results. Additionally, resource utilisation can be prognosticated. The price for these more detailed results is a more complex analysis.

Dependent on the size, characteristics and domains of the modelled system, different simulative or analytical evaluation techniques will be necessary to take factors due to contention of resources into account and to minimise possible failures in performance prediction. Therefore it is planned to evaluate performance based on Extended Performance Analysis Models by using different concepts and tools. For the implementation of this approach we require the definition of a *Tool-Independent Performance Analysis Model* like KLAPER [19]. This model will be the basis for several *Tool-Specific Performance Analysis Models* taking tool-specific information into account.

The next section describes our first experiences with using MDPE concepts to identify requirements for an implementation of the proposed process.

4 Initial Experiences and Future Work

To gain experience and to identify the challenges in MDPE, we attempt to directly derive a *Tool Specific Performance Analysis Model* from a *development model*. Since we target to build on standard (and, if possible, open-source) MDE tooling, we base our implementation on the Eclipse Modelling Framework (EMF) [11]. As a concrete Scenario, we choose an UML Activity Diagram as the *devel-*

opment model and an AnyLogic [20] simulation model as the *performance model*. Transformations are formulated in the ATL [21] model transformation language.

These choices have practical reasons: 1) UML is one of the most popular modelling languages and has good tool support for creating models. In the concrete example we were able to rely on EMF-based open-source tooling only (Eclipse UML2 Project [22] and TOPCASED editors [23]). 2) AnyLogic is a simulation tool which supports the creation of graphical representations of simulations and do on-the-fly analysis by simulation. This is suitable for first experimentations. 3) ATL was chosen, because it is the standard transformation language for EMF models.

First, we required means to enrich the UML diagram with performance information. To this end, we decided to use the UML profile for schedulability, performance, and time specification (SPT) [24]. In this profile, the stereotype `PStep` with a set of tagged values is defined. It can be annotated to a variety of UML elements. In our experimentation we only used the tagged values `HostExecutionDemand`, `Repetition`, and `Probability`.

Figure 2 displays an UML activity diagram where all actions have been annotated with a `HostExecutionDemand`—indicating their mean duration. The initial node was annotated with a `Repetition` value for the arrival rate. Each of the control flows from the decision node are annotated with a `Probability` value.

Second, we determined that the simplest analysis we could do is "running" the activity diagram in a simulation. In AnyLogic, we modelled a library of elements that correspond to activity diagram elements: We defined their semantics with respect to the simulation and gave graphical representations to make it easy for the developer to relate the simulation to the original diagram.

The AnyLogic library elements representing actions, decision, initial, and final nodes can be seen in Figure 3. The bar in front of an action represents the queue of unprocessed signals at the current simulation timestamp. A bar at a final node indicates the amount of arrived signals at the current simulation timestamp. An action that is currently active is filled.

Third, as the most challenging step, a transformation was written in ATL. It was expected that the transformation is straightforward, since we have made the required types of UML model elements available in AnyLogic through the provided library. However, it proved to be non-trivial to provide a complete transformation. The resulting transformation consists of 750 lines of ATL code, which are all hand-written. The implementation, together with example models and a detailed description of the example was released on the ATL Use Cases web page[4].

The simulation model from Figure 3 was created by the transformation without additional efforts. The figure shows the simulation running at a concrete timestamp (3108 time units have passed). A possible bottleneck of the system can be observed by monitoring the simulation: the queue in front of the `Success` action is growing but never shrinking.
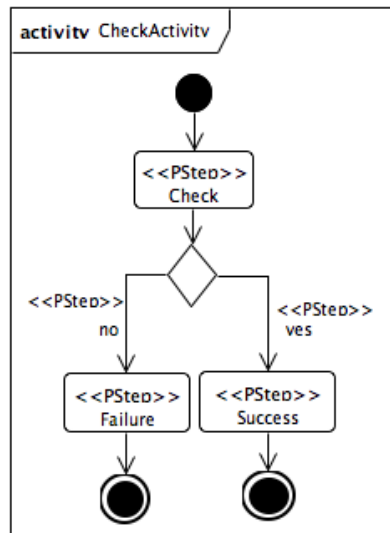
---

