Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Siegfried Nolte

QVT - Relations Language

Modellierung mit der Query Views Transformation

Springer

Siegfried Nolte
Beethovenstr. 57
22941 Bargteheide
Deutschland
siegfried.nolte@alice-dsl.net

Xpert.press ISSN 1439-5428
ISBN 978-3-540-92170-7 e-ISBN 978-3-540-92171-4
DOI 10.1007/978-3-540-92171-4
Springer Dordrecht Heidelberg London New York

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Einbandentwurf: KuenkelLopka GmbH, Heidelberg

Gedruckt auf säurefreiem Papier

9 8 7 6 5 4 3 2 1

Springer ist Teil der Fachverlagsgruppe Springer Science+Business Media (www.springer.com)

Vorwort

Warum ein Buch über QVT Relations Language?

Dieses Buch beschäftigt sich mit der Transformation von Modellen. Dazu gibt es zurzeit bereits mehrere Mittel und Techniken, die jedoch vorwiegend die Transformation nach Text unterstützen, also die Generierung von Code-Artefakten aus formalen Modellen. Das ist schon etwas, schon ein Stück modellgetriebene Entwicklung. Die Transformation eines Modells, welches in einer grafischen Modellierungssprache erstellt worden ist, in ein anderes, ebenfalls grafisch repräsentiertes, ist bis heute noch die Ausnahme. Im Sinne des von der *Object Management Group* (OMG) veröffentlichten MDA-Konzeptes ist jedoch die Modelltransformation neben der Modellierung von Sachverhalten eine klassische Aktivität der modellgetriebenen Anwendungsentwicklung.

Modellierung erfolgt mit formalen Modellen, mit Modellen, die auf der Basis von Metamodellen formal spezifiziert worden sind – auch dies ein standardisiertes Konzept der OMG. Und die Modellierungssprache UML ist ein Klassiker, der heutzutage aus der Entwicklung von DV-gestützten Anwendungssystemen nicht mehr wegzudenken ist. Anmerken möchte ich an dieser Stelle, dass ich, wenn ich hier und im Weiteren von Standard rede, nicht Standard im Sinne der Normierung und Standardisierung durch ein nationales (DIN, ANSI) oder internationales Institut (ISO) meine. Standard heißt die Spezifikation und Veröffentlichung durch die *Object Management Group*, ein internationales Konsortium, das sich unter Beteiligung verschiedener IT-Unternehmen zusammengetan hat, um objektorientierte Technologien zu standardisieren. Das soll uns soweit reichen. Die Konzepte und Spezifikationen der OMG im MDA-Kontext stehen im Mittelpunkt dieses Buches.

Aber warum nun ein Buch über QVT? Mit Ausnahme der publizierten Spezifikation vom April 2008 gibt es bisher keins. Und die Spezifikation ist als Lehrmaterial meiner Meinung nach nicht brauchbar. Auch in den mir bekannten MDA-Büchern wird kaum oder gar nicht auf QVT eingegangen. Doch QVT ist ja genau das Mittel zur Transformation von Modellen, welches im Rahmen des MDA-

Konzeptes mit der MOF/QVT-Spezifikation Anfang 2008 als Standard herausgegeben worden ist. Diese Lücke will das vorliegende Fachbuch füllen.

Für wen ist das Buch gedacht?

Modellierung von Sachverhalten der realen Welt ist ein wesentlicher Bestandteil des ingenieursmäßigen Software-Entwicklungsprozesses. In der Szene der Entwicklungswerkzeuge ist die modellgetriebene Software-Entwicklung, die neben der Modellierung auch einen generativen Aspekt verfolgt, schon seit langem ein beliebtes Feld, das bereits mit Kompetenz und Erfahrung bedient wird. Mit Together 2006 zum Beispiel hat Borland ein Werkzeug auf den Markt gebracht, welches neben der Modellierung mit unterschiedlichen Modellierungssprachen – UML2, BPMN, SysML etc. – auch Modelltransformation mit QVT *Operational Mappings* unterstützt. Mit der Veröffentlichung des QVT-Standards, sogar schon mit der Vorankündigung, der *Final Adopted Specification*, Juli 2007, gibt es auch andere Hersteller, die QVT-Produkte entwickeln und anbieten. Eins davon, mediniQVT der ikv++ technologies AG, werde ich hier vorstellen und anwenden. Mit mediniQVT, ein Eclipse-basiertes Werkzeug zur Modelltransformation mit der *Relations Language*, sind nahezu alle Beispiele dieses Buches entwickelt worden.

Wie wir sehen, konzentrieren sich die Hersteller von Modellierungsprodukten zunehmend auf die QVT, was sicher auch darin begründet ist, dass sie zumeist ebenfalls in den Gremien der OMG vertreten sind. Ein weiterer interessanter Trend ist, dass das QVT-Konzept auch von der Eclipse *Modeling Tool* Initiative aufgegriffen worden ist und nun auch aus der Szene interessante Lösungen zu erwarten sind, zum Beispiel SmartQVT und QVTO, Transformationswerkzeuge, welche *Operational Mappings* unterstützen. Mit *Operational Mappings* und SmartQVT werde ich mich in einem weiteren Buch befassen.

Aber hier erst einmal die *Relations Language*. Dieses Fachbuch ist als Lehrbuch gedacht, mit dem die Sprache *Relations Language* erlernt und geübt werden kann. Ich habe versucht, das Thema anhand umfassender und ausführlicher Beispiele anschaulich und nachvollziehbar aufzuarbeiten. Grundkenntnisse der Modellierung mit UML, insbesondere im Zusammenhang mit Modellen und Metamodellen, setze ich dabei voraus, Erfahrungen mit höheren Programmiersprachen sind sicher hilfreich.

