

Thema:

Modellgetriebene Entwicklung von betrieblichen Informationssystemen

Ausarbeitung

im Rahmen des Seminars Warenwirtschaftssysteme

im Fachgebiet Betriebswirtschaftslehre

am Lehrstuhl für Wirtschaftsinformatik und Informationsmanagement

vorgelegt von: Ulrich Wolfgang

Abgabetermin: 2006-12-06

Inhaltsverzeichnis

Inhaltsverzeichnis	II
Abkürzungsverzeichnis	III
1 Modellgetriebene Softwareentwicklung	1
2 Ansätze.....	2
2.1 Model Driven Development	2
2.1.1 Nutzbarkeit von Unternehmensmodellen.....	4
2.1.2 Meta- und Metametamodellierung.....	5
2.1.3 Transformation	6
2.1.4 Vorteile des Model Driven Development	10
2.1.5 Werkzeuge.....	11
2.2 Model Driven Architecture.....	11
2.2.1 Standards der OMG.....	13
2.2.2 UML als Sprache für die MDA.....	13
2.2.3 Meta-Object Facility	17
2.2.4 Nutzen der Metamodellierung für die MDA.....	18
2.3 Universal Application.....	18
3 Vergleich und Bewertung der Ansätze	20
3.1 Model Driven Architecture und Computer-Aided Software Engineering.....	20
3.2 Model Driven Development und Model Driven Architecture.....	21
3.3 Model Driven Development und Universal Application.....	21
4 Ausblick.....	23
Literaturverzeichnis.....	24

Abkürzungsverzeichnis

3GL	Third generation language
AS	Action Semantics
CASE	Computer-Aided Software Engeneering
CIM	Computation Independent Model
CORBA	Common Object Request Broker Architecture
CWM	Common Warehouse Metamodel
DSL	Domain Specific Language
EAI	Enterprise Application Integration
EDOC	Enterprise Distributed Object Computing
Java EE	Java Platform, Enterprise Edition
MDA	Model Driven Architecture
MDD	Model Driven Development
MDSO	Model Driven Software Development
MOF	Meta-Object Facility
MOF-QVT	Meta-Object Facility Queries/Views/Transformations
OAW	OpenArchitectureWare
OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
UA	Universal Application
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XUML	Executable Unified Modeling Language

1 Modellgetriebene Softwareentwicklung

Die modellgetriebene Softwareentwicklung ist ein Vorgehensmodell für die Entwicklung von Software. Mit der Nutzung von Modellen als durchgängige Beschreibung von Softwaresystemen stellt sie eine Innovation in der Folge der Paradigmen der prozeduralen Programmierung, der objektorientierten Programmierung und der komponentenorientierten Programmierung dar. Mit jeder Innovationsphase der Paradigmen ist eine zunehmende Abstraktion von der zugrunde liegenden Computerarchitektur verbunden, deren Grad auch in der modellgetriebenen Softwareentwicklung mit dem Modell als zentralem Entwicklungsgegenstand angehoben wird.

Die Vision der modellgetriebenen Softwareentwicklung ist die Trennung der Implementierungskonzepte von den technologieunabhängigen Konzepten zur teilautomatisierten Erstellung von Software. Dies kann im Fall des Model Driven Development (MDD) und der Model Driven Architecture (MDA) mit der Absicht geschehen, die Implementierung aus Geschäftsmodellen zu gewinnen. Einen alternativen Ansatz stellt die Universal Application (UA) dar, bei der Modelle als Konfigurationsparameter einer generischen Standardsoftware aufgefasst werden.

Die Seminararbeit erläutert im Einzelnen die Ansätze des Model Driven Development, der Model Driven Architecture in ihren verschiedenen Subkategorien und der Universal Application und vergleicht sie anschließend kritisch miteinander in ihren unterschiedlichen Ausrichtungen.

2 Ansätze

2.1 Model Driven Development

Das Model Driven Development (MDD), auch Model Driven Software Development (MDSO) genannt, ist ein Vorgehensmodell für die Entwicklung von Software. Zentral für den Entwicklungsprozess sind die Architektur und Modelle der zu erstellenden Software, also die fachliche Ebene. Die mit dem MDD verwandte Model Driven Architecture (MDA) stellt eine Standardisierung der MDD durch die Object Management Group (OMG) dar und wird in einem eigenen Kapitel gesondert beschrieben.

Die Erstellung von Software im MDD lässt sich nach ihrem Abstraktionsgrad geordnet nach der Geschäftsprozessmodellebene, der Ebene der fachlichen Modelle, der Ebene der technischen Modelle und abschließend der Programmiersprachenebene trennen. Das Ziel des MDD ist die automatisierte Transformation der fachlichen Modelle über die technische Ebene auf die Programmiersprachenebene, forward engineering genannt. Dies steht im Gegensatz zum reverse engineering, bei dem aus der Implementierung Quellcode, fachliche Modelle und Produktdefinitionen gewonnen werden.

Auf der Modellebene wird zwischen dem fachlichen Platform Independent Model (PIM) und dem technischen Platform Specific Model (PSM) unterschieden. Das PIM abstrahiert von technologischen Details, während das PSM die Konzepte einer Plattform verwendet, um ein System zu beschreiben.¹ Eine Plattform ist eine Menge von Subsystemen und Technologien, die eine Menge zusammenhängender Funktionalität durch Schnittstellen und spezifische Nutzungsmuster bereitstellt, die durch die Plattform unterstützte Applikationen nutzen können. Die Applikationen müssen dazu keine detaillierten Informationen über die Implementierung der Funktionalität der Plattform besitzen. Beispiele für Plattformen sind Java EE, Microsoft .NET und Intel x86.²

Das PSM wird aus dem PIM mittels eines Transformators gewonnen. Aus dem PSM wird mittels eines Transformators Programmcode generiert. So kann aus einem Modell mit einem hohen Abstraktionsgrad eine implementierungsnahe Umsetzung gewonnen werden. Transformationen sind auch von PIM zu PIM, also auf derselben Abstraktionsebene, möglich. Dies kann bei einer Transformation zwischen Modellen derselben Sprache, z.B. bei der Normalisierung eines ERM oder dem Coderefactoring von C#-Quellcode, genutzt werden. Eine Transformation ist die automatische Generierung eines Zielmodells aus einem

¹ Vgl. Rossbach, Stahl, Neuhaus (2003), S. 3.

² Vgl. Miller, Mukerji (2003), S. 13.

Quellmodell, entsprechend einer Transformationsdefinition.³ Die Transformationsdefinition besteht aus Transformationsregeln, die beschreiben, wie ein Modell in einer Quellsprache in ein Modell in einer Zielsprache übersetzt werden kann.

Die Transformatoren setzen eine formale Semantik der Modelle voraus, die durch Meta-Modelle sichergestellt wird, in denen die Sprache der Modellebene definiert wird. Dabei ist eine Definition der UML für objektorientierte Programmierung im Allgemeinen denkbar, möglich ist aber auch die Definition von Domain Specific Languages (DSL). Da domänen-spezifische Sprachen bei der Anwendung in ihrer Domäne sehr mächtig sein können, außerhalb ihrer Domäne aber oft versagen, ist die Unterstützung verschiedener DSL innerhalb eines MDD-Projektes sinnvoll. Durch die Anpassung der Transformatoren an verschiedene DSL können mehrere DSL in einem Projekt genutzt werden, und die darin erstellten Modelle in eine Endimplementierung transformiert werden.

