

Model Driven Architecture



Dr. Roland Petrasch ist freier IT-Berater und Professor für Informatik (Lehrgebiet Software-Engineering) an der TFH Berlin. Seine Lehr- und Forschungsschwerpunkte sind u.a. Vorgehensmodelle, IT-Projekt- und Qualitätsmanagement, Modelle und Methoden des Software-Engineerings sowie Requirements- und Usability-Engineering. Seit über 20 Jahren beschäftigt er sich mit der Software-Entwicklung und arbeitete in der Industrie als Software-Entwickler, Berater und Projektleiter.



Dipl.-Inf. (FH) Oliver Meimberg studierte Informatik an der TFH Berlin. Seit ca. zehn Jahren beschäftigt er sich mit Methoden des Software-Engineerings, insbesondere mit der generativen Programmierung und der modellgetriebenen Software-Entwicklung. Er ist Geschäftsführer des Software-Unternehmens form4 GmbH & Co. KG.

Dipl.-Inf. (FH) Karsten Thoms beschäftigt sich schwerpunktmäßig mit modellgetriebenen Entwicklungsverfahren, der J2EE-Plattform und mit bekannten Java Open-Source Frameworks. Er ist aktiv an der Entwicklung des openArchitectureWare-Projekts beteiligt. Als Senior-Consultant bei der itemis GmbH & Co. KG begleitet er Kundenprojekte über alle Projektphasen. Zurzeit ist Herr Thoms an der Entwicklung eines Produktes bei T-Systems beteiligt, wo er u.a. die modellgetriebene Entwicklung unter Verwendung von openArchitectureWare einführt.

Dipl.-Inf. (FH) M.Sc. Florian Fieber studierte Informatik und Information Systems an der TFH Berlin und der Pforzheim Graduate School. Er hat mehrjährige Berufserfahrung als Software-Entwickler und war u.a. bei IBM in Deutschland und dem Software-Unternehmen Openly Informatics Inc. in den USA tätig. Zur Zeit arbeitet Herr Fieber als Forschungsassistent an der TFH Berlin und beschäftigt sich dabei mit modellbasierter Software-Entwicklung sowie dem Projekt- und Qualitätsmanagement.

Roland Petrasch · Oliver Meimberg

Model Driven Architecture

Eine praxisorientierte Einführung in die MDA

Mit Gastbeiträgen von Florian Fieber und Karsten Thoms



dpunkt.verlag

Roland Petrasch
petrasch@SoftwareQuality.de

Oliver Meimberg
oliver.meimberg@form4.de

Lektorat: René Schönfeldt
Copy-Editing: Sandra Gottmann, Münster
Satz & Herstellung: Birgit Bäuerlein
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: Koninklijke Wöhrmann B.V., Zutphen, Niederlande

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

ISBN 3-89864-343-3

1. Auflage 2006
Copyright © 2006 dpunkt.verlag GmbH
Ringstraße 19
69115 Heidelberg

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen. Insbesondere sind Model Driven Architecture, MDA, Object Management Group, Unified Modeling Language, UML, MOF, XMI Warenzeichen oder eingetragene Warenzeichen der Object Management Group (OMG), Inc., USA

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

5 4 3 2 1 0

Geleitwort

The world is full of ancient and immense tributes to the builder's art. Notre Dame Cathedral in Paris inspires to worship God, as it was intended to do – imagine being a 15th century peasant and coming upon such magnificence! Angkor Wat, once freed from the jungle that overran it, brings the same sense of awe (and in fact, for the same general purpose). Eighth century monuments in the first capital of united Japan, in Nara, tower over the visitor, reminding him of the significance of that unification and the power of the warlords and their newly-imported Buddhist ideas. Entire cities of pyramids to the Gods, to the stars, to the seasons fill plains near México City in Teotihuacán, and on the sands near the Nile Delta of Egypt.

That these ancient monuments still stand, some after literally thousands of years, appears to speak to deep knowledge about the construction of permanence. Especially visitors from a young country (such as my own) often wonder at the abilities of technologically poor builders and societies to create such gravity-defying feats. How in the world did 15th Century European builders know how to build the Catedral de México to stand five hundred years against gravity, and not collapse? Even more amazing, ancient Egyptian temples have stood millennia – how did they do it?