MDA beschreibt ein Konzept, in dem sich vorwiegend Analytiker und Anwendungsentwickler wiederfinden wie auch, gerade in dem Zweig der Architekturentwicklung, natürlich IT-Architekten. Diese Analytiker- und Entwicklerrollen möchte ich primär ansprechen. Gerade IT-Architekten sollen in die Lage versetzt werden, Metamodelle zu entwickeln und darauf aufbauend Modelltransformationen vorzunehmen. Aber es würde mich freuen, wenn auch Manager von IT-Organisationen wie auch Leiter von IT-Entwicklungsprojekten einige interessante Eindrücke und Erkenntnisse aus dem Buch entnehmen könnten, insbesondere was

die Einordnung der Modelltransformation in einen modellgetriebenen Entwicklungsprozess bedeutet.

Wie sollte man es lesen?

Das Hauptthema ist natürlich die *Relations Language*. Zuvor gebe ich in einem einführenden ersten Kapitel einen entwicklungsgeschichtlichen Einblick in das Thema der ingenieursmäßigen Software-Entwicklung, um darauf aufbauend eine Einordnung des modellgetriebenen Vorgehens in den systematischen Entwicklungsprozess zu begründen. Daran schließt eine Erläuterung der wesentlichen Begriffe und Akronyme des MDA-Konzeptes an. Im Weiteren erfolgt eine Erläuterung der Architektur des QVT-Ansatzes. Dieser beschreibt drei Transformationssprachen, *Core Language*, *Relations Language* und *Operational Mappings*, die ich kurz vorstellen und im Zusammenhang einordnen werde.

Das zweite Kapitel beschäftigt sich mit Metamodellierung. Wie die Modellierung mit formalen Modellierungssprachen, so ist auch der QVT-Ansatz zur Transformation grundsätzlich metamodellbasiert. Den Sachverhalt möchte ich verdeutlichen; an verhältnismäßig einfachen Beispielen, bei denen ich mich auf die Vorgabe der QVT-Spezifikation stütze, soll die Entwicklung von Metamodellen erläutert werden.

Im dritten Kapitel geht es um die *Relations Language*. Auch diese soll ausgehend von einem sehr einfachen Beispiel, bei dem ich das berühmte `HelloWorld` bemühen werde, erläutert werden. Danach stelle ich die Entwicklungsumgebung mediniQVT vor, mit der alle in diesem Buch vorgestellten Beispiele erarbeitet worden sind. mediniQVT ist ein freies Produkt, das die spezifizierte QVT *Relations Language* sehr konsequent unterstützt und damit für ein Üben mit der *Relations Language* recht hilfreich ist. In dem folgenden Kapitel gehe ich nun formal auf sämtliche Sprachkonzepte ein, um so die Sprache in ihrer Gänze vorzustellen.

Dem folgen mehrere vollständige Beispiele, die ich Schritt für Schritt entwickeln werde, um so die Erarbeitung von Transformationsscripten anhand eines exemplarischen Vorgang zu verdeutlichen, zunächst das Standardbeispiel der QVT-Spezifikation `UML2RDBM` mit den im zweiten Kapitel erarbeiteten einfachen Metamodellen `SimpleUML` und `SimpleRDBM` und anschließend ein umfassendes Projekt auf der Basis der UML2. Zuletzt werde ich einige fortgeschrittene Konzepte erläutern, zum Beispiel die Arbeit mit UML-Profilen, BlackBoxes etc.

Wem bin ich zu Dank verpflichtet?

Auf jeden Fall meiner Familie, zu allererst meinen Eltern Marie Louise und Gerhard, denen es gelungen ist, drei Kindern in einer früher schon wirtschaftlich schwierigen Zeit den Abschluss einer akademischen Ausbildung zu ermöglichen, womit letztendlich der Grundstein für dieses Buch gelegt worden ist. Dann den wichtigsten Frauen in meinem Leben, Konstanze und Christine, die während der Entstehung dieses Buches nicht nur auf einen Teil gemeinsamer Zeit verzichten,

sondern auch noch einen Teil ihrer Zeit aufbringen mussten, um sich durch frühe Fassungen des Buches hindurchzuarbeiten. Beide gehören nicht zu dem oben angeführten Leserkreis, doch ihre Kritik und Anregungen waren sehr hilfreich.

Als nächstes natürlich den Personen und Institutionen, die mich bei der Veröffentlichung des Buches unterstützten, zum einen dem Springer-Verlag und zum anderen Frau Monika Riepl von der Letex publishing services oHG. Der Springer-Verlag ist stets – so auch in diesem Fall – bereit, auch neue, manchmal unerfahrene Autoren bei der Veröffentlichung von Büchern zu unterstützen. Und Frau Riepl hat mir behutsam und mit Kompetenz geholfen, eine angemessene Form des Buches zu realisieren.