Der der MDD zugrunde liegende Prozess lässt sich grob in die folgenden Schritte untergliedern:⁴

- Es muss eine klare Definition und Abgrenzung der Domäne für das Zielsystem erstellt werden, damit der Umfang der Modellierungskonstrukte begrenzt ist und die Semantik der Modellierungssprache eindeutig ist.
- Mit Hinblick auf die technische Ebene muss eine passende Architektur entworfen werden. Dazu müssen Architekturmuster, Basistechnologien, Frameworks und Fremdkomponenten ausgewählt werden.
- Die Architektur sollte mittels einer Referenzimplementierung für ausgewählte Anwendungsfälle aus der Domäne validiert werden, um sie als adäquat für die Lösung einstufen zu können.
- Zu der Modellierung müssen für die Zieldomäne und Architektur passende Modellierungssprachen entworfen oder selektiert werden. Die Auswahl der Unified Modelling Language (UML) ist oftmals aufgrund des sehr hohen Verbreitungsgrades möglich und wünschenswert, alternativ können aber Eigenentwürfe erfolgen oder bereits existierende domänenspezifische Sprachen gewählt werden.
- Um die Transformation durchführen zu können, müssen Transformationsregeln definiert werden und Transformatoren erstellt oder selektiert werden.

³ Vgl. Kleppe, Warmer, Bast (2003), S. 23 f.

⁴ Vgl. Ehlert (2005), S. 1.

- Zur Erstellung der Modelle muss ein Modellierungswerkzeug gewählt werden. Eine Eigenerstellung des Werkzeuges ist ökonomisch nur in Spezialfällen sinnvoll, da bereits einige Produkte verfügbar und erprobt sind.
- Mittels des Modellierungswerkzeuges wird die Modellerstellung vorgenommen. Um das Modell in Quellcode überführen zu können, ist die Generierung der Implementierungsartefakte für die Zielsprache notwendig. Diese werden durch eine Transformationsdefinition in eine Relation zu den Modellkonzepten und deren formaler Semantik gesetzt.

2.1.1 Nutzbarkeit von Unternehmensmodellen

Geschäftsprozessmodelle dienen der vereinfachten Abbildung von Geschäftsprozessen zu einem bestimmten Zweck. Geschäftsprozesse bestehen aus einer zeit- und sachlogischen Abfolge von Einzelfunktionen, die in Geschäftsanwendungsfällen geschäftliche Abläufe beschreiben. Sie lassen sich in Ablaufmodellen in Form von Aktivitätsdiagrammen darstellen.

Zur Nutzung durch das MDD muss aus einem Geschäftsprozessmodell der für das System relevante Part in ein Systemanalysemodell extrahiert werden. Dieses enthält in Systemanwendungsfällen nur die Einzelschritte der Geschäftsanwendungsfälle, die von dem zu entwickelnden System ausgeführt werden sollen.

Das Systemanalysemodell wird anschließend in ein PIM transformiert. Um die Transformation zu vereinfachen, ist es sinnvoll, schon das Systemanalysemodell in der Modellierungssprache UML abzufassen.⁵ Die formale Fundiertheit und Transformierbarkeit der Sprache des Systemanalysemodells ist wichtig, da dies eine manuelle oder teilautomatisierte Transformation in das PIM mit den gewünschten Eigenschaften der Flexibilität, Verfolgbarkeit, inkrementellen Konsistenz und Bidirektionalität erleichtert.

Strukturelle Aspekte von Geschäftsmodellen lassen sich im Vergleich zu dynamischen Prozessaspekten relativ einfach in ein PIM transformieren, da sie einen Zustand ohne zeitliche und anderweitige konditionale Gültigkeitsaspekte beschreiben. Die Strukturkonzepte in den Modellen entsprechen weitgehend der Umsetzung auf der Implementierungsebene. Da die dynamischen Aspekte durch ihre sachlogische Darstellungsweise nicht dem reaktiven ereignisorientierten Schema auf der Implementierungsebene entsprechen, muss eine zusätzliche Verbindung zwischen diesen beiden Ebenen geschaffen werden. Dazu werden die Systemanwendungsfälle des Systemanalysemodells mit Zustandsdiagrammen versehen

⁵ Vgl. Weilkiens, Oestereich, Stahl (2003), S. 9.

und so mit Informationen für eine Transformation angereichert. Die Zustandsdiagramme enthalten Ereignis-Reaktions-Kombinationen und beziehen sich inhaltlich jeweils auf einen Systemanwendungsfall, der ihren Kontext darstellt. Bei z. B. der Verwendung von UML für die statischen Aspekte des Systemanalysemodells werden die Zustände mit UML-Stereotypen versehen, um eine Zuordnung der dynamischen zu den statischen Aspekten herzustellen. Diese Zuordnung wird anschließend von einem Transformator interpretiert und über das PIM und PSM in Quellcode übertragen.⁶

Durch diese Methodik ist eine Nutzung von Geschäftsprozessmodellen für das MDD möglich und erschließt auch diese als eine Ressource für die Entwicklung von Anwendungssystemen.

2.1.2 Meta- und Metametamodellierung

Die MDD bzw. MDA lässt sich in 4 Ebenen untergliedern, die aufeinander aufbauen:⁷

Ebene M0: Die Ebene M0 ist die Instanzebene. Sie repräsentiert das laufende System mit seinen aktuellen Instanzen wie z.B. Objekten in einem objektorientierten Programm oder Datensätzen in einem relationalen Schema. Dabei muss unterschieden werden zwischen den modellierten Sachverhalten in Geschäftsmodellen und in Computermodellen: Geschäftsmodelle beschreiben Realweltobjekte, also z. B. eine Rechnung als ein einen Prozess prägendes Objekt, Computermodelle dagegen beschreiben Repräsentationen der Realweltobjekte, also z. B. den Datensatz, der die Rechnung repräsentiert.

Ebene M1: Die Ebene M1 stellt die Abstraktion der Ebene M0 dar. Dabei wird eine Abbildung der Realität durch Aggregation der relevanten Eigenschaften der Instanzen zu einem bestimmten Zweck erstellt. Die Elemente des Abbilds können als Konzepte verstanden werden, die untereinander in Beziehung gesetzt sind. Sie werden durch Attribute in ihren Eigenschaften beschrieben. Auf der Ebene M0 werden die Attribute im Rahmen ihres Typs ausgeprägt. Die Attribute können in den Werten ihrer Ausprägungen durch Constraints beschränkt werden.

Zwischen den Instanzen auf der Ebene M0 und Modellelementen auf der Ebene M1 besteht eine Beziehung, durch die sich die Instanzen klassifizieren lassen. Instanzen müssen durch mindestens ein Modellelement klassifiziert werden können, um im Sinne des Modells zulässig zu sein. Im Rahmen der MDA können Modelle im herstellerunabhängigen Format XML Metadata Interchange (XMI) gespeichert werden.

⁶ Vgl. Weilkiens, Oestereich, Stahl (2003), S. 11.

⁷ Vgl. Frankel (2003), S. 105 ff.

Ebene M2: Die Ebene M2 stellt das Modell eines Modells, also das Metamodell, dar. Dabei werden die als Instanzen aufgefassten Modellelemente des M1 als Konzepte des M2 wiedergegeben. Analog zu der Beziehung von Ebene M0 zu Ebene M1 drücken Instanz-Konzept-Beziehungen zwischen M1 und M2 aus, ob das Modell M1 im Sinne von M2 zulässig ist.

Mit den auf der Ebene M2 spezifizierten Modellierungssprachen können syntaktisch und formal semantisch korrekte Modelle erstellt werden. Z. B. hat die OMG die Sprache UML in einem Metamodell spezifiziert. Die Modellelemente eines UML-Metamodells auf der Ebene M2 sind unter anderem die UML Class und das UML Attribute.

