

**Ralf Reussner ( Hrsg.)**

**MINT**  
**Modellgetriebene Integration  
von Informationssystemen**

**Projektbericht**



---

universitätsverlag karlsruhe



Ralf Reussner (Hrsg.)

**MINT – Modellgetriebene Integration von Informationssystemen**  
**Projektbericht**



# MINT – Modellgetriebene Integration von Informationssystemen

## Projektbericht

herausgegeben von

Ralf Reussner, *Universität Karlsruhe (TH)*

unter Mitarbeit von

Xinghai Chi, *BTC AG*

Heiner Feislachen, *BTC AG*

Cord Giese, *Delta Software Technology GmbH*

Thomas Goldschmidt, *FZI*

Michael Gründler, *BTC AG*

Steffen Kruse, *Universität Oldenburg*

Thomas Kühn, *andrena objects AG*

Jing Shui, *BTC AG*

Ulrike Steffens, *OFFIS*

Niels Streekmann, *OFFIS*

Jochen Winzen, *andrena objects AG*

Malte Zilinski, *Universität Oldenburg*



Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01ISF07 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

## Impressum

Universitätsverlag Karlsruhe  
c/o Universitätsbibliothek  
Straße am Forum 2  
D-76131 Karlsruhe  
www.uvka.de



Dieses Werk ist unter folgender Creative Commons-Lizenz  
lizenziert: <http://creativecommons.org/licenses/by-nc-nd/3.0/de/>

Universitätsverlag Karlsruhe 2009  
Print on Demand

ISBN: 978-3-86644-416-4

# Vorwort

In einer Umgebung sich rasch verändernder Anforderungen und kürzer werdender Innovationszyklen ist es eine wichtige Eigenschaft heutiger betrieblicher Software, schnell und kosteneffizient an neue oder sich ändernde Geschäftsprozesse anpassbar zu sein. Oft werden aber bestehende Systeme Anforderungen ausgesetzt, für die sie ursprünglich nicht entworfen wurden, wie zum Beispiel Web-Anbindungen im Bereich des sog. Business-to-business-eCommerce. Dies zeigt sich zum einen in über die Zeit gewachsenen Softwaresystemen, die die erforderliche Flexibilität nicht erreichen, und zum anderen in heterogenen Softwaresystemlandschaften, deren Datenhaltung und Arbeitsabläufe nicht integriert sind, obwohl dieses für die Geschäftsprozesse des Unternehmens vorteilhaft wäre. Gerade KMU stellt die Integration bestehender Software mit neuen Anwendungen vor oft kaum zu bewältigende finanzielle und technische Herausforderungen. Die Teilnahme der KMU an endkundenbezogenen oder unternehmensübergreifenden eCommerce-Abläufen wird damit verhindert, wodurch sich offensichtliche Wettbewerbsnachteile ergeben.

Ein Großteil der deutschen Softwareentwicklung findet im Auftrag von hochgradig spezialisierten mittelständischen Unternehmen der sogenannten Sekundärindustrie statt. Die für diese Unternehmen benötigte Software kann i.d.R. nur durch eine enge Zusammenarbeit zwischen Kunde und IT-Unternehmen entwickelt werden. Neben Maßnahmen zur Steigerung der Effizienz bei der Entwicklung neuer Softwaresysteme ist auch hier die effiziente Anpassbarkeit und Integration bestehender Software von hoher Bedeutung. Da in die Entwicklung bereits bestehender Softwaresysteme ein beträchtlicher finanzieller Aufwand geflossen ist und ein für die Geschäftsidee essenzielles Know-how implizit in dieser Software steckt, ist ihre Abschaltung unmöglich. Stattdessen sind Verfahren von Interesse, die eine schrittweise Migration einer solchen unternehmenskritischen Software zu einer neuen und flexiblen Softwarearchitektur erlauben.