Und schließlich hat die ikv++ technologies AG mit der Herausgabe des Produktes mediniQVT und der Antwort auf viele Fragen im Forum dazu beigetragen, dass die Beschreibung der Sprachkonzepte mit vielen Beispielen unterlegt und untermauert werden konnte. Für die Beispiele im Kapitel Metamodelle ist das freie UML-Werkzeug TOPCASED eingesetzt worden. Fast alle Abbildungen sind UML-Diagramme, die mit dem Werkzeug MagicDraw der Firma nomagic erstellt worden sind.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Modellgetriebene Software-Entwicklung – die Geschichte

Die moderne Software-Entwicklung ist zunehmend mit der Aufgabe konfrontiert, immer kompliziertere Anforderungen aus der realen Welt mit immer komplexeren Anwendungssystemen und Software-Lösungen zu unterstützen. Der Entwickler steht dabei grundsätzlich vor zwei Problemen, zum einen dem Beherrschen seiner eigenen komplexen Technologien, zum anderen dem Verstehen der Sachverhalte, Strukturen und Gegebenheiten der realen Welt, mit der er es zu tun hat.

Um diesen grundlegenden Problemen zu begegnen, hat sich im Lauf der Zeit die Einsicht ergeben, den Software-Entwicklungsprozess aus einer Anwendungsprogrammierung herauszuheben und zunehmend nach ingenieursmäßigen Grundsätzen zu gestalten. Das führte in den späten 60er Jahren zu der Definition des Begriffs Software-Engineering [Bau68, Bau93]. Eines der wesentlichen Prinzipien des Software-Engineering ist, sich bei der Entwicklung von Software nicht unmittelbar mit der Programmierung zu beschäftigen, sondern die Sachverhalte und Strukturen der realen Welt über mehrere aufeinander aufbauende Abstraktionsebenen zu betrachten und darzustellen.

Den Ausschnitt der realen Welt, welcher der Gegenstand unserer jeweiligen Betrachtung ist, wollen wir im Folgenden als Fachdomäne bezeichnen. In einer solchen Fachdomäne existiert in der Regel eine Menge von

- Prozessen und Vorgängen, in denen in irgendeiner Weise produziert wird und betriebliche Werte, veräußerbare Produkte geschaffen werden,
- Gegenständen, die Objekte der Prozesse im Rahmen einer betrieblichen Wertschöpfung sind,
- Personen, die als Erfüllungssubjekte mit der Produktion in irgendeiner Weise beschäftigt sind,
- Regeln, Verfahren, räumliche und zeitliche Vorgaben und vieles mehr.

Die Welt des DV-Experten – DV-Welt – ist natürlich eine Domäne an sich mit ebenfalls recht komplizierten Sachverhalten und Gegebenheiten. Setzen wir einmal voraus, dass ein erfahrener Software-Entwickler sich in dieser Domäne auskennt und die Mittel zur Problemlösung beherrscht. Dann bleibt das zweite Grundproblem, die Fachdomäne, die eine DV-gestützte Lösung ihrer Probleme anfordert, zu verstehen, um sie in angemessener Weise mit einem zuverlässigen und leistungsfähigen DV-System zu versorgen. Die konservative Methode hierfür, die sich auch heute noch einer gewissen Beliebtheit erfreut, ist die VHIT-Methode, „Vom-Hirn-ins-Terminal“ (Abbildung 1.1).

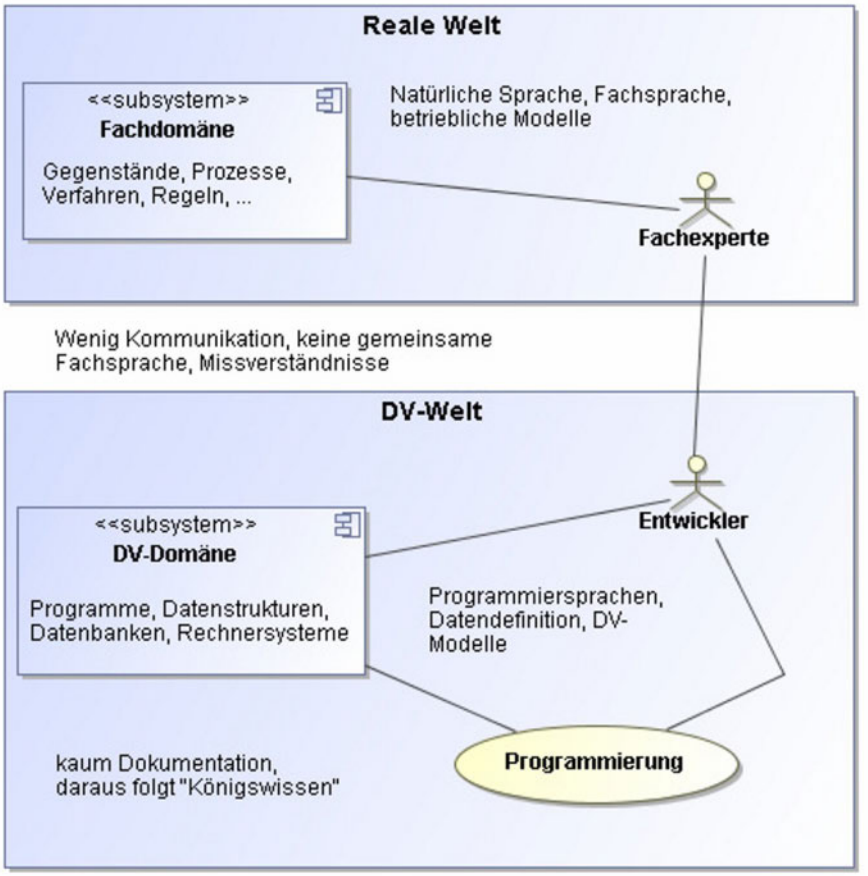

Abb. 1.1: SWE – die konservative Methode

Nehmen wir zum Beispiel die Fachwelt „Wohnungsbaukreditgeschäft“. In dieser Domäne bewegen sich Kreditsachbearbeiter, die Privatpersonen beraten mit der Zielsetzung, sie bei der Finanzierung von privaten Wohneigentumsvorhaben mit Darlehen zu unterstützen. Selbstverständlich ist diese Unterstützung nicht umsonst, sie kostet eine Leihgebühr, einen Zins. Aber das ist ein anderes Thema. Der SW-Entwickler beobachtet die Fachdomäne, erfragt fachliche Sachverhalte im Rahmen der Finanzierungsberatung der Darlehensvergabe, notiert die Erkenntnisse und setzt sie unmittelbar um.