Ebene M3: Ein Modell der Ebene M3 ist das Metametamodell der Instanzen der Ebene M0. Analog zu den Beziehungen zwischen Elementen der Ebene M1 und Ebene M2 sind die Beziehungen zwischen Elementen der Ebene M2 und M3 vorhanden. Somit spezifiziert ein Modell der Ebene M3 eine Sprache, mit der Sprachen wie UML spezifiziert werden können. Dies stellt verschiedene Modellierungssprachen auf eine einheitliche Basis, was für die MDD-Frameworks und die MDA die Grundlage der Transformation zwischen in verschiedenen Modellierungssprachen erstellten Modellen darstellt. Die OMG sieht auf der Ebene M3 die Meta-Object Facility (MOF) vor, in der zur MDD gehörenden OpenArchitectureWare (OAW) dagegen werden Ecore, Javabeans und EMOF, eine Untermenge von MOF, genutzt.

In Anwendungsprogrammen wird die theoretische Trennung von Instanzen und Metadaten aber nicht immer vollständig vollzogen. Viele Computersysteme wie z. B. Datenbanksysteme speichern zugleich Daten und Metadaten an dem physikalisch selben Ort. Auch in Modellierungsumgebungen werden häufig Daten der Ebene M1 zusammen mit Daten der Ebene M2 vorgehalten, z.B. um performant Syntax- und formale Semantikchecks auf Modellen durchführen zu können

2.1.3 Transformation

In der MDD bzw. MDA werden Modelle gleicher und verschiedener Abstraktionsebenen ineinander überführt, um aus abstrakten Modellen über Zwischenschritte ausführbaren Programmcode zu erzeugen. Die Transformation ist ein Prozess, bei dem ein Modell aus einem anderen generiert wird. Der Transformationsprozess wird beschrieben durch eine Transformationsdefinition, die aus Transformationsregeln besteht, und der mittels eines

Transformationswerkzeuges durchgeführt wird.⁸ Im Zuge der Transformation muss die Bedeutung des Quellmodells im Zielmodell erhalten bleiben.

Dazu wird jedes Quellmodellelement mit mindestens einem Zielmodellelement assoziiert. Dies kann zum einen durch die Verbindung der Instanzen des Quell- und Zielmodells auf der Ebene M1 geschehen, ist aber sehr aufwendig, da es zu einer individuellen Zuordnung vieler Modellelemente führt. Die Transformationsdefinitionen großer Modelle sind damit nicht mehr beherrschbar. Alternativ werden die Elemente des Quell- und Zielmetamodells auf der Ebene M2 miteinander assoziiert.⁹ Über die Instanztypbeziehung zwischen den Modellen auf der Ebene M1 und den Metamodellen auf der Ebene M2 lassen sich abschließend die Modellelemente der Ebene M1 den Transformationsregeln zuordnen. Aufgrund der besseren Handhabbarkeit ist dieser Ansatz in der MDD und MDA der übliche.

In der MDA werden Transformationsdefinitionen durch das Sprachenbündel Queries, Views, and Transformations (MOF-QVT) standardisiert, in der MDD wird z. B. in der OpenArchitectureWare dazu der Standard Xtend genutzt.

Unabhängig von gegebenen Standards sind einige Eigenschaften in Bezug auf Transformationen erwünscht:

Flexibilität, Anpassbarkeit

Die Anpassbarkeit von Transformationsregeln soll die Verwendbarkeit in diversen sprachlichen Kontexten ermöglichen. So soll die Spezialisierung von Regeln möglich sein, z.B. ist es erforderlich, dass Eigenschaften der Regeln wie die Länge eines Varchar in einem ERM-Diagramm im Gegensatz zu einem Varchar in einem UML-Diagramm unterschiedlich ausgeprägt sein können. Die Spezialisierung ermöglicht die Anpassung einer generalisierten Transformationsregel für den entsprechenden Kontext.

Die vom Modellierer gewünschte Kontrolle über den Transformationsprozess setzt eine Kontrolle über die Details der Transformationsdefinition voraus, da die Entscheidungen des Transformators auf Basis der Metamodellelemente nicht immer problemadäquat sind. Zur Lösung des Problems sind drei Ansätze denkbar:¹⁰

1. *Manuelle Kontrolle:* Auf der Modellebene M1 wird einzelnen Modellelementen wie z. B. Klassen und Attributen zugeordnet, mit welcher Transformationsregel sie

⁸ Vgl. Kleppe, Warmer, Bast (2003), S. 73.

⁹ Vgl. Mellor, Scott, Uhl, Weise (2004), S. 51.

¹⁰ Vgl. Kleppe, Warmer, Bast (2003), S. 74.

transformiert werden sollen. Dieser Ansatz ist bei Modellen mit einer großen Anzahl an Modellelementen sehr aufwendig und wartungsintensiv.

2. *Konditionsbasiert*: Den Transformationsregeln werden Konditionen zugeordnet, die den notwendigen Kontext für die Gültigkeit der Regel festlegen. Die Grundlage einer Kondition kann jeder beliebige Bestandteil des Quellmodells sein, wie z.B. ein Teilstring von Attributbezeichnern, Klassennamen oder Stereotypen. Die Automatisierung der Transformation wird stark vereinfacht, wenn die Konditionen disjunkt sind, da durch diese Prämisse Kollisionen vermieden werden, und somit auf eine Kollisionsbehandlung verzichtet werden kann.
3. *Transformationsparameter*: Eine Parametrisierung der Transformationsdefinitionen ermöglicht die Konfiguration der Regeln. In den Regeln werden anstatt von konstanten Werten und Bezeichnern Variablen eingesetzt, die vor der Transformation ausgeprägt werden müssen. Variablen können z.B. Längen von Strings oder Präfixe in Namen von Klassenmethoden sein.

Die Ansätze 2 und 3 setzen voraus, dass die Entscheidungsgrundlage im Quellmodell expliziert ist. Falls dem nicht so ist, muss Ansatz 1 gewählt werden. Die Entscheidungsgrundlage ist dann das implizite Wissen des Modellierers über die Inhalte der abzubildenden Domäne.

Nachverfolgbarkeit

Die Zuordnung der Elemente in Quell- und Zielmodell muss bidirektional navigierbar sein, um Änderungen an Quell- oder Zielmodell an das jeweils andere Modell weitergeben zu können. Wenn generierte Elemente im Zielmodell wie z.B. dem PSM geändert werden, um sie zu verfeinern, kann ein Konflikt bei dem nächsten Transformationsvorgang aus dem PIM auftreten, da die Änderung des PSM nicht an das PIM weitergegeben worden ist. Eine Mindestanforderung in dieser Konfliktsituation ist eine Warnung und Nennung der Konfliktstelle. Wünschenswert ist ein Änderungsvorschlag im PIM durch den Transformator. Falls der Zusammenhang zwischen PIM und PSM eindeutig definiert ist und eine Problemlösung durch das System bestimmbar ist, sieht eine optimale Unterstützung des Modellierers die automatisierte Umsetzung des Vorschlages durch die Modellierungsumgebung vor.¹¹

Auch können im Laufe eines Projektes hinzukommende Modifikationen des Programmcodes auf diese Weise besser auf der Ebene des PIM nachgebildet werden, wenn der Zu-

¹¹ Vgl. Kleppe, Warmer, Bast (2003), S. 75 f.

sammenhang des Codes über das PSM mit der im PIM definierten Funktionalität eindeutig dokumentiert ist.