Die modellgetriebene Entwicklung versucht, der Forderung nach erhöhter Flexibilität bei der Softwareentwicklung durch den Einsatz von Generatortechnologie Rechnung zu tragen. Allerdings werden die Vorteile modellgetriebener Ansätze auch für KMU aus zwei Gründen verringert: Zum einen konzentrieren sich bestehende modellgetriebene Ansätze auf die Neuentwicklung und vernachlässigen die in der Praxis ausgesprochen wichtige Problematik der Integration und Evolution bestehender Softwaresysteme. Zum anderen kapseln heutige modellgetriebene Verfahren zwar technische Plattformspezifika, benötigen aber dennoch Eingabemodelle, die konzeptionell häufig kaum abstrakter sind als Programm-Code und damit nur ähnlich kostenintensiv zu erstellen.

Im Forschungsprojekt MINT (Modellgetriebene Integration von Informationssys-

---

temen) wurde ein innovatives modellgetriebenes Software-Entwicklungsverfahren geschaffen und validiert, um die antizipierten Vorteile modellgetriebener Software-Entwicklung zu nutzen und insbesondere in KMU sinnvoll einzusetzen. Das Verfahren ermöglicht die Nutzung von Architekturbeschreibungen in Form von Modellen und unterstützt die Integration von bestehenden Software-Systemen. MINT wurde vom Bundesministerium für Bildung und Forschung (BMBF) als Verbundprojekt im Rahmen der Forschungsoffensive Software-Engineering 2006 gefördert. An der Durchführung des Projekts waren mit der andrena objects AG, der Business Technology Consulting AG (BTC) und der Delta Software Technology GmbH (DSTG) drei Software entwickelnde Unternehmen beteiligt. DSTG und andrena befinden sich als KMU in dem oben beschriebenen Spannungsfeld. Wissenschaftlich begleitet wurde das Projekt vom FZI Forschungszentrum Informatik in Karlsruhe, vom OFFIS Institut für Informatik in Oldenburg sowie von der Abteilung Lernende und Kognitive Systeme der Universität Oldenburg.

Das Projekt MINT wurde im Juni 2008 erfolgreich abgeschlossen. Für alle Partner lieferte es bedeutende Impulse für ihre weiteren Arbeiten im Bereich der modellgetriebenen Integration, die sich in verschiedenen industriellen Projektaktivitäten ebenso widerspiegeln wie auf wissenschaftlicher Ebene, beispielsweise im ebenfalls durch das BMBF geförderten Projekt IF-ModE, in dem mit OFFIS und der DSTG zwei der MINT-Projektpartner ihre Aktivitäten in Sachen modellgetriebene Softwareentwicklung gemeinsam fortsetzen.

Die Zusammenarbeit im MINT-Projekt zeichnete sich durch die aktive Mitarbeit aller Partner und insbesondere durch das persönliche Engagement der Mitarbeiter aus, denen an dieser Stelle ein ganz persönlicher Dank gebührt. Ebenfalls bedanken möchte ich mich bei Ralf Reussner, der als wissenschaftlicher Leiter das Projekt umsichtig und kompetent von Anfang an in die richtige Richtung gelenkt hat. Das Bundesministerium für Bildung und Forschung hat durch die Förderung das Projekt MINT erst ermöglicht. Herrn Dr. Weber, der beim Projektträger, dem Deutschen Institut für Luft- und Raumfahrt (DLR), das MINT-Projekt intensiv betreut, unterstützt und mit großem Interesse begleitet hat, gilt unser besonderer Dank.

Die Ergebnisse des MINT-Projekts sollen ermutigen, modellgetriebene Softwareentwicklung in der Praxis auch zukünftig als eine realistische Option in Betracht zu ziehen, um aus der aktuell schwierigen Marktsituation konkurrenzfähig und gestärkt hervorzugehen. Daher möchten wir unsere Resultate teilen und stellen sie interessierten Lesern in Form dieses Buches zur Verfügung. Wir hoffen, Ihnen damit Anregungen für Ihre eigene Arbeit zu geben und wünschen Ihnen viel Spaß bei der Lektüre.