Diese Herangehensweise ist prompt und schnell und kann bei wenig komplexen Anforderungen noch einigermaßen die gewünschten IT-Leistungen erbringen. Bei komplizierten betrieblichen Vorgängen, wie zum Beispiel der Sachbearbeitungsunterstützung bei Wohnungsbaufinanzierungen, reicht das alleinige Programmieren, also das unmittelbare Umsetzen einer Benutzeranforderung in Code, längst nicht mehr aus, um die fachlichen Anforderungen nachhaltig zu erkennen, zu verstehen und zu erfüllen. Hinzu kommt, dass die Fachleute in ihrer Domäne oft in einer eigenen Fachsprache untereinander kommunizieren, die von Außenstehenden wie z.B. DV-Experten nur schwer oder gar nicht verstanden wird.

Ein weiterer Aspekt ist, dass bei einer unmittelbaren Umsetzung eine spezielle Anforderung zwar schnell und spontan erfüllt sein mag, aber derartige Systeme in der Regel nur schwer zu bedienen und zu pflegen sind, insbesondere dann, wenn das „Königswissen“ des Spezialisten aus irgendeinem Grund verloren gegangen ist. Der erste Ansatz, die Kommunikation zwischen Fachexperten und Entwickler zu verbessern, besteht darin, in gemeinsamen Analysen die fachlichen Anforderungen herauszuarbeiten und in Konzepten festzuhalten (Abbildung 1.2).

Das heißt, die Fachexperten tragen Dokumente und Informationen zu der fachlichen Thematik zusammen, mit denen in oft gemeinsamer Arbeit Fachkonzepte geschrieben werden. Diese Fachkonzepte sind meist in Ermangelung einer gemeinsamen formalen Sprache in Prosa, also in Form von Freitext in natürlicher Sprache, erstellt. Damit gibt es zumindest schon dokumentierte Fakten und Forderungen. Doch die Inhalte sind immer noch unscharf formuliert, ungenau und teilweise missverständlich, insbesondere nachdem einige Zeit verstrichen ist, ohne dass die fachlichen Spezifikationen gepflegt worden sind. Eine ständige zeitnahe Überarbeitung solcher Konzepte und Dokumente findet in der Regel nicht statt.

Mittlerweile ist die gemeinsame Erarbeitung von Konzepten zwischen Fachexperten und IT-Experten aus der Anwendungsentwicklung nicht mehr wegzudenken. Es hat zudem einen intensiven Prozess gegeben, die Sprachen zwischen den Fachwelten durch Formalisierung immer weiter zusammenzubringen.

