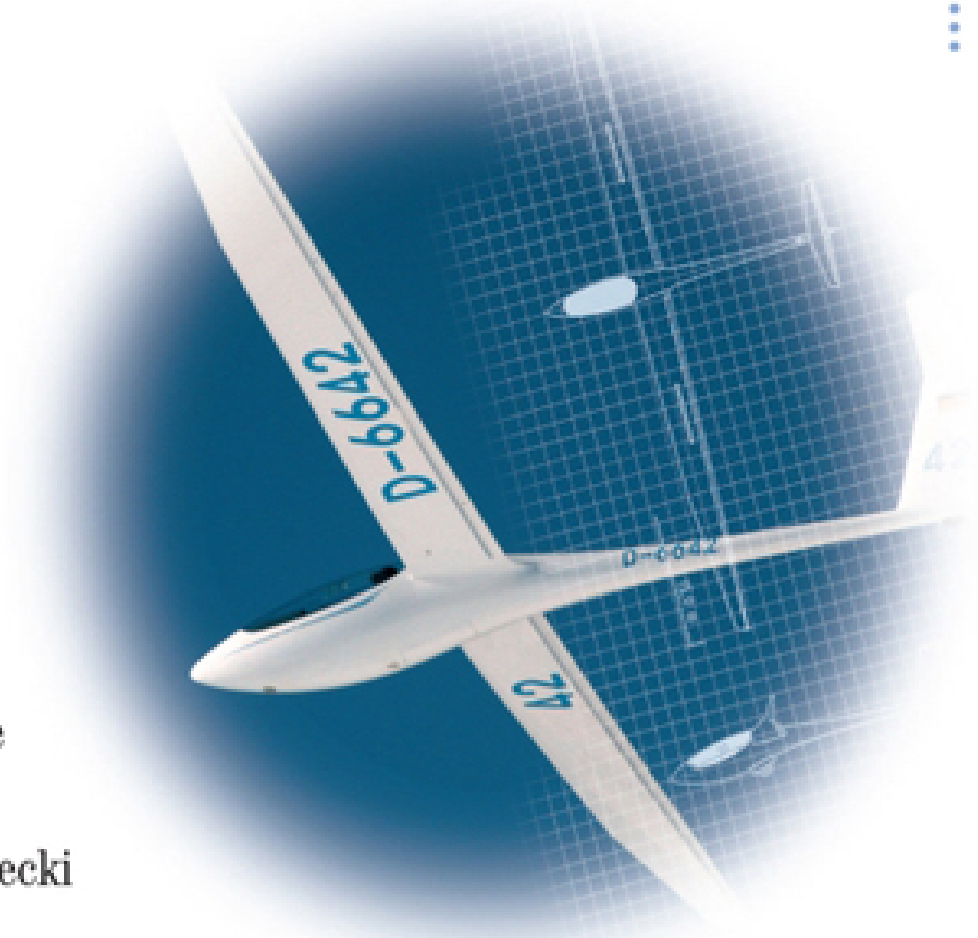
WILEY

TIMELY. PRACTICAL. RELIABLE.

Model-Driven Software Development

Technology, Engineering, Management



Thomas Stahl,
Markus Völter

with Jorn Bettin, Arno Haase
and Simon Helsen

Foreword by Krzysztof Czarnecki

Translated by Bettina von Stockfleth

Contents

Model-Driven Software Development

Model-Driven Software Development

Technology, Engineering, Management

Thomas Stahl

and

Markus Völter

with

Jorn Bettin, Arno Haase and Simon Helsen

Foreword by Krzysztof Czarnecki

Translated by Bettina von Stockfleth

John Wiley & Sons, Ltd

Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk

Visit our Home Page on www.wiley.com

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Stahl, Thomas (Tom)

Model-driven software development : technology, engineering, management / Thomas Stahl and Markus Völter, with Jorn Bettin, Arno Haase, and Simon Helsen ; foreword by Krzysztof Czarnecki ; translated by Bettina von Stockfleth.

p.cm.

Includes bibliographical references and index.