Oldenburg, im Juli 2009

Ulrike Steffens



# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
1.1. Rahmenbedingungen . . . . .	2
1.1.1. Aufgabenstellung . . . . .	2
1.1.2. Voraussetzungen des Projekts . . . . .	2
1.1.3. Planung und Ablauf des Projekts . . . . .	5
1.1.4. Wissenschaftlicher und technischer Stand . . . . .	6
1.1.5. Zusammenarbeit mit anderen Stellen . . . . .	7
1.2. Übersicht des MINT-Ansatzes . . . . .	8
1.3. Aufbau des Dokuments . . . . .	9
<b>2. Grundlagen</b>	<b>11</b>
2.1. Modellgetriebene Software-Entwicklung . . . . .	11
2.2. Integration von Softwaresystemen . . . . .	13
2.2.1. Das Dublo-Muster . . . . .	14
2.2.2. Der BALES-Ansatz . . . . .	15
2.2.3. Serviceorientierte Architekturen . . . . .	17
2.3. Modellierungswerkzeuge . . . . .	18
2.3.1. Eclipse Modeling Framework (EMF) . . . . .	18
2.3.2. Graphical Modeling Framework (GMF) . . . . .	18
2.4. Transformationswerkzeuge . . . . .	19
2.4.1. Vorauswahl von MDA-Werkzeugen . . . . .	19
2.4.2. AndroMDA . . . . .	20
2.4.3. openArchitectureWare . . . . .	21
2.4.4. Software Factories . . . . .	24
2.4.5. Bewertung der untersuchten Werkzeuge . . . . .	26
2.5. Werkzeuge für die modellgetriebene Datenintegration . . . . .	28
2.5.1. SCORE Adaptive Bridges – Data Architecture Integration . . . . .	28
2.5.2. HyperSenses . . . . .	33
2.6. Visual Composer . . . . .	36
2.6.1. Motivation . . . . .	36
2.6.2. Einführung in SAP Visual Composer . . . . .	37
2.6.3. Anwendungsszenario mit Visual Composer . . . . .	40
2.6.4. Ergebnisse . . . . .	43
2.7. Anwendungsszenarien für die Prozessintegration . . . . .	45
2.7.1. Störungserfassungsdomäne . . . . .	47
2.7.2. Instandhaltungsdomäne . . . . .	47
2.8. Die andrena Testumgebung . . . . .	49

<b>3. Domänenspezifische Modellierung mit der MINT-XL</b>	<b>53</b>
3.1. Domänenspezifische Modellierung von Integrationsaspekten . . . . .	53
3.2. MINT-XL . . . . .	55
3.3. Erweiterungsmechanismen . . . . .	58
3.3.1. Allgemeine Erweiterung: Mappings . . . . .	59
3.3.2. Domänenspezifische Erweiterung: Störungsmanagement . . . . .	60
3.4. Anwendungsbeispiel MINT-XL . . . . .	62
3.5. Umsetzung . . . . .	63
3.5.1. Umsetzung von MINT-XL mit EMF . . . . .	63
3.5.2. Erstellung des MINT-PIE-Editors mit GMF . . . . .	67
<b>4. Umsetzung wissensintensiver Prozesse</b>	<b>71</b>
4.1. Wissensintensive Prozesse . . . . .	71
4.2. MINT Vorgehensmodell . . . . .	72
4.3. Metamodellerzeugung . . . . .	73
4.4. CIM . . . . .	74
4.5. PIM . . . . .	76
4.6. PSM . . . . .	79
4.7. Zielsystem . . . . .	80
4.8. Zusammenfassung . . . . .	80
<b>5. Modellgetriebene Prozessintegration</b>	<b>83</b>
5.1. Modelle . . . . .	84
5.1.1. Fachliches Modell . . . . .	84
5.1.2. Plattformunabhängiges Architekturmodell . . . . .	86
5.1.3. Annotationsmodell . . . . .	87
5.1.4. Plattformabhängiges Architekturmodell . . . . .	88
5.1.5. Altsystemmodell . . . . .	89
5.1.6. Technisches Mappingmodell . . . . .	92
5.1.7. Code . . . . .	95
5.2. Transformationen . . . . .	98
5.2.1. Vom fachlichen Modell zum plattformunabhängigen Architekturmodell . . . . .	98
5.2.2. Vom fachlichen Modell zum technischen Mapping . . . . .	100
5.2.3. Vom plattformunabhängigen Architekturmodell zum Annotationsmodell . . . . .	102
5.2.4. Vom plattformunabhängigen zum plattformabhängigen Architekturmodell . . . . .	103
5.2.5. Vom plattformabhängigen Architekturmodell zu Code . . . . .	104
5.3. Vorgehen . . . . .	106