Inkrementelle Konsistenz

Zum Zielmodell hinzugefügte spezifische Informationen sollen auch nach einer erneuten Transformation erhalten bleiben. Somit müssen Änderungen an Elementen des Quellmodells und Elementen des Zielmodells kombiniert werden, indem das Quellmodell einen minimalen nötigen Einfluss auf das Zielmodell nimmt.¹² Dies ist zu unterscheiden von der Eigenschaft der Verfolgbarkeit, bei der Änderungen am Zielmodell einer Transformation auf das Quellmodell rückwirken sollen. Inkrementelle Konsistenz geht von der Bedeutungsdivergenz des Quell- und Zielmodells aus, und hat als Ziel, Bedeutungsmodifikationen des Quellmodells mit denen des Zielmodells zu kombinieren.

Bidirektionalität

Transformationen sollen in beide Richtungen durchgeführt werden können. Dies kann mittels einer einzelnen Transformationsdefinition für beide Richtungen erfolgen oder mit zwei jeweils individuellen Transformationsdefinitionen, die einander entgegengerichtet sind.¹³

Eine einzelne Transformationsdefinition für beide Richtungen bringt Probleme durch surjektive und injektive Abbildungen der Menge der Quellmodellelemente auf die Menge der Zielmodellelemente mit sich. Wenn z.B. ein Zustand auf einen Boolean abgebildet wird, kann dies nicht umgekehrt werden, da dies eine Basis für die Induktion eines Konkretums auf ein Abstraktum voraussetzt. Die Transformation ist nur dann invertierbar, wenn eine bijektive Abbildung der Modellelemente vorliegt.

Zwei entgegengesetzte Transformationsdefinitionen sind problematisch, da nicht beweisbar ist, ob sie wirklich zueinander invers sind. Auch hierbei kann die Bidirektionalität nur erreicht werden, wenn die Abbildung der Modellelemente rein bijektiv ist.

Eine Voraussetzung für die Bijektivität ist, dass der Abstraktions- und Ausdrucksgrad beider Modellsprachen äquivalent ist, da die formale Semantik beider Sprachen aufeinander abbildbar sein muss. Dies reduziert den Ansatz auf Bidirektionalität zwischen PIM und zwischen PSM, da das PIM durch die Konzeption der MDD einen höheren Abstraktionsgrad als das PSM hat.

¹² Vgl. Kleppe, Warmer, Bast (2003), S. 76 f.

¹³ Vgl. Kleppe, Warmer, Bast (2003), S. 77.

2.1.4 Vorteile des Model Driven Development

Durch die Automatisierung des Implementierungsvorganges werden fehlerträchtige *Routinearbeiten* reduziert. Dies wirkt entlastend vor allem in der Testphase, da diese sich hauptsächlich auf konzeptionelle Fehler konzentrieren kann. Da die Testphase in üblichen Vorgehensmodellen der Softwareentwicklung im Verhältnis zu den restlichen Phasen der Softwareerstellung viel Zeit in Anspruch nimmt, wirkt sich dies auch auf die Kosten aus.

Der hohe Abstraktionsgrad der fachlichen Modelle ermöglicht es, aus derselben Modellbasis Implementierungen für verschiedene *Plattformen* und Technologien zu generieren. Die Plattformunabhängigkeit wirkt sich vorteilhaft aus sowohl auf die Nutzbarkeit der Modelle für verschiedene Zielplattformsysteme als auch für verschiedene Zielplattformversionen desselben Zielplattformsystems. Somit kann auch auf kurze Zyklen in der Plattformversionierung reagiert werden.

Die fachlichen Modelle avancieren mit dem MDD zu Verwaltungs- und Sicherungsobjekten, an denen isoliert der Aufwand durch Systemänderungen und –wartungen entsteht. Somit wird der Aufwand durch Änderungen des Plattformcodes reduziert. Da eine Modellbasis für mehrere Plattformen genutzt werden kann, müssen neue Anforderungen an das Softwaresystem nicht mehrmalig umgesetzt werden.

Durch die Nutzung einer Modellbibliothek mit *Standardmustern* für bekannte Problemstellungen kann die Entwicklungsgeschwindigkeit erhöht werden. Die Nutzung von Modellbibliotheken auf der fachlichen Ebene entspricht der Wiederverwendung von Programmcode in Klassenbibliotheken und Frameworks in Programmiersprachen der dritten Generation (3GL).

Betriebswirtschaftlich gesehen werden die fachlichen Modelle mittels MDD nicht mehr als rein dokumentatorische Aufwandsposition im Sinne eines Seiteneffektes der Programmierleistung aufgefasst, sondern als zu pflegendes und kurzfristig in Lösungen transformierbares Wissen des Unternehmens, das als Ausgangsbasis für Wartungen und weitere Projekte dienen kann.¹⁴

In den Softwareerstellungsprozess können auch Stakeholder wie z.B. der Auftraggeber integriert werden, da die Arbeit auf der fachkonzeptionellen Ebene keine Implementierungskennnisse voraussetzt.

¹⁴ Mellor, Scott, Uhl, Weise (2004), S. 10 f.

2.1.5 Werkzeuge

Im Laufe der Entwicklung und Nutzung von MDD wurden Computerprogramme erstellt, die den Prozess des MDD softwaretechnisch unterstützen. Diese lassen sich in die Kategorien der reinen Modellierungswerkzeuge, der reinen Transformatoren und der integrierten MDD Werkzeuge einteilen:

Reine Modellierungswerkzeuge stellen nur die grafische Darstellung, aber keine Transformation der erstellten Modelle bereit. Sie können als Modelleditor aufgefasst werden. Somit müssen Modelle mittels eines Dateiformates wie z.B. XMI exportiert werden, um sie mit gesonderten Transformatoren zu verarbeiten.

Reine Transformatoren ohne Darstellungs- und Modellierungsfunktionalität dienen dazu, Modelle über ein Dateiformat wie z.B. XMI in ein internes Modellformat zu importieren, zu transformieren und anschließend wieder zu exportieren.

Integrierte MDD Werkzeuge dienen der Modellierung, Modelltransformation und Codegenerierung in einem Werkzeug. Somit sind keine Import- und Exportvorgänge nötig, da die gesamte Funktionalität integriert ist. Dies verhindert Kompatibilitätsprobleme beim Datenaustausch und den Rüstaufwand bezüglich der Integration verschiedener Werkzeuge. Auch ist die Navigierbarkeit und Synchronisation zwischen fachlichem und technischem Modell optimal. Beispiele für integrierte MDD-Werkzeuge sind Eclipse mit OpenArchitectureWare, ObjectiF, Together Architect und Visual Studio 2007.

2.2 Model Driven Architecture

Model Driven Architecture (MDA) ist ein Standard der Object Management Group (OMG), welche 1989 gegründet wurde und heute ein offenes Konsortium aus ca. 800 Firmen ist. Die Meinungen über die konstituierenden Merkmale der MDA sind gespalten, lassen sich aber in drei Gruppen einteilen:¹⁵

1. *UML PIM*: Dieser Ansatz sieht vor, das PIM rein in der Unified Modeling Language (UML) auszudrücken. Die in UML enthaltenen Spracherweiterungsmöglichkeiten dienen zum Anlegen domänenspezifischer Sprachen, um der Anforderung an eine problemorientierte Modellierung zu entsprechen. Problematisch ist die Tatsache, dass UML über sein Metamodell semantisch nicht eindeutig definiert ist, und somit eine Transformation in Code nicht deterministisch sondern vom Transformator abhängig abläuft.

¹⁵ Vgl. Fowler (2005).