[4] http://www.eclipse.org/m2m/atl/usecases

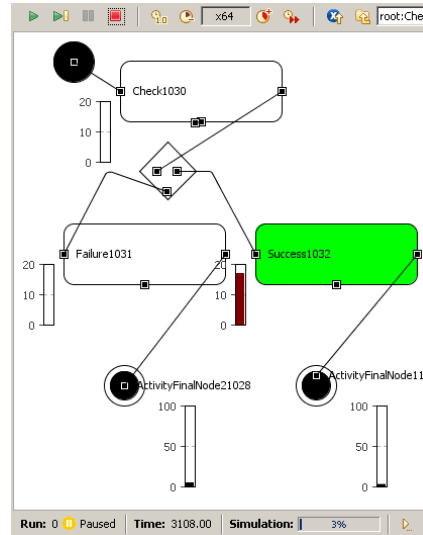**Fig. 2.** An UML activity diagram

Fig. 3. A derived AnyLogic transformation

In the following we discuss observations we made by creating this first implementation and their consequences for an MDPE tool architecture. This architecture is sketched in Fig. 4 and will be explained in the next sections.

4.1 Abstraction

One of the main benefits of MDPE is early performance prediction. That is, running analysis on abstract system definitions early in the development process. While such abstract models are machine-readable because they conform syntactically to their metamodel, they are not formal in the sense of having complete semantics. This raises conceptual and technical issues: On the conceptual side, we have to assure that we end up with a performance model that is not only syntactically correct, but also makes sense wrt. the analysis semantics. While some information gaps are filled with the additional performance information, these might not always be sufficient to run an analysis. Sometimes, analysis tools need additional parameters which can be derived by other means (e.g., setting them to a standard value, when it is considered harmless). Such information often cross-cuts the whole performance model. Practically, this is often the case in the transformation to AnyLogic. As a consequence, static information had to be repeatedly inserted at different points in the transformation. It would be desirable to separate this information into model fragments formulated in the analysis formalism and then weave them into the transformation result using Aspect-Oriented Modelling techniques (cf. lower right corner of Fig. 4).
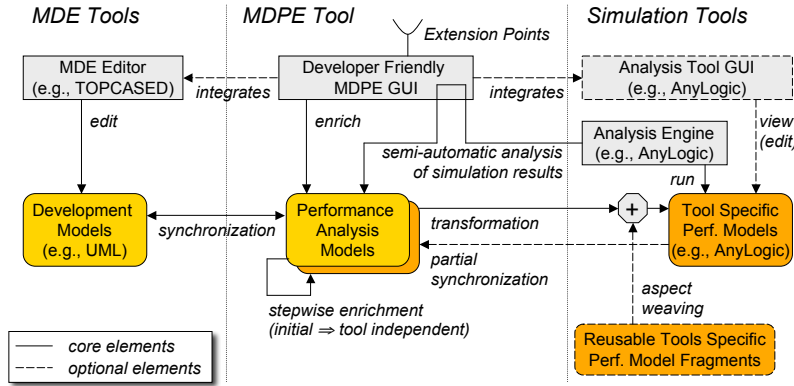
Fig. 4. MDPE Tool Architecture

4.2 Location and Definition of Performance Information

In the example we applied a profile to add performance information to the UML model directly. While that is a valid approach when UML is used to do performance modelling[5], the UML models handled are development models of our systems. Thus adding the performance information there pollutes the model with additional information that is of little value for the ongoing MDE design process.

Thus, this information should be added to the Initial and Extended Performance Analysis Models only (cf. Fig. 1). For these models we plan to develop a metamodel—not from scratch, but based on an existing work. Parts from UML together with the SPT profile [24] are a promising candidate to build on.

Since one of our aims is to let the software modeller do analysis without consulting performance engineering experts, the provision of performance information should be tool-supported and automated as much as possible. Consequently, we do not plan to provide a graphical editor for our performance models. They should rather be produced from development models and automatically acquired or user defined performance data. For the last case we anticipate providing tool-guidance for developers by providing wizards or similar means (cf. upper middle of Fig. 4). This user interface should offer extension points to integrate with performance data sources (e.g., collected performance data from experiences in a company). In some cases, it might be useful to integrate with MDE and analysis GUIs if possible, since it is an easy way to give better feedback to the user (cf. upper part of Fig. 4). For instance, observing an actual running simulation in AnyLogic and getting a visualisation of relations to the development model can leverage understanding of analysis results for non-performance experts.

---

[5] UML can be used for performance modelling by applying one of the standardised profiles SPT [24] or MARTE [25].

4.3 Structural Discrepancies

In the example, the analysis formalism was easily aligned with activity diagrams (by providing an AnyLogic library). Such correspondences between development and performance model elements might not always exist. It is not clear if this is a problem. Missing correspondences might also indicate that useful analysis can not be done in this combination. Nevertheless, we have to keep this in mind when designing a tool-independent performance metamodel, because it aims at combining arbitrary MDE formalisms with arbitrary performance engineering formalisms. It is clear that not all MDE models are useful for MDPE (e.g., static UML class diagrams without attached behaviour). But there is a subset of MDE formalisms which should be supported.

