Preface

One of the basic principles of software engineering is abstraction, which mainly refers to separation of the essential from the non-essential. In terms of software development, the essential usually refers to the functionality to be implemented and the non-essential to aspects such as the technical platform on which the software will eventually be deployed. However, non-essential aspects are not unimportant. They also have to be considered when designing and developing a software system, but they do not have to be considered at the very first stage when more fundamental issues have to be considered.

Abstractions are provided by models. A model is mainly a representation of the essential aspects of the underlying subject and thus contains less complexity. Less complexity obviously allows the prediction of system characteristics, analyzing specific properties, and also communicating with the various roles involved in the development process more easily. However, implementing a model means expressing it at a very low level of abstraction, i.e. at a level at which it is understood by a computer.

Modeling and model transformation to the required abstraction level constitute the core of model-driven development. In model-driven development, essential aspects of software are expressed in the form of models, and transformations of these models are considered the core of software development. Models can particularly be transformed into a technical implementation, i.e. a software system. Such an approach can avoid restricting oneself to a specific technology in the early stages of the development process and can ensure a consistent architecture throughout the lifecycle of a software system.

The aim of this book is to give an overview of the current achievements in model-driven development. In the introductory chapter *Models, Modeling, and Model-Driven Architecture (MDA)*, Brown, Conallen and Tropeano first explain the terminology used in the following chapters of the book and introduce basic principles and methods in model-driven development. Achievements in model-driven development are then considered from a conceptual point of view in Part I of the book that comprises the following chapters:

- *A Systematic Look at Model Transformations.* Metzger focuses on model transformations and presents a classification scheme to consider the differences between the modeled system, the model itself and the formalism used.
- *Tool-support for Model-Driven Development of Security-Critical Systems with UML.* Jürjens and Shabalin show the use of UML in model-driven development. In particular, they give a formal semantics for a subset of UML which can be used to analyze the interaction of a system with its environment and UML specifications.
- *Caste-Centric Modeling of Multi-Agent Systems: The CAMLE Modeling Language and Automated Tools.* Zhu and Shan introduce the CAMLE approach to model-driven development of multi-agent systems by combining graphical modeling with formal specification.
- *Using Graph Transformation for Practical Model Driven Software Engineering.* In this chapter, Grunske et al. consider model transformations using graph transformation theory, in particular to specify and apply model transformations.
- *A Generalized Notion of Platforms for Model Driven Development.* Atkinson and Kühne consider two of the basic terms in model-driven development, platform and platform model. They show the origin of these terms and propose an alternative definition for them.

Part II then considers technical achievements and technical infrastructures of model-driven development in the following chapters:

- *A Tool Infrastructure for Model-Driven Development Using Aspectual Patterns.* Hammouda introduces a concern-based approach to model-driven development and presents a tool, called MADE, which particularly supports model generation, checking and tracing.
- *Automatically Discovering Transitive Relationships in Class Diagrams.* Egyed considers the problem of abstracting class diagrams of certain complexity with tool support. The approach proposed uses a large number of abstraction rules and is used for model understanding, consistency checking and reverse engineering.
- *Generic and Domain-Specific Model Refactoring using a Model Transformation Engine.* Zhang, Lin and Gray propose an approach for refactoring at the model level with the use of behavior-preserving transformations. Their chapter also covers a model transformation engine for refactoring various types of models.
- *A Testing Framework for Model Transformations.* Lin, Zhang and Gray discuss validation and verification of model transformation at the model level rather than late in the development process at the source code level. The framework presented is integrated in the transformation engine presented in the previous chapter and provides means for typical testing activities.
- *Parallax – An Aspect-Enabled Framework for Plug-in-Based MDA Refinements Towards Middleware.* Silaghi and Strohmeier present the Parallax framework, an open and extensible tool which particularly supports configuring application designs with regard to specific middleware concerns and adapting to different middleware infrastructures.

- *Evolution and Maintenance of MDA Applications.* Seifert and Beneken investigate the life cycle of applications developed according to the model-driven development approach. They particularly focus on long-term aspects and consider the maintenance of such applications and the progress in model-driven development.

The chapters in Part III finally summarize experience gained in actual projects employing model-driven development:

- *Intents and Upgrades in Component-Based High-Assurance Systems.* Elmqvist and Nadjm-Tehrani describe their experience using model-driven development in the area of high-assurance components, particularly components used as part of embedded systems.
- *On Modeling Techniques for Supporting Model Driven Development of Protocol Processing Applications.* Alanen et al. use model-driven development in the area of protocol processing applications. They give an overview of a respective method and summarize their experience.
- *An Integrated Model-driven Development Environment for Composing and Validating Distributed Real-time and Embedded Systems.* Trombetti et al. employ model-driven development in the area of distributed real-time and embedded applications. They present an integration of tool suites for model-driven development and model checking in this area.
- *A Model-Driven Technique for Development of Embedded Systems Based on the DEVS formalism.* Wainer, Glinsky and MacSween propose a model-driven approach to the development of embedded systems with real-time constraints based on the formal technique of DEVS, and summarize their experience using this approach.
- *Model Driven Service Engineering.* Bræk and Melby consider problems associated with expressing platform-independent models and their behaviors, and also discuss how to handle implementation and platform-dependent properties. They suggest possible solutions to those problems based on their experience.
- *Practical Insights into Model-Driven Architecture: Lessons from the Design and Use of an MDA Toolkit.* Brown, Conallen and Tropeano finally summarize their experience in the design and use of a model-driven architecture toolkit at IBM.

Work on this book officially began in April 2004 with an email to the `seworld` mailing list, which was followed by individual invitations sent to the leading experts of the field. Researchers and practitioners have been invited to summarize their research results and experience in model-driven development in the form of book chapters. Fortunately, we received a large number of very high-quality contributions, which shows that model-driven development will not be a short-lived hype in software engineering. We are very grateful for the contributions and would like to thank all authors for their effort.

Leipzig and Bonn,
May 2005

Sami Beydeda
Matthias Book
Volker Gruhn

Contents