Just one last example: the enormous Church on the Spilled Blood, along the Griboedova Canal in St. Petersburg. Tall and imposing, with nearly every surface inside & out covered with magnificent mosaic tile work, surviving even the 75-year experiment with Communism that found it used primarily as a warehouse. How did its builders know how to build such permanence?

In fact, this final example is different than all the others: the Church on the Spilled Blood, in historical terms, was built only yesterday. A late 19th Century construction, it was *designed*. Architects made plans, plans which were used not only to sell an overall design to

the buyer (in fact, the Tsar of all the Russias), but also plans which were used by engineers to estimate stresses and strains; used by builders to calculate construction methodology and choose materials; used by procurers to choose & purchase those building materials and deliver them to the now-hallowed site.

In contrast to the Russian example, the ancient Egyptians and Mayans (and even the Medieval builders of Europe) were comparatively more impressive: they did *not* have all the engineering knowledge they needed to build their monuments. So how is it that these more ancient monuments survive? It makes the question even more pressing. The answer, of course, is that they were lucky. It has been estimated that some 90% of all Medieval cathedral construction failed and is no longer with us; I would be surprised if the odds were any better for the ancients. Even those still standing have featured partial collapses, repaired over the years; the Giralda Cathedral in Sevilla sports a dome much newer than most of its structure. The same is true across Europe, and all of the world.

By contrast, when one of the last governments of the Soviet Union decided to restore the masterful Church on the Spilled Blood, they had a running start: they had the plans, they had the blueprints. They did not have to reverse-engineer the structure to repair it to its former (and now current) beauty. There were even photographs, albeit monochrome, which helped in reconstruction. The same can be said of the magnificent Frauenkirche of Dresden; for half a century after World War II it remained a pile of rubble, yet now the lovely cathedral lives again, thanks to plans, blueprints and meticulous craftsmanship.

This is, in fact, what we call *engineering*: a craft which builds on knowledge, and *plans that construction*. And by that measure, it is unfair and unreasonable to call most software development in 2006 »engineering,« since plans rarely feature in software construction. This I daresay contributes to the absolutely abysmal odds of success in large and complex software development projects – by some estimates, not even 20% of software development projects result in systems built as needed, or used as intended. Complex systems, whether buildings or bridges, ships or software, are hard to build and must be properly planned, especially if they must be changed or maintained (or integrated with other systems) after delivery.

The »art« of software development is, fortunately, changing. Like the ghosts of Medieval European cathedrals that did *not* survive, dead software systems don't leave visible failures on the landscape. They do, however, burn a huge hole in many pocketbooks! The renewed interest in software development as an engineering discipline comes from the

convergence of several forces in the business and software worlds, primarily among these the move to model-driven development.

Most of the pushback against software design over the last 50 years has come from two facts: software appears easy to build (that is, it has a very low cost of entry, compared to the materials that make up buildings, silicon chips or ships); and software designs are not considered deliverables by those that purchase software systems (compare this to buildings – blueprints are absolutely demanded by purchasers of buildings, as they know those blueprints will be useful in the future!). The only way to get past these roadblocks is for modeling tools to deliver ease of use (i.e., to not raise the cost of entry for software development); and for models to be accepted (in fact, *demanded*) by software buyers (and for the same reason, so they can maintain and integrate the software over time).

This is the promise of model-driven engineering, whether it be ships or software: that the design is a deliverable, and that the design *is part of the development process*. And this is the promise of the Model Driven Architecture Initiative of the Object Management Group (OMG) – to deliver standards for software development that simplify software construction from software blueprints, but more importantly deliver lower maintenance and integration costs over the software life-cycle through the existence of understandable software designs in a standardized set of languages.

OMG's Model Driven Architecture (MDA) is a practical set of standards for delivering higher return on investment in software development, and all that remains is to deliver practical, day-to-day advice to the engineer to leverage these tools and techniques. Reducing theory to practice can be difficult, but in this case there is already half a decade of experience building, delivering, maintaining and integrating systems based on MDA-standard models. The National Cancer Institute and Chubb Insurance in the United States, Germany's Kreditwerk and Deutsche Bank Bauspar, Austrian Railways, and thousands of other business, governments & universities around the world have discovered the power of developing systems using models.

And soon you also will be able to! Enjoy this practical guide to delivering flexible, agile systems that meet real business needs and lower costs using Model Driven Architecture.