<b>6. Modellgetriebene Datenintegration</b>	<b>109</b>
6.1. Persistenzadapter . . . . .	109
6.2. Persistenzadapter-Generierung mit SCORE . . . . .	110
6.3. Schema-Import . . . . .	110
6.4. Die Datenzugriffsschicht . . . . .	112
6.5. Die Service-Schicht . . . . .	117
6.5.1. Mappings . . . . .	121
6.6. Adapter-Generierung . . . . .	127
6.6.1. Der generierte Zielcode . . . . .	128
6.7. Zusammenfassung . . . . .	130
<b>7. Evaluierung der Ergebnisse</b>	<b>131</b>
7.1. Das Vorgehen . . . . .	131
7.2. Der Goal Question Metric (GQM)-Plan . . . . .	134
7.2.1. GQM-Plan für Wartbarkeit . . . . .	136
7.2.2. GQM-Plan für Performance . . . . .	139
7.3. Ergebnisse der Evaluierung . . . . .	141
7.3.1. Aufsetzen des Experiments . . . . .	141
7.3.2. Ergebnisse für Goal 1: Wartbarkeit . . . . .	141
7.3.3. Ergebnisse für Goal 2: Performance . . . . .	145
7.4. Einschränkungen und Validität der Evaluierung . . . . .	151
<b>8. Entwurfsentscheidungsunterstützung beim Entwurf von Integrationssystemen</b>	<b>153</b>
8.1. Best Practices und Anti-Patterns . . . . .	153
8.1.1. Übersicht . . . . .	153
8.1.2. Best Practices - allgemeine Erkenntnisse . . . . .	153
8.1.3. Best Practices - Entwurfsmuster . . . . .	154
8.1.4. Best practices für die verschiedenen Szenarien . . . . .	156
8.1.5. Anti-Patterns für die verschiedenen Szenarien . . . . .	158
8.2. Entwurfsentscheidungsmatrix . . . . .	160
<b>9. Fazit</b>	<b>165</b>
<b>A. Veröffentlichungen</b>	<b>169</b>
<b>Abbildungsverzeichnis</b>	<b>173</b>
<b>Tabellenverzeichnis</b>	<b>175</b>
<b>Literaturverzeichnis</b>	<b>183</b>



# 1. Einleitung

*Ralf Reussner, Ulrike Steffens, Niels Streekmann*

Das Projekt Modellgetriebene Integration von Informationssystemen (MINT) befasst sich mit der Definition und Validierung modellgetriebener Entwicklungsverfahren für die Integration bestehender heterogener betrieblicher Systeme. Der Fokus des Projekts liegt dabei auf betrieblichen Informationssystemen für KMU. Das entwickelte Verfahren soll KMU die Möglichkeit geben, ihre Softwaresysteme kosteneffizient an sich ändernde Geschäftsprozesse und neue Anforderungen, z. B. im eCommerce-Bereich, anpassen zu können. Software entwickelnden Unternehmen soll zudem eine Steigerung ihrer Flexibilität und Effizienz bei der Softwareentwicklung und damit ihrer Wettbewerbsfähigkeit ermöglicht werden.

Der im Projekt angestrebte modellgetriebene Ansatz hebt sich von bestehenden Ansätzen durch die Konzentration auf die Integration von bestehenden Systemen im Gegensatz zu einer vollständigen Neuentwicklung eines Softwaresystems ab. Ein weiterer Schwerpunkt des Projekts ist die Vereinfachung der Beschreibung von Softwaresystemen unter Einbindung von Fachexperten durch den Einsatz domänenspezifischer Sprachen und Verfahren zur geleiteten Erstellung von in diesen Sprachen spezifizierten Modellen. Bisherige modellgetriebene Verfahren verwenden zwar ebenfalls rechnerunabhängige Eingabemodelle als Ausgangspunkt der Entwicklung, diese sind jedoch häufig kaum abstrakter als Programm-Code und somit für Fachexperten nicht handhabbar.

