

Integrating a Performance Analysis Kit into Model-Driven Development

Matthew J. Rutherford

Department of Computer Science
University of Colorado
Boulder, Colorado, 80309-430 USA
`rutherfo@cs.colorado.edu`

Abstract. Model-driven development is a generative programming technique in which domain-specific features and engineering decisions are described using models of desired properties and behavior at various levels of abstraction. Producing models at the requisite level of specificity is an expensive undertaking. Generally, organizations offset the costs associated with generative techniques like model-driven development by reusing high-level domain-specific models across members of a product family. Most generative techniques focus on the automation of design and implementation activities. However, we believe that the increased level of formalism required by model-driven development can also be leveraged during other activities like analysis, verification, testing, maintenance and deployment. Furthermore, as design and implementation activities become increasingly automated, we feel that verification and testing activities must leverage the models and the generative engine in order to be feasibly address the generated systems. This paper presents a high-level view of a performance analysis kit as an example of a verification activity that can be integrated into model-driven development. By automating simulation code generation and instrumentation, data collection and data reduction, and by using existing models as inputs into the performance analysis kit, we hope to make it feasible to conduct performance analyses in an ongoing manner throughout the development life cycle. Our research will involve the creation of the model-driven performance analysis kit, its integration into a model-driven development environment, and an evaluation of the kit's utility during the design and implementation of a complex distributed system.

1 Introduction

Model-driven development is a generative programming technique in which domain-specific features and engineering decisions are described using models of desired properties and behavior at various levels of abstraction. Typically, different models are combined and repeatedly refined until they are detailed enough to be rendered into programming language source code and other artifacts. As described by Czarnecki and Eisenecker [4], the benefits of generative programming techniques like model-driven development include improved developer productivity,

high-level reuse, improved implementation consistency, and enhanced runtime efficiency of the resulting software.

Generally, organizations are motivated to use generative techniques like model-driven development by the promise of high-level reuse of models across members of a product family. In this context, most generative techniques are focused on the automation of design and implementation activities. However, we believe that the formalism required by model-driven practices can also be leveraged during activities that take place during testing, verification, deployment and maintenance. In fact, we believe that for model-driven development to achieve widespread adoption, these “secondary” activities must benefit at least as much as the “primary” activities of design and implementation.

To demonstrate our claim, we focus on the challenge of supporting ongoing simulation-based performance analyses throughout the development life-cycle of a distributed system. Traditionally, performance simulations are only conducted in early phases of the development life-cycle, when designers and architects are trying to validate their high-level design decisions before dedicating the resources needed to actually implement the system. Simulations are not generally conducted in an ongoing fashion because of the effort required to maintain the simulation code in parallel with the application code. However, we believe that ongoing performance simulations can be a useful way of ensuring that the performance of the system is not being degraded as development proceeds.

Performance analysis is an important part of the production of high-quality distributed software systems. In general, performance analysis is concerned with properties like execution speed, accuracy of results and memory usage. For distributed software systems, these properties must be analyzed in the context of concurrent users accessing multiple software components communicating across a network that exhibits variable latency and connectivity. Experimental performance analyses require the software system to be implemented before they can be conducted. In order to be useful during design activities, our performance analysis kit will employ straightforward discrete event simulations. Two desirable goals of design-time simulation are (1) applicability of results to the actual system implementation, and (2) the ability to analyze many different decisions efficiently. Unfortunately, in many cases, there is tension between the fulfillment of these goals since the level of detail needed to satisfy (1) takes time to achieve, thereby reducing the time left for (2).

In our research, we intend to show that careful integration of performance analysis and model-driven development allows designers and developers to continually analyze the decisions they are making without requiring a large amount of extra effort to keep the simulation code up-to-date with the current specifications. By showing how model-driven development techniques can integrate with analysis techniques we hope to convince the reader that generative techniques are not only appropriate for design and implementation activities, but also for a host of different activities throughout the development life cycle.

2 Discussion

In the model-driven development process, developers use a modeling language to describe various properties and behaviors of the software system they are creating. Different aspects of the system are described using different models. Formal semantics are associated with the models and they are augmented, transformed, merged and expanded until they are detailed enough to serve as models of specific software artifacts. Developers are therefore required to create both models and transformations between models. Meta-models, describing the models themselves, may also be required.

Using these definitions, a *model-driven kit* is a collection of meta-models, models and transformations that have well defined semantics. Meta-models describe input models to the kit that must be satisfied for the kit to be used. Once these input models have been supplied, the transformations are applied generating the expected output models, meta-models or transformations.

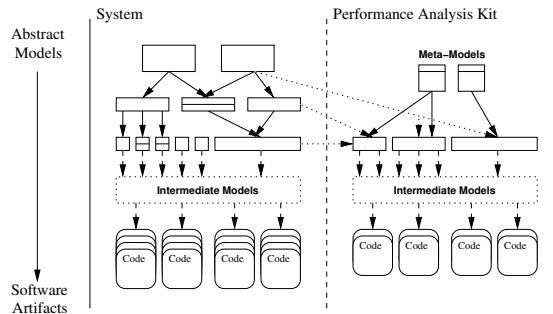


Fig. 1. Using a Performance Analysis Kit with Model-Driven Development

Figure 1 shows how the performance analysis kit relates to the rest of the model-driven development. Moving from top to bottom in Figure 1 the models progress from abstract to concrete. The left side of Figure 1 shows the main system models, meta-models and transformations. The right side of the figure shows the performance analysis kit. The dotted arrows moving from left to right represent the transformations that are needed to map existing system models into models that satisfy the input requirements of the kit. Throughout the course of development these mappings from system models to kit input models have to be maintained as the system models change. However, once the kit's inputs are satisfied, the resulting outputs are automatically generated.