2. *MOF*: Mittels der Meta-Object Facility (MOF) entworfene domänenspezifische Sprachen können standardisiert ineinander transformiert werden. Die Sprache MOF ermöglicht z. B. das Modellieren der Sprache UML in einem Metamodell. Somit ist der Prozess der MDA nicht auf UML als Modellierungssprache begrenzt. Da in der MOF die Transformationsdefinition nicht spezifiziert wird, wird sie durch die Spracherweiterung Queries, Views, Transformations (MOF-QVT) um diesen Aspekt ergänzt.
3. *Executable UML (XUML)*: Bei dem XUML-Ansatz wird aus einem in UML verfassten Modell durch einen Compiler Programmcode generiert. UML ist in dieser Interpretation eine Programmiersprache. Dies entspricht der Zielvorstellung ausführbarer Modelle.

Allen Ansätzen ist gemein, dass die OMG fordert, die Spezifikation von der Implementierung auf einer gegebenen Plattform zu trennen. Analog zum MDD basiert MDA auf den vier Ebenen des Geschäftsmodells, PIM, PSM und der Implementierungsebene. Das Geschäftsmodell umfasst das praxisnahe Domänenwissen unabhängig von technologischen Details, und wird auch computation independent model (CIM) oder domain model genannt. Es orientiert sich nicht an der Funktionalität des zu implementierenden Systems, sondern beschreibt die Umwelt, in der das System laufen soll. Somit stellt es die Schnittstelle für die Domänenexperten dar. Im PIM ist das Domänenwissen des Geschäftsmodells plattformneutral und technologieorientiert umgesetzt. Es abstrahiert vom konkreten Zielsystem, orientiert sich aber an grundlegenden Konzepten wie z.B. der objektorientierten Programmierung. Das Geschäftsmodell und PIM sind sprach- und systemunabhängig¹⁶.

Die Ziele der OMG bzgl. MDA sind in allen drei Ansätzen die Erhaltung der Portabilität, der Wiederverwendbarkeit durch die Trennung der Modelle in der Architektur und der Interoperabilität, also der Herstellerunabhängigkeit durch Standardisierung.¹⁷ Im Vergleich zu den Zielen der Abstraktion, Modularisierung, Kapselung und Wiederverwendung in der objektorientierten Programmierung wird der Schwerpunkt hinsichtlich der Nutzung von Modellen deutlich.

Im Folgenden wird die Architektur der MDA mit dem Schwerpunkt auf dem MOF-Ansatz als Hauptströmung in der MDA erläutert.

¹⁶ Vgl. Miller, Mukerji (2006), S. 19 f.

¹⁷ Vgl. Miller, Mukerji (2003), S. 12.

2.2.1 Standards der OMG

Die OMG standardisiert als Konsortium die Elemente des MDA-Prozesses, um die Interoperabilität zwischen Produkten verschiedener Hersteller zu gewährleisten. Dazu hat sie als Grundlage bereits bestehende und durch die OMG entwickelte Technologien gewählt, und teilweise für die MDA erweitert. Als domänenspezifische Sprache für das PIM und PSM wird UML unter Anwendung von UML-Profilen vorgegeben. Diese sind in der Metametasprache MOF formal definiert, um für die automatisierte Transformation nutzbar zu sein. Die Verwendung von MOF für die Definition der Sprachen stellt sicher, dass Werkzeuge, die mit MOF umgehen können, alle in MOF formal definierten Sprachen unterstützen. Zur Standardisierung der formalen Definitionen der Transformationen zwischen Modellen werden die Sprachen der MOF-QVT genutzt. Mittels der Object Constraint Language (OCL) können in UML verfasste Modelle um Einschränkungen auf Modellelementen angereichert werden.¹⁸

2.2.2 UML als Sprache für die MDA

Die UML stellt für die Umsetzung des MDA-Prozesses eine zentrale Rolle dar. Sie wurde hauptsächlich von Grady Booch, Ivar Jacobson und James Rumbaugh entwickelt, und im November 1997 als Standard in der Version 1.x von der OMG aufgenommen. Diese pflegt die Sprache seitdem und erweiterte ihren Umfang deutlich, um sie im April 2006 als Version 2.1 zu veröffentlichen. Aktuell wird sie durch eine Task Force überarbeitet um im Dezember 2006 in einer Version 2.2 vorgelegt werden zu können.

Der Umfang der Spezifikation von UML wuchs durch das Hinzufügen neuer Elemente von einer Spezifikation der Version 1.x auf drei Teilspezifikationen der Version 2.0. Die UML 2.0 Infrastructure Specification legt die Basis für die weiteren Teilspezifikationen, indem sie Sprachelemente wie die Klasse, Assoziation und Multiplizität spezifiziert. Diese können durch die Spezialisierung verfeinert werden. Die UML 2.0 Superstructure Specification definiert Modellelemente, die für spezifische Einsatzzwecke dienen, wie z. B. den Anwendungsfall, den Zustandsautomaten und die Aktivität. Die Teilspezifikation UML 2.0 Object Constraint Language spezifiziert die OCL. Die drei Teilspezifikationen werden durch das Dokument UML 2.0 Diagramm Interchange erweitert, das das Layout der Diagramme spezifiziert. Diagramme dienen der grafischen Darstellung von Modellen, erweitern diese also um grafikbezogene Daten wie z. B. Positionsdaten von Modellelementen.

¹⁸ Vgl. Kleppe, Warmer, Bast (2003), S. 33 f.

Mit UML lässt sich sehr effektiv die Modellierung von Strukturen durchführen. Dazu werden die zu modellierenden statischen Konzepte der Realwelt in Strukturmodellen erfasst, die durch Strukturdiagramme dargestellt werden können. UML 2 umfasst die sechs Strukturdiagramme Klassen-, Kompositionsstruktur-, Komponenten-, Verteilungs-, Objekt- und Paketdiagramm. Dynamische Konzepte können durch die sieben Verhaltensdiagramme Anwendungsfall-, Aktivitäts-, Sequenz-, Kommunikations-, Interaktionsübersichts-, Zeitverlaufs- und Zustandsdiagramm abgebildet werden.

Der Schwachpunkt von klassischem UML für die Nutzung in der MDA ist die mangelnde Eindeutigkeit in der Modellierung dynamischer Verhaltenskonzepte. Die formale Semantik von UML ist zur eindeutigen Abbildung dieser Aspekte insuffizient spezifiziert, z. B. kann die Beziehung eines Anwendungsfalldiagramms zum Programmcode für diverse Plattformen nur ungenau festgelegt werden. Da ein UML-Verhaltensdiagramm mehrdeutig interpretiert werden kann, ist es nicht möglich, aus einem rein in UML verfassten PIM mit Verhaltensaspekten ein PSM abzuleiten.¹⁹

UML mit Object Constraint Language

Die Object Constraint Language (OCL) erweitert die Ausdrucksmächtigkeit von UML und MOF um eine Abfrage- und Ausdruckssprache, die es erlaubt, Einschränkungen und Zusatzinformationen für UML-Modelle bzw. MOF-basierte Sprachen zu definieren. In der Entwicklung der OCL wurde die anfänglich rein constraintbasierte Sprache um Abfrageaspekte erweitert. Der Name Object Constraint Language trifft also nur eine Teilmenge der ermöglichten Funktionalität.

Textuelle Ausdrücke in OCL können aus vier Teilen konstruiert werden:²⁰

1. Der Kontext, in dem der Ausdruck gelten soll, gewöhnlicherweise das UML-Modell bzw. Klassen oder Objekte desselben.
2. Eine Eigenschaft wie z.B. ein Attribut eines Objektes oder einer Klasse.
3. Eine Mengen- oder Arithmetikoperation, die Vergleiche oder Wertmanipulationen ermöglicht und einen Rückgabewert erzeugt.
4. Schlüsselwort wie z. B. if, then, else, also konditionale Aspekte, die den Ausdruck nur unter einer Bedingung gelten lassen.