ISBN 0-470-02570-0 (pbk. : alk. paper)

1. Computer software—Development. I. Völter, Markus. II. Title.

QA76.76.D47S697 2006

005.1—dc22

2006007375

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN-13: 978-0-470-02570-3
ISBN-10: 0-470-02570-0

Foreword

by Krzysztof Czarnecki

Modeling is a key tool in engineering. Engineers routinely create models when analyzing and designing complex systems. Models are abstractions of a system and its environment. They allow engineers to address their concerns about the system effectively, such as answering particular questions or devising required design changes. Every model is created for a purpose. A particular model may be appropriate for answering a specific class of questions, where the answers to those questions will be the same for the model as for the actual system, but it may not be appropriate for answering another class of questions. Models are also cheaper to build than the real system. For example, civil engineers create static and dynamic structural models of bridges to check structural safety, since modeling is certainly cheaper and more effective than building real bridges to see under what scenarios they will collapse.

Models are not new in software development. Over the past few decades, the software industry has seen numerous analysis and design methods, each with its own modeling approaches and notations. More recently, we have witnessed the remarkable progress of Unified Modeling Language (UML), which now has a larger market penetration than any single previous modeling notation. Still, analysis and design models rarely enjoy the same status as code. The reality of most software projects is that models are not kept up-to-date with the code, and therefore they become obsolete and useless with time.

Model-Driven Software Development (MDSD) puts analysis and design models on par with code. Better integration of such models and code should significantly increase the opportunity to effect change through the models, rather than simply modifying the code directly. MDSD encompasses many different techniques across the entire spectrum of software development activities, including model-driven requirements engineering, model-driven design, code generation from models, model-driven testing, model-driven software evolution, and more.

The Model-Driven Architecture (MDA) initiative by the Object Management Group (OMG) has also certainly contributed a great deal to the recent surge of interest in software modeling and model-driven techniques. But the effects of that initiative have been both good and bad. On the positive side, I'm glad that modeling has been moved into the center of interest and that organizations are now trying to figure out how their current practices can be leveraged through model-driven techniques. At the same time, the marketing hype around MDA has tended to create some unrealistic expectations. Putting all this hype aside, I do think that MDSD has great ideas to offer, many of which can be put to work in practical situations today. Realizing these potentials requires a solid understanding of current MDSD technology, its applicability, and its limitations.

The authors of this book are at the forefront of MDSD research and practice. Markus and Jorn have organized and participated in a series of MDSD workshops at several OOPSLA conferences. Simon has participated in OMG's standardization efforts on model transformation. All the authors are pioneering this technology in practice in several domains, ranging from enterprise applications to embedded software in both small and large organizations, such as b+m, Siemens, and BMW.

I’m very pleased to introduce this book to you. In my view, this is one of the rare books in the model-driven space that talks not only about the vision, but also about what is possible today, and how to do it. After a minimal but necessary dose of basic concepts and terminology, the authors cover a wide range of MDSD technology topics such as metamodeling, component architectures and composition, code generation, model transformation, MDA standards, and MDSD tools. I particularly like the hands-on approach that the authors have taken. They illustrate available tools and techniques through concrete modeling examples and code snippets, and they give numerous practical tips and ‘mind-the-gap’ hints. In addition to the technology topics, the authors also present a comprehensive treatment of essential software engineering aspects such as testing, versioning, process management, and adoption strategies, as they apply to MDSD. The content is topped off with two case studies, that were inspired by realistic applications from the authors’ collective experiences.

The authors present a broad perspective of MDSD that goes beyond MDA to cover a range of related approaches including software product lines, domain-specific languages, software factories, and aspect-oriented and generative programming. As in any young, dynamic, and still evolving field, the abundance of competing ideas, concepts, and parallel terminologies in today’s model-driven space can be bewildering. As a result, the authors had to do a lot of ‘sifting through the mud’ to give us a clear and balanced picture of the entire model-driven field. In this book, they have done just that, tremendously well.

I invite you to explore this new and exciting field, and this book is a great place to start!