Richard Mark Soley, Ph.D.
Chairman and CEO
Object Management Group, Inc.
7 April 2006

Inhaltsverzeichnis

	Vorwort	1
1	Einleitung	9
1.1	Probleme bei der Software-Entwicklung	9
1.2	Artefakte	12
1.3	Formalisierung	19
	1.3.1 Formale Sprachen	19
	1.3.2 Formalisierung bei Artefakten der Software-Entwicklung	26
1.4	Zusammenfassung	33
2	Grundlagen der MDA	35
2.1	Überblick	35
2.2	Modelle und SW-Architekturen	39
	2.2.1 Modelle und Modellierung: grundlegende Begriffsbestimmung	39
	2.2.2 Modelle und Metamodelle	48
	2.2.3 Software-Architekturen	53
2.3	MDA-Spezifikationen	61
	2.3.1 Standard-Modellierungssprache: Unified Modeling Language und Object Constraint Language	61
	2.3.2 Erweiterungen der UML	73
	2.3.3 Meta Object Facility (MOF)	78
	2.3.4 Mapping von MOF-Modellen: XML und Java ...	84
	2.3.5 Weitere Spezifikationen	89
	2.3.6 MDA Foundation Model	90

2.4	Modelle und Plattformen im Kontext von MDA	93
2.5	Modelle der MDA: CIM, PIM und PSM	100
2.5.1	Computation Independent Model (CIM)	100
2.5.2	Platform Independent Model (PIM)	102
2.5.3	Plattform Model (PM)	103
2.5.4	Plattform Specific Model (PSM) und Implementation (PSI)	105
2.5.5	Transformationen	106
2.6	Zusammenfassung	111
3	Qualität, Modelle und Transformationen	113
3.1	Qualität von Modellen = Produktqualität	113
3.2	Vom Platform Independent Model zum Plattform Specific Model	125
3.3	Ansätze zur Code-Generierung und MDA-Transformationen	129
3.3.1	Code-Generierung	129
3.3.2	Transformationsansätze der MDA	131
3.3.3	Das Problem der erneuten Generierung (Re-Generierung)	135
3.4	Abschließendes Beispiel	139
3.5	Zusammenfassung	147
4	Entwicklungsprozesse und MDA	149
4.1	Software-Entwicklungsprozesse, Rollen und Methoden	149
4.2	Software-Produkt- und Prozessqualität	154
4.3	Entwicklungsprozesse für MDA-Projekte	157
4.3.1	Überblick	157
4.3.2	Vorgehensweise für die MDA im Rahmen dieses Lehrbuches	159
4.4	Zusammenfassung	165
5	Technisches Umfeld von MDA	167
5.1	Aufgaben und Arbeitsweise eines MDA-Tools	167
5.1.1	Aspekte eines MDA-Tools	167
5.1.2	Aspekte der UML-Modellierung	170
5.1.3	MDA-Suite oder »rohes« Framework?	171

5.2	Überblick: MDA-Tools	172
5.2.1	Bewertungskriterien	172
5.2.2	Kurzbeschreibung der Tools	174
5.2.3	Vergleich der MDA-Tools	176
5.2.4	Weitere Tools	179
5.3	AndroMDA	179
5.3.1	Überblick	179
5.3.2	Architektur	180
5.3.3	Mitgelieferte Cartridges	184
5.3.4	Einbinden in die Entwicklungsumgebung	184
5.3.5	Entwickeln eigener Cartridges	186
5.4	openArchitectureWare/Open Generator	190
5.4.1	Überblick	190
5.4.2	Architektur	191
5.4.3	Metamodell	194
5.4.4	Modellvalidierung	195
5.4.5	Einbinden in die Entwicklungsumgebung	196
5.4.6	Entwicklung eigener Generatoren	197
5.4.7	Ausblick auf openArchitectureWare 4.0	200
5.5	Zusammenfassung	201
6	Anwendungsbeispiel: Katalogsystem – Iteration 1	205
6.1	Einleitung	205
6.1.1	Vision des Anwendungssystems	206
6.1.2	Technologie und Zielplattform	206
6.1.3	Methoden und Vorgehen	206
6.2	Entwicklungs- und Laufzeitumgebung	207
6.2.1	Benötigte Software	207
6.2.2	Projekt-Setup	209
6.3	Phase OOA (objektorientierte Analyse: CIM und PIM)	214
6.3.1	Aktivität »CIM erstellen: Systemidee formulieren«	214
6.3.2	Aktivität »CIM erstellen: Fachklassen modellieren«	215
6.3.3	Aktivität »PIM erstellen: Geschäftsprozesse modellieren«	216
6.3.4	Aktivität »PIM erstellen: Geschäftsprozesse verfeinern«	220
6.4	Phase Plattformmodellierung	232
6.4.1	Bereitstellen der Mapping-Dateien	232