¹⁹ Vgl. Kleppe, Warmer, Bast (2003), S. 35.

²⁰ Vgl. Frankel (2003), S. 83 ff.

Die Dynamik des Systems kann mittels Vor- und Nachbedingungen von Operationen in der Notation von UML ausgedrückt werden. Auch ist es möglich, Sachverhalte wie Initialwerte von Variablen, Ableitungsregeln und Abfragen zu erfassen. Die Menge der OCL-Ausdrücke spiegelt die Umwelt der Domäne auf abstraktem Niveau wider. Dies ermöglicht die Erstellung genauer, eindeutiger, umfassender und konsistenter also widerspruchsfreier Modelle. Damit ist die Transformation eines PIM in ein PSM und anschließend Programmcode inklusive der automatisierten Erstellung von Methoden möglich.

UML-Profile

UML enthält einen Standardmechanismus für die Erstellung von Profilen durch die Spezialisierung des UML-Metamodells für abgegrenzte Domänen. Dies bietet die Möglichkeit, den Sprachumfang von UML flexibel zu erweitern, ohne in das Dilemma zwischen Komplexität und Mächtigkeit einer Sprache zu gelangen und eine neue Modellierungssprache entwerfen zu müssen. Dazu werden die Basiskonstrukte der UML wie Klassen und Assoziationen ausdifferenziert und so mittels neuer Sprachkonzepte die formal semantisch korrekte Abbildung der Domäne ermöglicht.

Das Profil wird durch eine Menge von Stereotypen, Constraints und Tagged Values konstituiert. Modellelementen können mehrere Stereotype zugeordnet werden, wodurch sie semantisch angereichert und in einen Verwendungszusammenhang gestellt werden. Tagged Values können als Metaklassenattribute aufgefasst werden, die einen Typ und eine Ausprägung haben. In OCL definierte Constraints beschreiben die Einschränkungen für Modellelemente des Profils entsprechend ihres Stereotyps. Dies ermöglicht z. B., dass sämtliche Klassen mit dem Stereotyp <<JavaClass>> auf maximal eine Superklasse restringiert sind, um Mehrfachvererbungen im Javaquellcode zu vermeiden.²¹

Für einige Domänen wurden Profile bereits von der OMG vorgegeben und standardisiert. Beispiele dafür sind:

1. Das Common Warehouse Metamodel (CWM), das der Beschreibung, dem Zugriff und dem Austausch von Metadaten im Data-Warehouse-Prozess dient. Die Erweiterungen betreffen insbesondere Metaklassen zum Modellieren relationaler Datenbanken, OLAP, geschäftlicher Metadaten etc. Für die Modellierung der Datenbankaspekte irrelevante Metamodellelemente für die Abbildung von Verhaltensaspekten wurden entfernt.

²¹ Vgl. Kleppe, Warmer, Bast (2003), S. 140.

2. Das Enterprise Distributed Object Computing (EDOC) ist ein UML-Profil zur Modellierung kollaborierender Komponenten und Geschäftsprozesse unabhängig von Middleware Technologien.²²
3. Die Common Object Request Broker Architecture (CORBA) ist eine Middleware, in der Protokolle und Dienste plattformübergreifend spezifiziert werden. Sie dient der vereinfachten Erstellung verteilter Anwendungen in heterogenen Umgebungen wie z. B. dem Nachrichtenaustausch zwischen Softwaresystemen verschiedener Hersteller unabhängig von dem verwendeten Framework.
4. Das UML-Profil Enterprise Application Integration (EAI) unterstützt die Integration der Geschäftsfunktionen verschiedener Funktionsbereiche und Plattformen eines Unternehmens. Dazu werden Einzelsysteme über Schnittstellendefinitionen standardisiert und auf Basis einer Integrationsplattform interoperabel gestaltet.

Für nicht abgedeckte Domänen existieren Initiativen, die UML-Profile veröffentlichen, diese aber nicht einer Standardisierung durch die OMG unterziehen. Ein Beispiel dafür ist die UML/EJB Mapping Specification, die durch den Java Community Process standardisiert wird.

Executable UML

Executable UML (XUML) ist die Kombination von klassischem UML und Action Semantics (AS). Die Schwäche von UML als deklarativem Ansatz in der Abbildung dynamischer Konzepte wird durch den Einsatz von Action Semantics behoben. AS ist ein praxisnahes Framework für semantische Beschreibung von Programmiersprachen, das operationale, denotationelle und algebraische Semantiken kombiniert. Es ist geeignet zur formalen Beschreibung von Programmiersprachen. Durch die inhärente Erweiterbarkeit und Anpassbarkeit von AS ist gewährleistet, dass eine Modifikation der beschriebenen Programmiersprache nur eine entsprechende Modifikation der Beschreibung nach sich zieht.²³

Im Gegensatz zum hauptsächlich deklarativen OCL beschreiben Ausdrücke in AS imperativ das Verhalten. Die in AS verfassten Terme können auf Elemente eines UML-Modells verweisen, und somit im Kontext des Modells bewertet werden. Der Verhaltensaspekt eines Systems wird in XUML durch eine Zustandsmaschine ausgedrückt, bei der jeder Zustand ein Ankerpunkt für vordefiniertes Verhalten ist. Das Verhalten wird durch eine in AS verfasste Prozedur ausgedrückt. Dies bildet die Dynamik des Systems ab und ermöglicht

²² Vgl. Frankel (2003), S. 60.

²³ Vgl. Mosses (2004).

die Transformation eines in XUML verfassten PIM in PSM und anschließend Programmcode. Aus diesem Ansatz heraus ergeben sich mit XUML einige Nachteile bei der Erstellung eines PIM:

1. Da die Syntax der AS nicht standardisiert ist, lassen sich Ausdrücke in AS nicht standardisiert definieren.
2. Der Abstraktionsgrad von Ausdrücken in AS entspricht dem eines PSM. Die Semantik der dynamischen Aspekte wird nur feingranular beschrieben, sodass größere Zusammenhänge der modellierten Domäne verloren gehen. Eine plattformübergreifende Nutzbarkeit von in XUML verfassten Modellen ist damit nicht gegeben. Somit stellt die Erstellung eines PIM für die dynamischen Aspekte keinen Mehrwert mehr dar.²⁴

Dies führt zu einer eingeschränkten Anwendbarkeit von XUML begrenzt auf spezielle Domänen wie dem Embedded Computing.

2.2.3 Meta-Object Facility

Die OMG sieht als Standardmodellierungssprache der Metametamodellebene M3 die Meta-Object Facility (MOF) vor. Alle Modellierungssprachen der OMG wie UML und CWM werden aus der MOF abgeleitet bzw. dahingehend erweitert. UML wurde z. B. erst ab der Version 2 aus MOF ableitbar. Da die Sprachen der OMG durch MOF spezifizierbar sind, können sie durch das Dateiformat XML Metadata Interchange (XMI) gespeichert und standardisiert ausgetauscht werden.

Die OMG sieht keine weitere explizite Ebenendefinition M4 vor, also ein Modell der Sprache MOF, sondern stellt MOF als selbstbeschreibend dar. Das bedeutet, dass ein Modell der MOF wiederum in MOF abgefasst ist. Da XMI-Dateien Instanzen der MOF-Typen speichern, und MOF-Instanzen durch MOF-Typen darstellbar sind, kann auch die Sprache MOF durch das XMI-Dateiformat gespeichert werden.