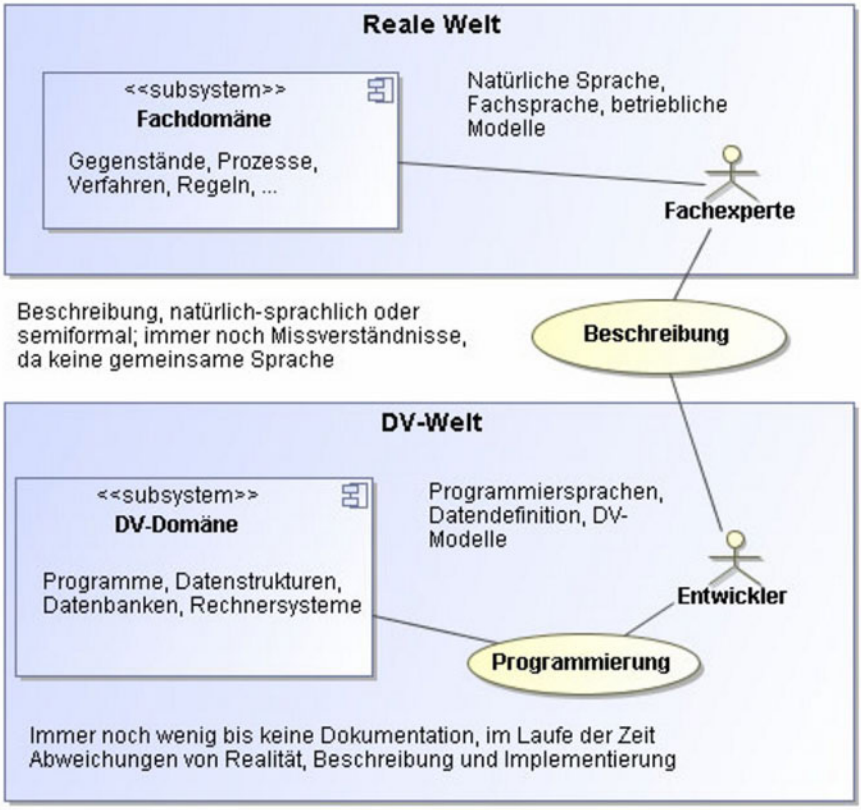

Abb. 1.2: SWE – erste Verbesserung durch Beschreibung

Daraus hat sich im Laufe der Zeit ein genereller modellbasierter Ansatz zur Lösung der Verständnisproblematik und für die Dokumentation der Erkenntnisse entwickelt (Abbildung 1.3). Dieser besteht aus dem Einsatz von zunächst diversen mehr oder weniger formalen, abstrakten Sprachen. Auch in der Fachdomäne selbst werden zunehmend formale Sprachen eingesetzt, um die fachlichen Gegebenheiten und Abläufe in strukturierter und im Fachbereich allgemein verständlicher Form abzubilden, zum Beispiel die Beschreibung von betrieblichen Dingen und Abläufen in Prozessketten [Oes95] oder unter Verwendung der aus der *Business Process Modeling Initiative* [BPMI] hervorgegangenen *Business Process Modeling Notation* [BPMN].

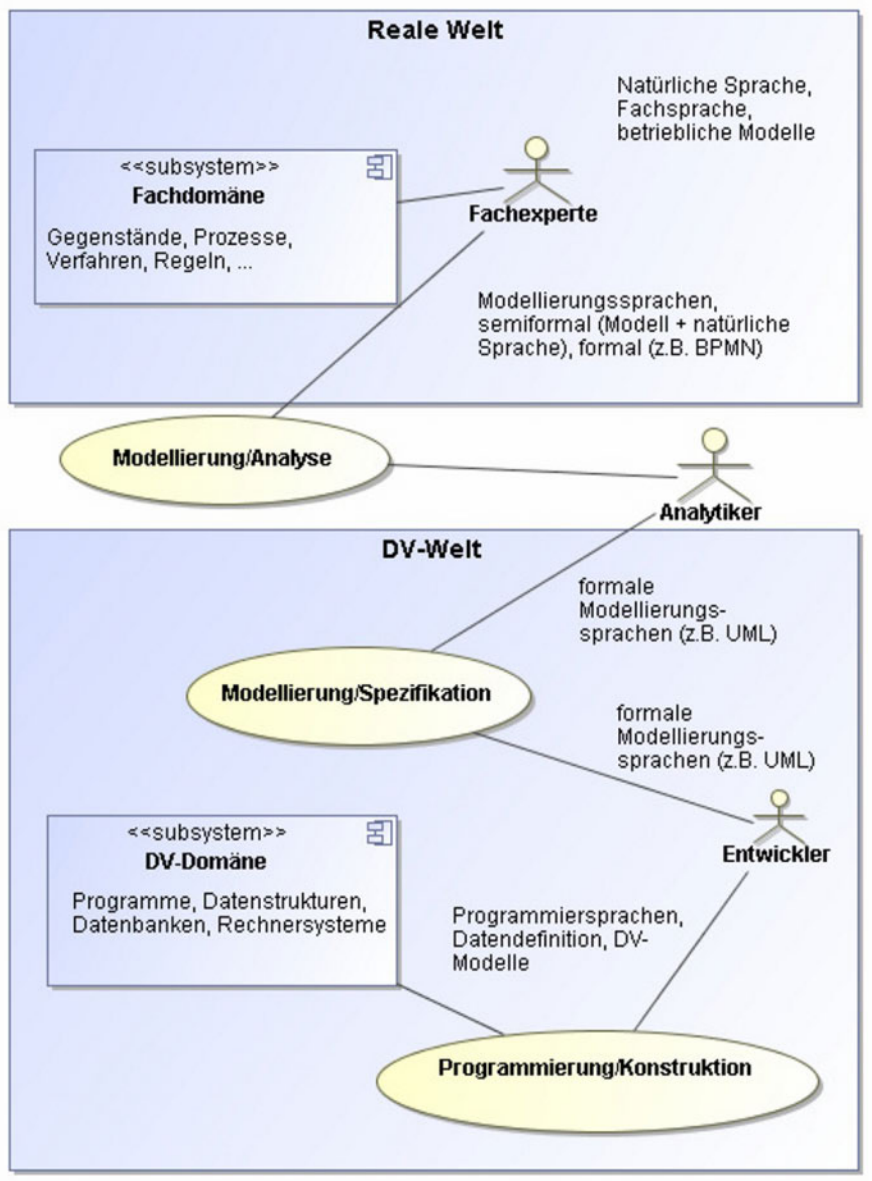

Abb. 1.3: SWE – Analyse, Spezifikation und Implementierung

So ist zu erwarten, dass zum Beispiel von dem Geschäftsprozess des Wohnungsbaukreditgeschäftes eine präzise Darstellung mit all seinen Vorgängen und Abläufen vorliegt. Diese formalen Sprachen sind von einem geschulten Analytiker erlernbar, die Modelle damit verstehbar und vermittelbar. Die weitere Entwick-

lung wird über mehrere Phasen durchgeführt, zum Beispiel Analyse, Konzeption, Konstruktion (Abbildung 1.4).