2.1 Performance Analysis Kit

As described above, the performance analysis kit, like any other model-driven kit, is a collection of meta-models, models, and transformations. Once the kit's input

models have been satisfied the transformations are applied, and the outputs are generated.

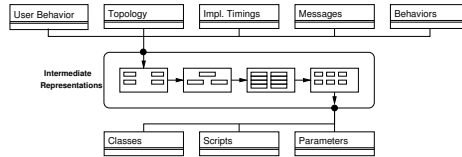


Fig. 2. Performance Analysis Kit

Figure 2 depicts the flow of data through this model-driven kit. Five meta-models are provided by the kit, these describe user behavior, network topology, implementation timings, communication messages, and application behaviors to be simulated. Each of these five meta-models must be satisfied by models provided by the developer. Once the input models have been provided, the kit's outputs can be created. These outputs include the source code of the simulation software and any data collection and analysis software that is needed. Additionally, the kit generates scripts to initiate the analysis and configuration files that may be needed.

So, for application developers to take advantage of this kit, they must create additional translations, and possibly additional intermediate models in order to map their system models into the kit's input models. It may be the case that some of the models required by the performance analysis kit represent information that is not available elsewhere in the system, in which case the developer would manually create the input models. The maintenance of these mappings from system models into performance analysis kit models is one additional development activity that must be performed as the system matures. However, because the developer has control over this process, they also dictate the level of detail that is present in the performance analysis.

2.2 Integration with Model-Driven Development Environment

In order to evaluate the effectiveness of the performance analysis kit, it must be integrated into a model-driven development environment that supports the basic functionality outlined above. This includes providing developers with the ability to create and edit models, meta-models and transformations. Good tool support for ensuring that models satisfy their meta-models is crucial. We are currently in the process of searching for an appropriate model-driven development environment.

2.3 Evaluating the Performance Analysis Kit

In order to evaluate the effectiveness of our performance analysis kit we propose to evaluate its use during the development of a distributed software system that supports content-based networking [2], an advanced network service.

The performance analysis kit will be evaluated in the following ways:

- **Model Transformation Effort.** The major new development activity that is introduced by using the kit is the maintenance of the transformations that create the kit’s input models from the system models. We will need to determine how much additional effort this involves.
- **Effectiveness of Ongoing Performance Analysis.** We will evaluate the usefulness and effectiveness of ongoing simulation for attaining performance goals. The emphasis of this evaluation will be on the effectiveness of performance analysis as a design tool.
- **Benefits of Automation.** To demonstrate that a generative technique is appropriate for the task of ongoing performance analysis we will quantify how much the generated simulation code changes as development progresses, illustrating the challenges that a developer would face when attempting to accomplish this without the level of automation achieved by our kit.

3 Related Work

A well know model-driven development approach is the Object Management Group’s (OMG) Model Driven Architecture (MDA)[9]. The primary domain of this framework is enterprise-level systems based on distributed object frameworks like Sun’s Enterprise JavaBeans, Microsoft’s .Net and the OMG’s CORBA. In MDA, UML models are used to describe platform independent and platform specific properties and features of the system.

Gervais [7] outlines a methodology that is based on both MDA and the Open Distributed Processing Reference Model (RM-ODP) [1]. Gervais proposes a process for modeling the domain-specific features of a system in a “Behavioral Model” and the high-level technological features of the system in an “Engineering Model”. These would be merged into a platform-specific “Operational Model”, which could then be used as the basis for generating implementation artifacts. This lines up with our model-driven approach where different aspects of the system are modeled independently and subsequently combined as the models become less abstract.

A model-based approach to the development of web applications is described in [5, 6, 3]. In this study many of the benefits of model-driven development are demonstrated, however, the abstract models are not used as a basis for analysis, which we believe is a central benefit to the use of generative techniques.

Grundy, Cai, and Liu [8] discuss the the SoftArch/MTE system. This system enables the automatic generation of prototype distributed systems based on high-level architectural specifications. While their emphasis is on experimenting with actual implementations on dedicated testbeds, their idea of performing analysis on high-level models of systems lines up well with ours.

4 Conclusion

Generative techniques, including model-driven development, are generally focused on the automation of design and implementation activities. We believe that some generative techniques are also appropriate for supporting testing, analysis, verification, deployment and maintenance activities by leveraging the higher levels of formalism that are typically present in model-driven systems.

In this paper we presented our ideas for a model-driven performance analysis kit to be integrated into the model-driven development process. We presented the high-level overview of our kit and also outlined how we intend to evaluate its utility. We hope that this initial work on a performance analysis kit will lead us to investigate other model-driven kits that provide automated support for development tasks.

References

1. ISO IS 10746-x. ODP reference model part x. Technical report, International Standards Organization, 1995.
2. Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, number 2538 in Lecture Notes in Computer Science, pages 59–68, Scottsdale, Arizona, October 2001. Springer-Verlag.
3. Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):137–157, 2000.
4. K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
5. Piero Fraternali and Paolo Paolini. Model-driven development of web applications: the AutoWeb system. *ACM Transactions on Information Systems (TOIS)*, 18(4):323–382, 2000.
6. Franca Garzotto, Paolo Paolini, and Daniel Schwabe. Hdma model-based approach to hypertext application design. *ACM Transactions on Information Systems (TOIS)*, 11(1):1–26, 1993.
7. M.P. Gervais. Towards an MDA-oriented methodology. In *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC 2002)*, 2002.
8. J. Grundy, Y. Cai, and A. Liu. Generation of distributed system test-beds from high-level software architecture descriptions. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, 2001.
9. J. Miller and J. Mukerji. Model driven architecture (MDA). *OMG Document ormsc/2001-07-01*, July 2001.