Eine weitere technologische Basis modellgetriebener Entwicklung ist der Einsatz von Generatoren. Diese vereinfachen die Wiederverwendbarkeit von Modellen bei der Umstellung auf neue Plattformen und unterstützen die Integration von bestehenden Systemen durch die automatisierte Generierung von Adaptern. Konkret werden im Projekt zwei Szenarien unterstützt: (a) Die Nutzung des modellbasierten Ansatzes für die Integration verschiedener Systeme durch bestehende Infrastrukturen, wie z. B. SAPs „NetWeaver“ und (b) die Nutzung des modellbasierten Ansatzes für die Kopplung moderner objektorientiert modellierter Geschäftslogik mit bestehenden relationalen Datenbanksystemen.

Weitere Informationen zum MINT-Projekt sind auf der Projektwebseite <http://www.mint-projekt.de> zu finden.

## 1.1. Rahmenbedingungen

Dieser Abschnitt bietet eine kurze Darstellung zu den Rahmenbedingungen des MINT-Projekts.

### 1.1.1. Aufgabenstellung

In einer Umgebung sich rasch verändernder Anforderungen im eCommerce-Bereich und kürzer werdender Innovationszyklen ist es eine wichtige Eigenschaft heutiger betrieblicher Software, schnell und kosteneffizient an neue oder sich ändernde Geschäftsprozesse anpassbar zu sein. Die modellgetriebene Entwicklung versucht, dieser Forderung nach Flexibilität Rechnung zu tragen. Oft werden aber bestehende Systeme Anforderungen ausgesetzt, für die sie ursprünglich nicht entworfen wurden, wie zum Beispiel Web-Anbindungen im Bereich des sog. Business-to-business-eCommerce. Dies zeigt sich zum einen in über die Zeit gewachsenen Softwaresystemen, die die beschriebene Flexibilität nicht erreichen, und zum anderen in heterogenen Softwaresystemlandschaften, deren Datenhaltung und Arbeitsabläufe nicht integriert sind, obwohl dieses für die Geschäftsprozesse des Unternehmens vorteilhaft wäre. Gerade KMU stellt die Integration bestehender Software mit neuen Anwendungen vor oft kaum zu bewältigende finanzielle und technische Herausforderungen. Die Teilnahme der KMU an endkundenbezogenen oder unternehmensübergreifenden eCommerce-Abläufen wird damit verhindert, wodurch sich offensichtliche Wettbewerbsnachteile ergeben.

Das im MINT-Projekt entwickelte modellgetriebene Integrationsverfahren stellt insbesondere für KMU eine wesentliche Unterstützung dar: Zum einen ermöglicht es Softwareanwendern die kosteneffiziente Anbindung bestehender Software an neue flexible Plattformen; zum anderen können Software entwickelnde Unternehmen durch den Einsatz modellgetriebener Verfahren ihre Flexibilität und Effizienz bei der Softwareentwicklung und damit ihre Wettbewerbsfähigkeit steigern.

Die modellgetriebene Entwicklung versucht, der Forderung nach Flexibilität durch eine Intensivierung der Wiederverwendung mittels Generortechnologie Rechnung zu tragen. Bestehende modellgetriebene Ansätze konzentrieren sich bisher allerdings auf die Neuentwicklung von Softwaresystemen und vernachlässigen die Problematik der Integration und Evolution bestehender Systeme. Weiterhin kapseln heutige modellgetriebene Verfahren zwar technische Plattformspezifika, benötigen aber dennoch Eingabemodelle, die konzeptionell häufig kaum abstrakter sind als Programm-Code.

Das MINT-Projekt beschäftigt sich daher mit der Definition und Validierung eines modellgetriebenen Entwicklungsverfahrens zur Integration bestehender heterogener betrieblicher Informationssysteme. Dabei werden explizit musterbasierte, domänenspezifische Architektursprachen genutzt.

### 1.1.2. Voraussetzungen des Projekts

Das Projekt MINT wurde im Rahmen der Forschungsoffensive „Software Engineering 2006“ vom Bundesministerium für Bildung und Forschung gefördert. Projektträger des

Projekts war das Deutsche Zentrum für Luft- und Raumfahrt (DLR). Das Projektkonsortium bestand aus drei industriellen und drei akademischen Partnern. Im Folgenden werden die Partner vorgestellt und ihre Vorarbeiten und Aufgaben im Projekt skizziert.