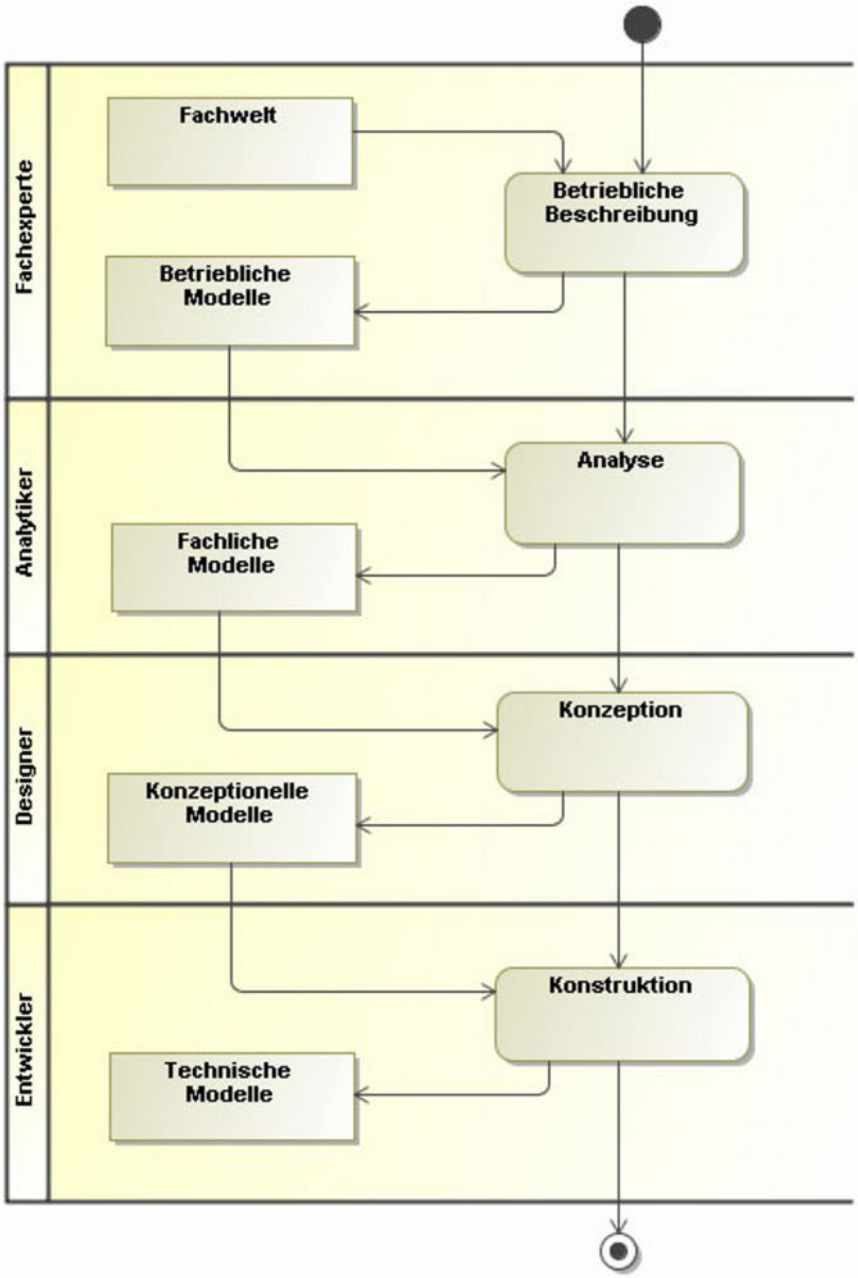

Abb. 1.4: Der klassische Anwendungsentwicklungsprozess

Der Weg von der Realität zur DV-Lösung wird als Abstraktion bezeichnet. Der Fachexperte arbeitet in seiner konkreten Domäne mit weitgehend eigenen domänenspezifischen Fachsprachen und gegebenenfalls auch speziellen formalen Sprachen, mit denen die Fachprozesse beschrieben werden können.

Im Rahmen einer Anwendungsentwicklung erfolgt zunächst eine Analyse der betrieblichen Sachverhalte, die vom Analytiker gemeinsam mit Beauftragten der Fachdomäne durchgeführt wird. Die Beschreibung ist bereits eine erste Spezifikation in einer formalen Form, die vom Analytiker in die DV-Welt hineingetragen werden kann. Auf der Basis der formalen Spezifikation, des Fachkonzeptes, können nun nach und nach weitere Entwicklungsarbeiten über mehrere Abstraktionsebenen bis hin zur Programmierung vorgenommen werden.

Die Ausdehnung der klassischen Software-Entwicklung zu einer modellgetriebenen Software-Entwicklung wollen wir uns nun im nächsten Kapitel im Detail ansehen. In jeder Phase befindet sich ein Experte, der die entsprechenden Fachsprachen seiner Ebene kennt und ein gewisses Grundlagenwissen über die jeweils andere Ebene besitzt. Der reale Sachverhalt und die Gegebenheiten der Fachdomäne werden mehrfach (semi-) formal beschrieben, sodass insgesamt eine verständliche und nachvollziehbare Dokumentation von dem entsteht, was letztendlich implementiert werden soll.

Das soll soweit zur Einstimmung reichen. Das Thema Software-Engineering und systematisches phasenorientiertes Vorgehen ist an anderer Stelle hinreichend beschrieben [Bal00, Boe76, Bow81] und auch in weiteren Büchern und Artikeln, die sich mit Vorgehensmodellen beschäftigen. Im Rahmen dieses Buches möchte ich mich nun mit der Beantwortung der Frage beschäftigen, wie die Phasenübergänge möglichst elegant zu bewerkstelligen sind, um so die Reibungsverluste, die sich im Laufe eines Software-Entwicklungsprozesses ergeben, möglichst gering zu halten. Dazu sollen Transformationstechniken dienen, hier insbesondere die der *Query Views Transformation* (QVT).