6.5	Phase OOD	233
6.5.1	Aktivität »PIM markieren«	234
6.5.2	Aktivität »PSM erstellen/Transformation durchführen«	238
6.6	Phase OOP	244
6.6.1	Aktivität »Code ergänzen«	245
6.6.2	Aktivität »Code testen«	248
6.7	Phase Testbetrieb	248
6.7.1	Aktivität »Build und Deployment«	248
6.7.2	Aktivität »Start der Applikation und testweise Benutzung«	251
6.8	Zusammenfassung	252
7	Anwendungsbeispiel Katalogsystem – Iteration 2	253
7.1	Einleitung	253
7.2	Projekt-Setup	253
7.3	Objektorientierte Analyse: CIM und PIM	256
7.3.1	Aktivität »PIM erstellen: Fachklassenmodell verfeinern«	256
7.4	Plattformmodellierung	260
7.4.1	Aktivität »Plattform modellieren: Beschreibung der Zielarchitektur«	261
7.4.2	Aktivität »Plattform modellieren: UML-Profil erstellen/erweitern«	268
7.5	Phase Generatorerstellung	270
7.5.1	Aktivität »Generatorerstellung: Projekt-Setup«	271
7.5.2	Aktivität »Generatorerstellung: Modellierung der Metafassaden«	271
7.5.3	Aktivität »Generatorerstellung: Generierung der Metafassaden«	275
7.5.4	Aktivität »Generatorerstellung: Implementierung der Metafassaden«	281
7.5.5	Aktivität Generator erstellen: Schreiben der Templates	285
7.5.6	Aktivität Generator erstellen: Konfiguration der Cartridge	292
7.5.7	Aktivität Generator erstellen: Build	297

7.6	Phase OOD	298
7.6.1	Aktivität OOD: PIM markieren	298
7.6.2	Aktivität »PSM erstellen: Transformation durchführen«	302
7.7	Phase OOP	304
7.7.1	Aktivität »PSM erstellen: Aktuelle Code-Artefakte identifizieren«	304
7.7.2	Aktivität »PSM erstellen: Ergänzungen durchführen – Sub-Viewpoint Structure«	304
7.7.3	Aktivität »PSM erstellen: Ergänzungen durchführen – Sub-Viewpoint Behavior«	307
7.8	Phase Testbetrieb	313
7.8.1	Aktivität »Build und Deployment«	313
7.9	Zusammenfassung	313
8	Anwendungsbeispiel Katalogsystem – Iteration 3	315
8.1	Entwicklungs- und Laufzeitumgebung	315
8.1.1	Projekt-Setup	315
8.2	Phase OOA	317
8.2.1	Aktivität »PIM erstellen: Persistenzschicht verfeinern«	317
8.3	Plattformmodellierung	318
8.3.1	Bereitstellen der Mapping-Dateien	318
8.4	Phase OOD	319
8.4.1	Aktivität »PIM markieren«	319
8.4.2	Aktivität »PSM erstellen/Transformation durchführen«	323
8.5	Phase OOP	328
8.5.1	Aktivität »Code ergänzen«	328
8.5.2	Aktivität »Code testen«	335
8.6	Phase Testbetrieb	336
8.6.1	Aktivität »Build und Deployment«	336
8.6.2	Aktivität »Start der Applikation und testweise Benutzung«	337
8.7	Zusammenfassung	338

9	Fazit und Ausblick	339
9.1	Die MDA: ein vorläufiges Fazit	339
9.2	Die MDA: Gegenwart und Zukunft	341
	Anhang	345
	Abkürzungsverzeichnis	347
	Glossar	349
	Quellenverzeichnis	355
	Literatur	355
	Richtlinien, Spezifikationen, Standards und Normen . . .	360
	Web-Links	362
	Stichwortverzeichnis	365