Die **andrena objects ag** ist ein mittelständisches Softwarehaus, das sich überwiegend mit der Erstellung von Individuallösungen beschäftigt. Bei der Umsetzung der Projekte setzt andrena auf agile Entwicklungs- und Management-Methoden. Für den Projektkontext sind die umfangreichen Erfahrungen mit 3-Schichten-Architekturen betrieblicher Software zu nennen. Bei diesen Architekturen spielt eine performante und zugleich wartbare Abbildung umfangreicher objektorientierter Domänenmodelle der „Geschäftslogikschicht“ auf eine relationale Datenbank eine wichtige Rolle. Dabei wurden bereits umfangreiche Erfahrungen mit verschiedenen Abbildungsverfahren gesammelt. Als praxisnahes Evaluationssystem für die verschiedenen Kopplungstechniken im Arbeitspaket 3 (siehe Abschnitt 1.1.3) bringt andrena ein typisches Softwaresystem (siehe Kapitel 2.8) ein, bei dem die Problematik unterschiedlicher OR-Mapping-Technologien eine große Rolle spielte.

Mit knapp 1300 Mitarbeitern bietet die **Business Technology Consulting AG (BTC)** IT-Beratungsdienstleistungen aus einer Hand an. Das Leistungsspektrum ist sowohl betriebswirtschaftlich als auch organisatorisch und technisch auf die Kundenbedürfnisse aus unterschiedlichen Domänen zugeschnitten. Insbesondere werden u.a. große Individualsoftwaresysteme entwickelt, mit denen u. a. Kunden-, Vertrags- und Abrechnungsdaten für die Energieversorgung sowie die Telekommunikation verwaltet werden. Durch die gezielte Ausrichtung auf die individuell optimierten Geschäftsprozesse der Kunden bieten diese Individualsoftwaresysteme ihren Anwendern massive Wettbewerbsvorteile gegenüber aktuellen Standardlösungen. Diese Vorteile gilt es durch moderne Softwaretechnologien zu bewahren und auszubauen. Erste Experimente mit einem von der BTC selbst entwickelten Generator haben Potentiale aber auch Grenzen heutiger Ansätze aufgezeigt. Diese Erfahrungen flossen in das Projekt MINT ein, um darauf aufbauend effiziente, flexible und zukunftsfähige Methoden für die Entwicklung und Integration von Individualsoftwaresystemen (siehe Kapitel 5) zur Verfügung zu stellen.

Die **Delta Software Technology GmbH (DSTG)** stellt als mittelständisches Softwarehaus Werkzeuge und Methoden für die integrative Anwendungsentwicklung und -modernisierung bereit. Dies geschieht unter Berücksichtigung aktueller Standards und auf Basis moderner Generatortechnologie. Mit HyperSenses und SCORE Adaptive Bridges stehen Werkzeuge zur effizienten Umsetzung der im Projekt MINT definierten Ziele zur Verfügung (siehe Kapitel 2.5). DSTG ist Hersteller fortschrittlicher generativer Werkzeuge für Softwareentwicklung, -migration und Architekturintegration und verfügt über mehr als 30 Jahre Erfahrung in der Erstellung von Entwicklungs- und Integrationstechnologie für Enterprise-Class-Applikationen mit mehr als 750 Installationen. DSTG arbeitet eng mit führenden Universitäten und Standardisierungsgremien zusammen. Im ebenfalls vom BMBF geförderten Projekt PESOA (Process Family Engineering in Service-Oriented Applications) konnte bereits die Einsatzfähigkeit der modellgetriebenen Generator-Entwicklung und -Konfiguration demonstriert werden [24].

Im MINT-Projekt wurden von DSTG Methoden zur Datenintegration (siehe Kapitel 6) entwickelt.