4.4 Transformation and Synchronisation

Our experience with providing a single ATL transformation has shown that the definition of the relationships between formalisms is not a trivial task and that it is difficult to ensure stable and complete transformations using this "single transformation" approach. Additionally, it is often required to not only transform, but update already transformed models, which is very important when refining models based on analysis results. Thus, a synchronisation mechanism is needed. In particular between development and performance analysis models, since structural changes, caused by analysis results, should reflect automatically on the design (cf. lower left corner of Fig. 4). Synchronisation between the tool-independent and tool-dependant performance models is also useful when the model structure can be changed directly in the analysis tool, which can be done, for instance, in AnyLogic (cf. lower right corner of Fig. 4). However, the analysis engines can also be hidden. The results they deliver have to be analysed, for instance by anti-pattern matching, to propose structural changes to the modeller (cf. upper middle of Fig. 4). Another related issue is versioning of models. It is required to run subsequent analysis on altered models, compare them, and choose the best design.

Some issues could be addressed by using additional features of ATL like several input and output models and high-level transformations. The ATL based tool Atlas Model Weaver (AMW) [26] could help with the synchronisation issue, since it allows the definition of relationships between models and the automatic generation of ATL transformation for both directions. Application of the Epsilon Platform [27] that supports several model-managing tasks is also considered. Another promising approach is Triple Graph Grammars [28] which offers solutions to incremental model synchronisation. We are in contact with researchers and developers of all the mentioned approaches. Some of us are involved in the development of the Reuseware Composition Framework [29, 30] which is momentarily extended for aspect-oriented modelling and could also be applied for tasks in MDPE. It has yet to be decided which transformation and synchronisation technologies should be used. MDPE is definitely a good application and use-case for several of these technologies. Thus, our work will also contribute to this area.

5 Related work

Current ideas in combining MDE and performance engineering are addressing different aspects. Various authors, e.g. [14, 15] propose use of transformations for generating performance models from annotated UML models in an automatic way. For instance [17], depicts how to map definitions of composite processes specified in Business Process Execution Language (BPEL) [31].

In [19] a language, called KLAPER, for performance models is proposed which is independent of the performance analysis technique like Queuing Petri Nets or Layered Queuing Models. Such a language simplifies performance evaluation of the same development models with the help of different performance analysis techniques.

In [16] a process that combines Model-Driven techniques and performance engineering is introduced which is called Software Performance Model-Driven Architecture (SPMDA). The SPMDA approach enables two kinds of performance models which are transformed from two fixed refinement steps into a Model-Driven Process. According to this, SPMDA does not enable stepwise performance engineering in terms of providing stepwise performance feedback for each refinement step in the development process.

To enable a minimally intrusive solution, it is not only necessary to automate generation of performance models, it is also important that a modeller can decide how much effort he invests in performance engineering. Therefore, we require stepwise performance feedback for each refinement step in a development process which includes earlier initial (maximal) performance feedback with minimal (extended) effort as shown for the Code Centric performance engineering approach.

The MDPE approach is based on SPE concepts. As in SPE the approach is stepwise by combining identification of anti-patterns for performance and different kinds of performance models. The approach is also usable for different stages in a development process to provide performance feedback throughout the whole development lifecycle. In comparison to SPE a high degree of automation can be reached because MDPE is based on formal development models with extensive tool support. This allows, for instance, automated generation of different *Tool Specific Performance Analysis Models* to minimise failures in performance prediction.

6 Conclusion

In this paper we proposed to extend Mode-Driven Engineering to Model-Driven Performance Engineering. This approach utilises Mode-Driven techniques to enable stepwise performance feedback. This includes earlier initial (maximal) performance feedback with minimal (extended) effort. The approach is repeatable at different stages of refinement in a Mode-Driven Engineering process. MDPE is applicable if behaviour models of the system are available on abstraction levels where performance data can be defined based on experiences or measurements on

existing systems. As a consequence, the possibility of performing early analysis could increase the acceptance of behavioural modelling in a MDE process.

We clarified that putting this process into industrial practices requires tools with high automation. Our first experiments identified requirements for such a tool architecture which we will continue to evolve and improve. For instance, a synchronisation mechanism between different development and analysis models is required to adequately integrate analysis results into the development process. To define transformations to analysis models, the semantics of the development models have to be clearer than they often are in traditional MDE processes.