Krzysztof Czarnecki
Waterloo, January 2006

Part I

Introduction

Chapter 1

Introduction

1.1 The Subject of the Book

This book is about *Model-Driven Software Development*, or ‘MDSD’. A less precise but common name for this discipline is *Model Driven Development* (MDD). Maybe you wonder why we decided to write such a book. We believe that Model-Driven Software Development is quite important, and will become even more so in the future. It is the natural continuation of programming as we know it today.

The application of models to software development is a long-standing tradition, and has become even more popular since the development of the Unified Modeling Language (UML). Yet we are faced with ‘mere’ documentation, because the relationship between model and software implementation is only intentional but not formal. We call this flavor of model usage *model-based* when it is part of a development process. However, it poses two serious disadvantages: on one hand, software systems are not static and are liable to significant changes, particularly during the first phases of their lifecycle. The documentation therefore needs to be meticulously adapted, which can be a complex task – depending on how detailed it is – or it will become inconsistent. On the other hand, such models only indirectly foster progress, since it is the software developer’s interpretation that eventually leads to

implemented code. These are the reasons why – quite understandably – many programmers consider models to be an overhead and see them as intermediate results at best.

Model-Driven Software Development has an entirely different approach: Models do not constitute documentation, but are considered equal to code, as their implementation is automated. A comparison with sophisticated engineering fields, such as mechanical engineering, vividly illustrates this idea: imagine, for example, a computer-controlled mill that is fed CAD[1] data that enables it to transform a model into a physical workpiece automatically. Or consider an automotive production line: your order for a car that includes custom features is turned into reality. Here, the actual production process is mostly automated.

These examples demonstrate that the *domain* is essential for models, just as for automated production processes. Neither the customer-oriented ‘modeling language’ for car manufacture – in this case, an order form – nor the manufacturer’s production line are able to build prefabricated houses, for example.

MDSD therefore aims to find domain-specific abstractions and make them accessible through formal modeling. This procedure creates a great potential for automation of software production, which in turn leads to increased productivity. Moreover, both the quality and maintainability of software systems increase. Models can also be understood by domain experts. This evolutionary step is comparable to the introduction of the first high-level languages in the era of Assembler programming. The adjective ‘driven’ in ‘Model-Driven Software Development’ – in contrast to ‘based’ – emphasizes that this paradigm assigns models a central and active role: they are at least as important as source code.

To successfully apply the ‘domain-specific model’ concept, three requirements must be met:

- Domain-specific languages are required to allow the actual formulating of models.
- Languages that can express the necessary model-to-code transformations are needed.
- Compilers, generators or transformers are required that can run the transformations to generate code executable on available platforms.

In the context of MDSD, graphical models are often used, but this is neither mandatory nor always suitable. Textual models are an equally feasible option. Typically, these models are translated into programming language source code to enable their subsequent compilation and execution.

1.1.1 MDA

If you are familiar with the Object Management Group's (OMG) Model Driven Architecture (MDA), you might think that this sounds a lot like MDA. This is correct to a certain extent. In principle, MDA has a similar approach, but its details differ, partly due to different motivations. MDA tends to be more restrictive, focusing on UML-based modeling languages. In general, MDSD does not have these restrictions. The primary goal of MDA is interoperability between tools and the long-term standardization of models for popular application domains. In contrast, MDSD aims at the provision of modules for software development processes that are applicable in practice, and which can be used in the context of model-driven approaches, independently of the selected tool or the OMG MDA standard's maturity.

At present (2005) the MDA standardization process is still in its fledgling stages, which means that, on one hand, some aspects of the original MDA vision must be omitted, while others need be interpreted pragmatically to get practicable results. On the other hand, a practical *methodological*

support for MDA is not necessarily the OMG's main focus. This is in part also reflected by MDA's goals.

In this book we take a closer look at the relationship between MDA and MDSD. For now, it is safe to state that MDA is a standardization initiative of the OMG focusing on MDSD.

1.2 Target Audience

Specific concepts, terminology and basic ideas must be understood by all those involved in an MDSD project, otherwise it cannot be completed successfully. The introductory chapters of this book are therefore mainly dedicated to these aspects.