1985 gründeten die Universität Karlsruhe und das Land Baden-Württemberg das **FZI Forschungszentrum Informatik**, mit dem Ziel, Forschungsergebnisse aus der Informatik der mittelständisch geprägten Wirtschaft des Landes zugänglich zu machen. Speziell in der Arbeitsgruppe von Prof. Dr. Ralf Reussner beschäftigt man sich intensiv mit Fragen der Architekturevaluation. Dabei werden insbesondere Fragen der Evaluation von Softwareperformanz und Skalierbarkeit untersucht [56, 55, 52, 53, 15, 4, 5, 49, 62, 61, 48, 63] und auch die Generierung von Adaptionen behandelt [5, 62, 63]. Weitere Arbeiten befassen sich mit der vergleichenden empirischen Evaluation von Performanz und Wartbarkeit von Datenbank-Kopplungstechniken der .NET-Plattform für betriebliche Anwendungen [67]. Im Rahmen dieser Untersuchungen konnten bereits umfangreiche methodische Erfahrungen gesammelt werden. Diese Vorarbeiten waren die Grundlage für die Evaluierung der in MINT entwickelten Methoden (siehe Kapitel 7 und 8).

Das 1991 gegründete „Oldenburger Forschungs- und Entwicklungsinstitut für Informatik- Werkzeuge und -Systeme“ **OFFIS** erforscht als An-Institut der Universität Oldenburg mit institutioneller Förderung durch das Land Niedersachsen neue Formen Computer-gestützter Informationsverarbeitung in Hard- und Softwaresystemen und setzt die Ergebnisse in anwendungsnahe Entwicklungen um. Speziell im OFFIS-Bereich „Betriebliches Informations- und Wissensmanagement“ wurden bereits zahlreiche Forschungs- und Industrieprojekte u. a. in den Themenbereichen der Architekturmodellierung [7], der musterbasierten Entwicklung und der Modellierung von Entwicklungsprozessen durchgeführt. Im Bereich der Migration und Integration betrieblicher Informationssysteme auf moderne 3-tier-Architekturen gibt es zahlreiche Projekte und Erfahrungen, die sich in einem Muster zur Migration dieser Systeme niedergeschlagen hat [27], welches auf serviceorientierte Architekturen angewandt wurde [68]. Im Projekt MINT war OFFIS als Koordinator tätig und bearbeitete inhaltlich hauptsächlich die Themenbereiche „Domänenspezifische Modellierung“ (siehe Kapitel 3) und „Modellgetriebene Prozessintegration“ (siehe Kapitel 5).

In der Abteilung Lernende und Kognitive Systeme der **Universität Oldenburg** werden seit 1985 wissensbasierte CBT-Programme, sogenannte Intelligent Problem Solving Environments (IPSE) [42], für unterschiedliche Wissensgebiete entwickelt, u. a. für den Softwareentwurf. Im ebenfalls im Rahmen der Forschungsoffensive „Software-Engineering 2006“ geförderte Projekt InPULSE befasste sich die Abteilung mit der ontologiebasierten Anwendung hierarchisierter Entwurfsmuster im Forward Engineering. Entwurfsmuster werden dabei zunächst mit wissensbasierten Methoden entwickelt und können dem Anwender dann über ein Assistenzsystem vorgeschlagen werden. Zur Unterstützung des Wissenserhebungsprozesses wurde das Werkzeug KARaCAs (Knowledge Acquisition with Repertory Grid Technique [60] and Formal Concept Analysis [21]) entwickelt. Auf Basis der extrahierten Mustermerkmale wurde eine Musterontologie erstellt und das Assistenzsystem zur Unterstützung des Forward Engineering entwickelt [43]. In MINT wurden diese Erfahrungen in die Entwicklung einer domänenspezifischen Sprache für die Prozessintegration (siehe Kapitel 3) und die Umsetzung



wissensintensiver Prozesse (siehe Kapitel 4) übernommen.

### 1.1.3. Planung und Ablauf des Projekts

Abbildung 1.1 zeigt eine Übersicht der Arbeitspakete des MINT-Projekts. Das Projekt gliedert sich in insgesamt fünf Arbeitspakete von denen drei (AP1, AP2 und AP3) technischer Natur sind und zwei (AP4 und AP5) die organisatorische Seite des Projekts repräsentieren. Im Folgenden werden die Inhalte dieser Arbeitspakete kurz skizziert und die Ergebnisse der Arbeitspakete mit den Kapiteln dieses Berichts verknüpft.