Das Sprachenbündel Queries, Views, and Transformations (MOF-QVT), das Bestandteil der MOF ist, standardisiert die Transformationsdefinitionen für alle in MOF erstellten Sprachmodelle. Es umfasst Sprachen für die Erstellung von Sichten auf Modelle, für Abfragen auf Modelle und für die Erstellung von Transformationsdefinitionen. Somit wird sichergestellt, dass auch die Transformationsdefinitionen als wesentlicher Teil der MDA herstellerunabhängig sind.

²⁴ Vgl. Kleppe, Warmer, Bast (2003), S. 35 f.

2.2.4 Nutzen der Metamodellierung für die MDA

Dem Ansatz der Abstraktion von Modellierungssprachen kann man entgegen halten, dass die Spezifikation der Ebenen willkürlich ist. Eine Modellierungssprache kann auf vielerlei Art und Weise in einem Modell abgebildet werden. Dennoch sieht die OMG in der Standardisierung durch Meta- und Metametamodelle eine Notwendigkeit und Vorteile, die hauptsächlich in den folgenden Punkten begründet liegen.

1. Um Transformatoren ein Lesen, Schreiben und Interpretieren von Modellen zu ermöglichen, ist eine eindeutige Definition von Sprachen durch Metamodelle nötig.
2. Zur Erstellung der Transformationsdefinitionen sind entsprechend den Quell- und Zielmodellen Metamodelle der Quell- und Zielsprachen nötig. Die Elemente der Metamodelle dienen als Referenzpunkte für die Transformationsregeln. Dies ermöglicht eine Erfassung der formalen Semantik, und somit eine Transformation zwischen PIM und von PIM zu PSM.
3. Die Spezifikation der Modellierungssprachen der OMG wird durch die Ableitung von der MOF vereinheitlicht. Dies ist wichtig für eine eindeutige Dokumentation im Standardisierungsprozess der MDA durch die OMG und externe Standardisierungsinitiativen.
4. Die Spezifikation einer Modellierungssprache durch MOF ermöglicht den Austausch der Sprache durch das XMI-Dateiformat und sorgt so für Interoperabilität.

2.3 Universal Application

Universal Application (UA) ist ein Fachbegriff für eine alternative Art der Softwareentwicklung, bei der ein Anwendungssystem kundenspezifisch mittels Metadaten konfiguriert wird. Die Metadaten beschreiben die Daten und das Verhalten der Domäne des Kunden und werden von der Laufzeitumgebung der UA interpretiert. Die Vision ist die Vermeidung von Programmertätigkeiten durch Konfiguration im Gegensatz zu der Programmierunterstützung in der MDA und dem CASE-Ansatz. Die Vorgehensweise der Nutzung einer Laufzeitumgebung ermöglicht die Erstellung von Individualsoftware aus Standardsoftware.

Der Prozess der Entwicklung einer UA-Applikation verläuft iterativ und interaktiv. Im Mittelpunkt der Betrachtung steht dabei die Datenbankstruktur, nicht die spätere Visualisierung. Eine Iteration besteht aus dem Beschreiben der Problemstellung durch Metadaten, der Ermittlung eines Vorschlages durch die UA und der anschließenden Verfeinerung der Metadaten durch den Entwickler. Das Ergebnis des Entwicklungsprozesses sind aus-

schließlich Metadaten im Gegensatz zu der Programmcodeerstellung in den klassischen Verfahrensmodellen der Softwareentwicklung. Die Änderungen der Metadaten können optimalerweise am laufenden System vorgenommen werden, um die Auswirkungen direkt nachvollziehen zu können. Die Anwendungslogik des Anwendungssystems wird weitestgehend durch die in den Metadaten enthaltenen Formel- und Constraintdefinitionen auf Attributen und Tabellen der Datenbank von der Laufzeitumgebung ermittelt. Da der Einsatz einer UA datenorientiert motiviert ist, lässt sich ein hoher Grad an Funktionalität automatisiert bereitstellen. Algorithmen, die nicht aus den Metadaten ableitbar sind, können durch Entwickler explizit implementiert werden. Die definierte Funktionalität kann anschließend in der Visualisierung in Form von Elementen in Masken zusammengesetzt werden.²⁵

Die Erstellung der Geschäftsregeln erfolgt rein deklarativ. Die Umsetzung der Regeln in Anweisung wird durch die UA durchgeführt. Produkte wie der Process Server von Scopeland sollen auch die deklarative Definition von Prozessen ermöglichen, die anschließend durch die Laufzeitumgebung der UA umgesetzt werden. So sollen Geschäftsregeln implementierungsunabhängig das Unternehmen langfristig abbilden. Eine Einschränkung der Nutzbarkeit ergibt sich aber durch die Tatsache, dass die Prozesse datenbankorientiert umgesetzt werden, und hauptsächlich der Stapelverarbeitung von Datensätzen dienen.

Die nicht operationalen Ziele des Softwareentwicklungskonzepts UA sind eine größtmögliche Flexibilität, Kostenreduktion, die erhöhte Geschwindigkeit der Konfiguration gegenüber der Programmierung und die Investitionssicherheit durch Anpassbarkeit. Anbieter von UA-Produkten wie die Scopeland Technology GmbH und Tenfold werben mit Kostensparnissen von bis zu 90%. Das für den Benutzer sichtbare Ergebnis des Entwicklungsprozesses soll in der Nutzung von einer klassisch erstellten Applikation nicht zu unterscheiden sein. Eine Ähnlichkeit zum Vorgehensmodell des MDD ergibt sich aus der Möglichkeit, aus den Metadaten in Verbindung mit der UA statischen Programmcode zu erzeugen.

Der Vorteil der Entwicklung von Individualsoftware aus Standardsoftware im Gegensatz zur Entwicklung reiner Individualsoftware liegt in der beschleunigten Entwicklung und langfristigen Wartung der Laufzeitumgebung durch den Hersteller, die bei Individualsoftware oft nicht gegeben oder finanzierbar ist.

²⁵ Noack (2006).

3 Vergleich und Bewertung der Ansätze

3.1 Model Driven Architecture und Computer-Aided Software Engineering

Die Model Driven Architecture und der in den 1990-Jahren entwickelte CASE-Ansatz unterscheiden sich durch ihre Ziele und ihre Umsetzung. Das Ziel des CASE-Ansatzes ist es, den Softwareentwickler in den Schritten der Erstellung von Software durch eine integrierte Entwicklungsumgebung zu unterstützen. Die Entwicklungsumgebung soll die üblichen Entwicklungsphasen abdecken können und enthält klassischerweise einen Modelleditor, ein Repository mit Entwurfsmustern, einen in einen Programmeditor integrierten Debugger und eine Dokumentationsfunktion

Klassisch zeichnet sich CASE durch statische Metamodelle und Transformationen aus, sodass die verwendeten Modellierungssprachen nicht flexibel an die Domäne anpassbar und zudem häufig proprietär sind. Dies macht die durch die MDA angestrebte Interoperabilität unmöglich. Die MDA hingegen sieht die manuelle Anfertigung der fachlichen Modelle und Architekturen und die anschließende teilautomatisierte Codegenerierung vor. Dabei liegt der Schwerpunkt auf der fachkonzeptionellen Unterstützung losgelöst von der Wahl einer Entwicklungsumgebung.²⁶

CASE hat sich als Unterstützung von imperativen Programmiersprachen als nützlich erwiesen, stellt aber nicht eine Weiterentwicklung im Sinne eines neuen Softwareentwicklungsparadigmas dar. Die Etablierung der MDA als neues nutzbares Paradigma setzt voraus, dass die Werkzeuge die Trennung zwischen PIM und PSM durchführen können, da sie sonst dem Anspruch der Plattformunabhängigkeit als treibendem Faktor nicht gerecht werden kann.