1.2.1 Software Architects

Three aspects concerning MDSD are relevant for the software architect:

- First, the approach requires a clear and concise definition of an application's architectural concepts.
- Furthermore, MDSD often takes place not only as part of developing an entire application, but in the context of creating entire product lines and software system families. These possess very specific architectural requirements of their own that the architects must address.
- In addition, a totally new development approach must be introduced to the project. This is due to the separation of models and modeling languages, of programs and generators, of respective tools and specific process-related aspects.

All these issues will be discussed in this book.

1.2.2 Software Developers

When dealing with a software development paradigm, it almost goes without saying that the role of the software developer is pivotal. To some extent, MDSD implies more precise and clearer views of aspects such as the meaning of models, the separation of domain-specific and technical code, the relationship between design and implementation, round-trip problems, architecture and generation, framework development, versioning and tests. When applied correctly, MDSD will make the software developer's work much easier, help to avoid redundant code, and enhance software quality through the use of formalized structures.

1.2.3 Managers and Project Leaders

Economic considerations such as the cost-value ratio or the break-even point underlie the decision to use Model-Driven Software Development. There is no 'free lunch': model-driven software comes with a price, too. There are many project contexts for which a model-driven approach can be recommended, but there are some circumstances under which we would advise against it. Even though the focus of this book is technical, we will take into account organizational and economic aspects that are relevant from the project's or company's viewpoint.

A model-driven approach also impacts project organization and team structure as well as the software development process. We will address this as well.

1.3 The Goals of the Book

The goal of the book is to convince you, the reader, that MDSD is a practicable method today, and that it is superior to conventional development methods in many cases. We want to encourage you to apply it sooner rather than later, since MDSD is neither merely a vision nor dry theory. To this end, we want to equip you with everything you need. If you already practice MDSD, this book might offer you some advice or provide further insight into specific topics or fields.

More specifically, we pursue a number of ‘subgoals’ - independent of the book’s structure – that we want to elaborate briefly here.

First, we introduce the theoretical framework for MDSD, its basic concepts and terminology. We also touch on related topics and approaches, such as OMG’s Model Driven Architecture (MDA). We also want to provide hands-on help for specific MDSD-relevant issues. Among these are metamodeling (with UML and MOF[2]), code generation, versioning, testing, as well as recommendations for choosing the right tools. Organizational and process-related issues are also very important to us. Additionally, we want to argue for MDSD from an economical standpoint.

Although it is impossible to work without at least some theoretical basis, the book first and foremost aims to provide practical support, as well as taking a more detailed look at some of the relevant theoretical issues mentioned above. Best practice, as well as the dissemination of concrete experiences are important to us, as well as, in part, personal opinions. A number of case studies from various domains supplement the more detailed parts of the book.

We also wish to answer prevailing questions and address current discussions, so an outlook on trends and visions in the MDSD context completes the book.

1.4 The Scope of the Book

This is not an MDA book. We describe the basic concepts and terminology of the OMG MDA standard as well the underlying vision, and we also offer a synopsis of the current state of standardization (see Chapter 12). However, the book represents our own views and experiences. Secondary literature about MDA can be found in sources such as [Fra02], as well as in the OMG specification itself.

Our book does not intend to define a cohesive, heavyweight MDSD development process. Instead, we report on best practices that lend themselves to being used in agile processes, as described by Crystal [Coc01], and in the context of product line development for the construction of customized development processes (see Section 16.2).

1.5 The Structure of the Book and Reader Guidelines

This book describes the model-driven approaches which the authors have successfully applied in practice for many years. We look at the subject-matter from a technological as well as from an engineering and management perspective, as you will see from the book's structure.