1.2 MDA – Model Driven Architecture

Ein Buch über QVT muss mit einem Beitrag über MDA (*Model Driven Architecture*) beginnen, auch wenn über MDA schon viel geschrieben worden ist [Gru06, Pet06, Kle03a, Mel04] und auch wenn über MDA noch mehr Meinungen existieren. MDA wird in diesem Buch sicher nicht im Vordergrund stehen, sondern eben QVT, der Vorschlag der OMG (*Object Management Group*) für die Transformation von Modellen. QVT wie auch MDA sind eng miteinander verbundene Konzepte der OMG. Das eine ist ohne das andere nicht denkbar. Doch an dieser Stelle soll lediglich eine Einblicknahme in die MDA erfolgen; ich werde dieses Thema nur an der Oberfläche berühren, tiefer gehende Erläuterungen finden sich in der angeführten Literatur. In diesem Zusammenhang möchte ich besonders auf die – zugegebenermaßen oft schwierig zu lesenden – Spezifikationen der OMG hinweisen. Ich werde mich in diesem Buch so konsequent wie möglich an den OMG-

Spezifikationen orientieren. Und QVT und MDA sind nicht alles; wir werden noch einiges mehr an „Drei-Buchstaben-Akronymen“ kennen lernen, die von der OMG in diesem Zusammenhang genannt werden, zum Beispiel UML (*Unified Modeling Language*), MOF (*Modeling Object Facility*).

Viele andere Werke greifen unter dem Titel MDA das Thema der modellgetriebenen Software-Entwicklung (MDSD) auf [Sta07], welches aber einer ganz anderen Quelle entspringt, der generativen Software-Entwicklung [Cza00]. Um es also gleich vorweg zu sagen, MDA ist modellgetriebene Architekturentwicklung, MDSD ist modellgetriebene Software-Entwicklung. Und das ist, obwohl in beiden Akronymen MD vorkommt, nicht dasselbe. MDSD stellt den Aspekt der ingenieursmäßig ausgeprägten Entwicklung von Software über eine modellgestützte Faktorisierung – gewissermaßen über den Einsatz von softwaretechnischen „Fertigungsstraßen“ – in den Vordergrund. Die Modellierung als intellektuelle Leistung durch Anwendung von Methoden und mit geeignetem Handwerkszeug ist eher nachrangig und dient im Wesentlichen der Vorarbeit für den Einsatz von Generator-Frameworks. Das Modell ist mehr oder weniger eine abstrakte Beschreibung als Eingabe für Generatoren, um damit automatisch und generativ Software-Komponenten zu produzieren. Bei der MDA dagegen befinden sich das Modell und die Arbeit an dem und mit dem Modell im Zentrum des Geschehens. Auf MDSD und die damit verbundenen Konzepte und Verfahren, die denen der MDA zum Teil durchaus ähnlich sind, werde ich nicht weiter eingehen. Hier sei insbesondere noch einmal auf [Sta07] verwiesen.

Das letztendliche Ziel ist natürlich grundsätzlich die Erstellung von Code in irgendeiner Form. Die Modellierung und Transformierung von Modellen soll nicht zum Selbstzweck verkommen, sondern am Ende auf der Basis von stabilen Architekturen qualitativ hochwertige Software-Artefakte liefern, zum Beispiel

- Java-Code aus einem *Java Metadata Interface* (JMI) Modell,
- eine Datenbank-Definition aus einem *Entity Relationship* Modell
- EJB-Deskriptoren aus einem EJB-Modell.

Zurzeit wird in der OMG auf der Basis des MOF-Konzeptes an der Spezifikation einer M2T-Templatesprache [M2T] gearbeitet.

1.2.1 Definition der Model Driven Architecture

MDA bedeutet in erster Linie die systematische Entwicklung von stabilen, tragfähigen IT-Architekturen durch Modellierung unter Anwendung von formalen Modellierungssprachen. Die Modellierung erfolgt über mehrere Abstraktionsebenen eines inkrementellen Entwicklungsprozesses, zum Beispiel aus einer fachlichen Ebene bis hin zu einer Implementierungsebene. In jeder Ebene werden die Gegenstände und die Prozesse an ihnen mit geeigneten Mitteln in formalen Modellen dargestellt.

Mit Hilfe von Transformationen werden die in Modellen beschriebenen Sachverhalte aus einer Entwicklungsebene in Modelle einer anderen Entwicklungsebene überführt. Dazu werden formale Architekturschemata eingesetzt, in denen die Transformationsanweisungen spezifiziert sind. Die Abbildung 1.5 zeigt exemplarisch den Anwendungsentwicklungsprozess unter Berücksichtigung von MDA-Aspekten.