Acknowledgement

References

1. Smith, C.U.: Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software. Addison-Wesley (2002)
2. Jakob Vo, J.R., Tretkowski, I.: Das polizeiliche Informationssystem INPOL (2002)
3. G. Bolch, S. Greiner, H.d.M., Trivedi, K.: Queuing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. John Wiley & Sons Inc., New York, USA (1998)
4. Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S.: The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. IEEE Transactions on Computers **44**(1) (1995) 20–34
5. Bause, F.: "qn + pn = qpn" - combining queueing networks and petri nets (1993)
6. Zorn, W.: Fmc-qe: A new approach in quantitative modeling. In: Proc. of MSV'07. (June 2007)
7. Robinson, S.: Simulation : The Practice of Model Development and Use. John Wiley & Sons Inc., New York, USA (March 2004)
8. Dugan, R.F.: Performance lies my professor told me: the case for teaching software performance engineering to undergraduates. SIGSOFT Softw. Eng. Notes **29**(1) (2004) 37–48
9. SAP AG: SAP's product innovation lifecycle
10. OMG: MetaObject Facility (MOF) specification version 2.0 (January 2006) URL http://www.omg.org/cgi-bin/doc?formal/2006-01-01.
11. Budinsky, F., Brodsky, S.A., Merks, E.: Eclipse Modeling Framework. Pearson Education (2003)
12. Michael Altenhofen, T.H., Kusterer, S.: Ocl support in an industrial environment (2006)
13. Smith, C.U., Williams, L.G.: New software performance antipatterns: More ways to shoot yourself in the foot. In: Int. CMG Conference, Computer Measurement Group (2002) 667–674

14. D'Ambrogio, A.: A model transformation framework for the automated building of performance models from uml models. In: WOSP '05: Proceedings of the 5th international workshop on Software and performance, New York, NY, USA, ACM Press (2005) 75–86

15. Gu, G.P., Petriu, D.C.: Xslt transformation from uml models to lqn performance models. In: WOSP '02: Proceedings of the 3rd international workshop on Software and performance, New York, NY, USA, ACM Press (2002) 227–234

16. Cortellessa, V., Marco, A.D., Inverardi, P.: Software performance model-driven architecture. In: SAC '06: Proceedings of the 2006 ACM symposium on Applied computing, New York, NY, USA, ACM Press (2006) 1218–1223

17. D'Ambrogio, A., Bocciarelli, P.: A model-driven approach to describe and predict the performance of composite services. In: WOSP '07: Proceedings of the 6th international workshop on Software and performance, New York, NY, USA, ACM Press (2007) 78–89

18. Knopfel, A., Grone, B., Tabeling, P.: Fundamental Modeling Concepts: Effective Communication of IT Systems. John Wiley & Sons (2006)

19. Grassi, V., Mirandola, R., Sabetta, A.: From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In: WOSP '05: Proceedings of the 5th international workshop on Software and performance, New York, NY, USA, ACM Press (2005) 25–36

20. XJ Technologies: AnyLogic — multi-paradigm simulation software (June 2007) URL http://www.xjtek.com/anylogic/.

21. ATLAS Group: ATLAS transformation language (June 2007) URL http://www.eclipse.org/m2m/atl/.

22. The Eclipse Foundation: Eclipse UML2 (June 2007) URL http://www.eclipse.org/uml2/.

23. The Topcased Project Team: TOPCASED (June 2007) URL http://www.topcased.org.

24. OMG: UML profile for schedulability, performance, and time specification (January 2005) URL http://www.omg.org/docs/formal/03-09-01.pdf.

25. OMG: UML profile for modeling and analysis of real-time and embedded systems (2007)

26. The AMW Project Team: Atlas Model Weaver (June 2007) URL http://eclipse.org/gmt/amw/.

27. The Epsilon Project Team: Epsilon Platform (June 2007) URL http://eclipse.org/gmt/epsilon/.

28. Schürr, A.: Specification of graph translators with triple graph grammars. In: WG '94: Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science, London, UK, Springer-Verlag (1995) 151–163

29. Software Technology Group, Technische Universität Dresden: Reuseware Composition Framework (June 2007) URL http://www.reuseware.org.

30. Henriksson, J., Johannes, J., Zschaler, S., Aßmann, U.: Reuseware – adding modularity to your language of choice. Proc. of TOOLS EUROPE 2007: Special Issue of the Journal of Object Technology (to appear) (June 2007)

31. Alves, A., Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., Sterling, Knig, D., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web services business process execution language version 2.0. OASIS Committee Draft (May 2006)