- *Part 1 – Introduction*. This part contains the introduction you are reading, plus an explanation of the most important basic ideas behind MDSD, and the basic terminology derived from the MDA. We then proceed to address the architecture-centric flavor of MDSD, which is ideally suited for a practice-oriented introduction. We convey the concrete techniques based on a comprehensive case study from the e-business/Web application field, followed by a more comprehensive MDSD concept formation building on the points made. This chapter is extremely important, particularly because the rest of the book is based on the terminology defined here. This is also where you can find the conceptual, artifact-related definition of MDSD. The first part of the book is completed by a classification of and distinction between related topics such as agile software development.
- *Part II – Domain Architectures*. Domain architecture is the core concept of MDSD. Among other aspects, it contains the domain's modeling language and the generation rules that are supposed to map the models to a concrete platform. This part of the book covers best practices for the construction of such domain

architectures. The chapter on metamodeling forms its basis, followed by a detailed examination of the special role of the target architecture in the MDSD context. The three following chapters take a more detailed look at building transformations, including a description of code-generation techniques and QVT-based model-to-model transformations (QVT is the OMG's standard for model-to-model transformations). A short comparison with interpreter-based approaches is also included. For building domain architectures, generic tools are best used, so the chapter on tool architecture and selection provides some background for this. Finally, we take a deeper look at the MDA standard in the last chapter of Part II.

- *Part III – Processes and Engineering.* In the third part of the book we deal with process-related aspects of MDSD and engineering issues that assume specific characteristics through MDSD. Here at last it should become clear that MDSD is not just a technology. We present a number of best practices that can be combined into a practical and pragmatic development process, look at architecture development, and take a glimpse into product-line engineering. Following that, we tackle testing and versioning issues. We finally look at two case studies, one from the embedded domain, the other from the world of enterprise systems.
- *Part IV – Management*. The last part of the book is aimed primarily at IT managers and project leaders. It can largely be read independently from the rest of the book. We take a closer look at economic and organizational aspects and discuss adoption strategies. The first chapter of this part includes a FAQ[3] section of MDSD-related questions.

We have taken the utmost care in structuring this book so that its didactic effect is optimal when read sequentially in

spite of its cyclic dependencies. However, since we address a divergent audience, some readers might initially wish to read the book selectively. In this context, please note that readers whose interest is primarily technical and who already possess some MDA knowledge can start directly with the case study in Part I, continue with Chapter 4, then immerse themselves in the technical issues addressed in Parts II or III before moving on to the rest of the book.

If you want to know what the economic advantages of MDSD are before learning about MDSD in more detail, please read Chapter 18 first. To gain a better understanding of it, we recommend that you also read Chapter 2.

1.6 The Accompanying Web Site

Some topics dealt with in this book are undergoing rapid evolution, while others could only be touched upon due to space limitations. The book's accompanying Web site can be found at *http://www.mdsd-book.org*. You can find up-to-date information there, as well as interesting links that we update on a regular basis.

1.7 About the Authors

Thomas Stahl works as chief software architect at b+m Informatik AG, where he is responsible for project-centric architecture management, including reusable software assets. He creates software architectures and frameworks and accompanies their use in both large and small projects. He also works as a consultant. His main focus is currently the field of Model-Driven Software Development, in which he has significant and practical long term experience. The

creation of a good MDSD-generator framework was a pioneering effort. It is a popular Open Source project (*http://www.openArchitectureWare.org*) and is supported by a very active developer community. Besides his project-related work in several domains, Thomas writes articles for IT magazines and speaks at software conferences. He spends his spare time, among other things, as an active musician. He can be reached at *t.stahl@bmiag.de*

Markus Völter works as an independent consultant in software technology and engineering. His work focuses on software architecture and Model-Driven Software Development, as well as on middleware. Markus has extensive experience with these topics in many sectors, including the automotive industry, science, health care, telecommunications, and banking, as well as Web-based systems and telematics. He has worked and consulted for many leading enterprises, mostly but not exclusively in Germany, in projects ranging from three to 150 developers. Markus is also a regular speaker at international software conferences, as well as an active member of the international patterns community.

In addition to this book and its German predecessor, Markus has also co-authored two patterns books, *Server Component Patterns* and *Remoting Patterns*, both published in Wiley's Software Design Pattern series. He has also contributed to a German book on software architecture. As with Tom Stahl, Markus is also a contributor to the openArchitectureWare framework. When not working, Markus spends his time in his sailplane. He can be reached via *http://www.voelter.de* or at *voelter@acm.org*.

1.8 About the Cover

When we discussed the cover we thought that we wanted something that resembled the concept of 'model-driven' in