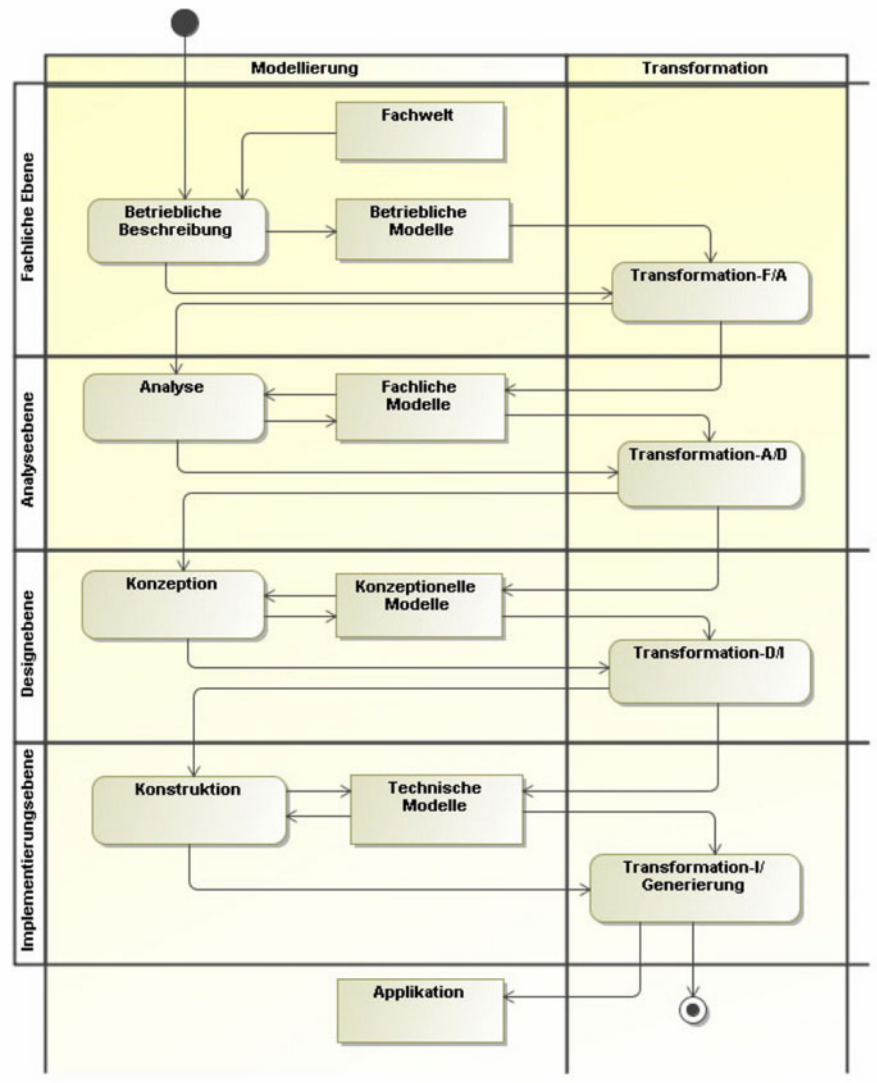

Abb. 1.5: Ein modellgetriebener Anwendungsentwicklungsprozess

Der MDA-Entwicklungsprozess ist demnach eine iterierende Abfolge von Modellierung und Transformation. In den Entwicklungsebenen findet Modellierung statt. Der Übergang von einer Entwicklungsebene zu der folgenden wird jeweils durch eine Transformation der Modelle unterstützt.

Durch Transformation wird nicht nur eine Überführung von Modellen vorgenommen, sondern eine Erzeugung von gültigen Modellen im Sinne einer formalen Modellierung. Ein willkommener Nebeneffekt eines MDA-orientierten Vorgehens ist es demnach, dass bei jedem Phasenübergang, von einer Entwicklungsebene zu einer anderen, unter formalen Gesichtspunkten gültige Ausgangsmodelle übergeben werden. In gewisser Weise kann auch die semantische, sachlogische Validität von Modellen kontrolliert und gewahrt werden, nämlich durch die Ergänzung von formalen Gültigkeitsbedingungen zum Beispiel mit der *Object Constraint Language* (OCL).

Bei der MDA haben das Modell und die Modellierung einen höheren Stellenwert als die Generierung. Im Rahmen einer Architekturentwicklung nach diesem Verständnis ist die Generierung von Software durch Transformation eines Modells auf der Implementierungsebene der letzte Schritt. Selbstverständlich ist MDA nicht Selbstzweck, sondern sie dient dem Zweck, stabile und anforderungsgerechte Softwaresysteme zu entwickeln. Die „Modell-nach-Modell"-Transformation, die Beschreibung der Architekturschemata mit der QVT und Durchführung der Modelltransformationen, wird im Zentrum dieses Buches stehen.

Die drei Grundziele der modellgetriebenen Architekturentwicklung [MDA03] sind:

1. Portabilität, die größtmögliche Unabhängigkeit eines Systems von möglichen Betriebsplattformen,
2. Interoperabilität, die Fähigkeit eines Systems, möglichst nahtlos mit anderen Systemen zusammenzuwirken,
3. Wiederverwendbarkeit, das Qualitätsmerkmal eines Systems, möglichst umfassend in möglichst vielen unterschiedlichen Kontexten verwendet werden zu können.

Der MDA-Ansatz versucht diese grundlegenden Ziele zu erreichen, indem eine konsequente Trennung der Spezifikation eines Systems von dessen Implementierung auf einer speziellen Plattform vorgenommen wird. MDA besteht einerseits aus einer von den Gegebenheiten einer bestimmten Plattform losgelösten Spezifikation eines Systems, Plattformunabhängigkeit (*platform independent modeling*). Auf der anderen Seite erfolgt eine Überführung der Spezifikation auf eine beliebige zugrundeliegende Plattform, wo eine weitere Modellierung unter plattformspezifischen Gesichtspunkten (*platform specific modeling*) erfolgt. Zuletzt wird eine Überführung des plattformspezifischen Modells in die Implementierungsschicht vorgenommen (*implementation modeling*). Man kann bereits erkennen, dass MDA im Sinne der Architekturentwicklung auch ein spezielles Vorgehen beschreibt, so dass es naheliegt, das MDA-Schichtenmodell in einem Vorgehensmodell zu integrieren. Ansatzweise ist das bereits in Abbildung 1.5 zu sehen. Weitere Überlegungen dazu finden sich in [Pet06].