QVT-MOF muss als junger Standard für die Transformationsdefinitionen etabliert und durch die Werkzeuge unterstützt werden, um die herstellerunabhängige Verbreitung der MDA zu fördern. Es bleibt abzuwarten, ob dies in der Reinform einer vollen Standardkonformität geschehen wird, oder ob sich herstellerspezifische Dialekte herausbilden werden.

Ein dauerhafter Gegenstand der Optimierung wird die Transformation der Geschäftslogik in Algorithmen sein, da die Abstraktion eines Modells mit einer Reduktion des Informationsgehaltes der konkreten Abläufe einhergeht. Eine weitgehende Automatisierung in einem mittelfristigen Zeitrahmen scheint nur im Falle des Einsatzes von Musteralgorithmen für Standardproblemmuster möglich.

²⁶ Rossbach, Stahl, Neuhaus (2003), S. 4.

3.2 Model Driven Development und Model Driven Architecture

Das Verhältnis der MDA zum MDD wird sehr unterschiedlich aufgefasst. Während die OMG MDA gegenüber dem MDD als einen neuen Ansatz positioniert, fassen die Anhänger der MDD die MDA nur als eine weitere Ausprägung der schon länger existierenden MDD auf.

Die MDA unterscheidet sich durch den hohen Grad an Standardisierung, die Vorstellung von ausführbaren Modellen in Form von XUML und die Nutzung des Metametamodells MOF von der MDD. MDD wird häufig mit dem Prozess der Softwareentwicklung und den damit verbundenen Aktivitäten assoziiert. Dagegen liegt der Schwerpunkt bei MDA auf der Erstellung formaler Frameworks, die die MDD nutzen kann.

MOF und der Substandard MOF-QVT stellen einen Mehrwert gegenüber den heterogenen Entwürfe im MDD dar. Durch MOF wird es möglich, Werkzeuge wie die Modellierungsumgebungen und Transformatoren interoperabel zu gestalten, und somit einen Markt für entsprechende Produkte zu etablieren. OpenArchitectureWare, das aus dem MDD kommt und im MDD in vielen Projekten genutzt wird, bringt eine Unterstützung für verschiedene Metametamodelle wie z. B. Ecore, Javabeans und EMOF mit. EMOF ist eine vereinfachende Untermenge des MOF. Dies ist ein Beispiel für die Auswirkung der MDA auf klassische Bereiche des MDD.

3.3 Model Driven Development und Universal Application

Sowohl mit MDD als auch mit UA kann Programmcode einer Individualsoftware als Ergebnis des Entwicklungsprozesses erstellt werden. Dies ist aber nicht das unterscheidungsrelevante Kriterium, da die Vorgehensweisen beider Ansätze grundsätzlich verschieden sind. Der MDD-Ansatz ist modellzentriert und soll die Erstellung von Programmcode im Idealfall automatisieren. Hingegen soll der UA-Ansatz die Generierung von Quellcode grundsätzlich vermeiden und durch die Konfiguration einer Standardsoftware ersetzen. Das Ergebnis der Entwicklung ist nicht Programmcode, sondern eine abstrakt deklarative Beschreibung.

Durch den Einsatz einer Standardsoftware bei der UA kann die Wartung der Laufzeitumgebung vom Hersteller übernommen werden. Dies stellt einen Vorteil gegenüber reiner Individualsoftware dar, erhöht aber auch die Abhängigkeit vom Hersteller, falls die Laufzeitumgebung wie üblich nicht als offener Quellcode für den Kunden verfügbar sein sollte. Durch eine schwache Standardisierung ist eine Migration zwischen verschiedenen Anbietern von UA zudem mit hohen Kosten verbunden.

Die Nutzbarkeit von UA ist im Gegensatz zum MDD beschränkt auf Anwendungssysteme mit dem Schwerpunkt auf Schnittstellenmasken für Datenbanksysteme. Eine Umsetzung von Prozessmodellen wird angestrebt, beschränkt sich aber weiterhin auf die Automatisierung von Datenbankoperationen.

Die durch die Unternehmen Tenfold und Scopeland Technology GmbH beworbenen Einsparungen von bis zu 90% des üblichen Budgets für vergleichbare Anwendungssysteme hängen von der Art der Anwendung ab. Vor allem die Entwicklung komplexer Algorithmen ist sowohl durch UA als auch durch MDD nicht ausschließlich mittels von Modellen möglich, sodass die manuelle Implementierung die Kostenreduktionseffekte in beiden Ansätzen reduziert.

Im Vergleich zum MDD stellt UA mit der Unterstützung durch einige wenige Anbieter momentan eine Nischenlösung dar. Die Tatsache, dass die Basissysteme der Anbieter nicht interoperabel sind, erhöht das Investitionsrisiko einer Umsetzung mit UA.

4 Ausblick

Die modellgetriebene Softwareentwicklung verspricht in Gestalt der MDD und MDA eine fundamentale Änderung des Softwareerstellungprozesses. Ein Vergleich der Entwicklungsschritte von der assemblerorientierten zu der prozeduralen Programmierung und der prozeduralen zu der objektorientierten Programmierung zeigt das Potenzial, das in der zunehmenden Abstraktion von der Maschine Computer liegt. Der Fokus des Softwareentwicklungsprozesses liegt nicht mehr auf dem Lösen computerspezifischer Problemstellungen, sondern auf der Umsetzung der geschäftlichen Anforderungen.

Der CASE-Ansatz als vielversprechender Kandidat für eine Weiterentwicklung brachte aufgrund der beschriebenen Restriktionen nicht den Durchbruch. Die OMG hat sich mit MDA die Vermeidung dieser Restriktionen und die Schaffung eines neuen Paradigmas der Softwareentwicklung vorgenommen. Ob sich MDA durchsetzen wird, hängt hauptsächlich davon ab, ob das Konzept der MDA zu mächtig ist und damit nicht mehr durch Hersteller in Werkzeuge umsetzbar sein wird.

Literaturverzeichnis

- Ehlert, M.: MDD - Rechnet sich das in meinem Projekt? In: Objektspektrum. 2005 02, S. 1-4.
- Fowler, M.: Language Workbenches and Model Driven Architecture. 2005.
<http://www.martinfowler.com/articles/mdaLanguageWorkbench.html> 2006-11-09.
- Frankel, D.: Model driven architecture: applying MDA to enterprise computing. New York, Chichester 2003.
- Kleppe, A.; Warmer, J. B.; Bast, W.: MDA explained the model driven architecture; practice and promise. Reading/Mass., Munich et al. 2003.
- Mellor, S. J.; Scott, K.; Uhl, A.; Weise, D.: MDA distilled principles of model-driven architecture. Boston/Mass., London 2004.
- Miller, J.; Mukerji, J.: MDA Guide Version 1.01. 2003. <http://www.omg.org/docs/omg/03-06-01.pdf> 2006-11-09.
- Mosses, P.: About Action Semantics. 2004.
<http://www.brics.dk/Projects/AS/AboutActionSemantics.html> 2006-11-09.
- Noack, K.: Thinking Scopeland. 2006.
<http://www.scopeland.de/de/thinkingscopeland/titel.html> 2006-11-09.
- Roßbach, P.; Stahl, T.; Neuhaus, W.: Grundlegende Konzepte und Einordnung der Model Driven Architecture (MDA). In: Javamagazin. 2003 09, S. 1-4.
- Weilkiens, T.; Oestereich, B.; Stahl, T.: Vom Geschäftsmodell zum Code - ein kurzer Weg mit MDA. In: Javamagazin. 2003 09, S. 9-13.