Abbildung 1.1.: Arbeitspakete des MINT-Projekts

Arbeitspaket 1 (Architektur-Metamodell und domänenspezifische Muster) zielt auf eine allgemeine Verbesserung des modellgetriebenen Ansatzes ab. Hierzu wurde eine modulare domänenspezifische Sprache zur Beschreibung von Integrationsprozessen in unterschiedlichen Anwendungsdomänen entwickelt. Diese Sprache (MINT-XL) wird ausführlich in Kapitel 3 beschrieben. Des Weiteren befasst sich das Arbeitspaket mit der geleiteten Entwicklung von Beschreibungen in dieser Sprache und der Überleitung von Modellen dieser Sprache in darunter liegende, implementierungsnähere MDA-Modelle. Die geleitete Entwicklung von Modellen wird beispielhaft in Kapitel 4 beschrieben. Die Arbeiten zur Überleitung in implementationsnähere Modelle sind in das in Kapitel 5 beschriebene Vorgehensmodell eingegangen.

Arbeitspaket 2 (Modellgetriebene Integration von Altsystemen) befasst sich gezielt mit einer Umsetzung des modellgetriebenen Ansatzes in bereits existierenden Software-systemlandschaften. Als zwei typische konkrete Einsatzszenarien greift es dabei die Integration von Individual-Software mit betrieblicher Standardsoftware sowie die Kopplung objektorientierter Geschäftslogik an existierende relationale Legacy-Datenbanken heraus. Zur Integration von Individual- und Standardsoftware wurde das bereits erwähnte, in Kapitel 5 beschriebene Vorgehensmodell sowie eine prototypische Umsetzung dieses Vorgehensmodells erarbeitet. Die Arbeiten zur Kopplung an existierende Datenbanken werden in Kapitel 6 beschrieben.

Ziel von Arbeitspaket 3 (Validierung der Ergebnisse) ist eine Validierung modellgetriebener Ansätze insbesondere anhand nicht-funktionaler Faktoren wie Wartbarkeit, Skalierbarkeit oder Performance. Es diene einerseits der Validierung der konkreten Ergebnisse innerhalb des MINT-Projekts; die entwickelten Verfahren sind jedoch hinreichend allgemein, so dass sie sich auch für die Validierung von Ansätzen außerhalb des Projekts eignen. Die Beschreibung des Evaluierungsvorgehens sowie der Ergebnisse der Evaluation sind in Kapitel 7 zu finden. Kapitel 8 stellt zudem Best Practices und Muster vor, die aus der Evaluation hervorgegangen sind und zur Unterstützung zukünftiger Design-Entscheidungen dienen können.

Die technischen Arbeitspakete 1. - 3. werden auf organisatorischer Seite durch zwei weitere Arbeitspakete ergänzt: eines zur Sicherung der Nachhaltigkeit der Projektergebnisse (Arbeitspaket 4), die im Kontext des MINT-Projekts als besonders relevant erachtet werden, und ein weiteres zur allgemeinen Koordination des Projekts (Arbeitspaket 5).

#### **1.1.4. Wissenschaftlicher und technischer Stand**

Die Abbildbarkeit der Anforderungen des hochspezialisierten deutschen Mittelstands in entsprechende Softwarelösungen ist eine Grundvoraussetzung für dessen Überlebensfähigkeit. In der Vergangenheit lag der Fokus vieler Unternehmen auf der Einführung von Standardsoftwaresystemen, die auf Basis von Konfigurationsmechanismen an die Geschäftsprozesse (siehe [38]) angepasst wurden. Diese Konfigurationsmechanismen haben eine begrenzte Ausdruckskraft. Um dennoch die Anforderungen abbilden zu können, werden deshalb zunehmend kleinere branchen- und/oder betriebstypische Anwendungen oder individuell entwickelte Softwaresysteme integriert. Insbesondere die Notwendigkeit der Integration von Zusatzlösungen und die individuelle Softwareentwicklung bewegen die großen Standardsoftwarehersteller (SAP, Oracle, Siebel) dazu, ihre Systeme auf Basis moderner Standards (Java, Web-Services, ...) zu öffnen.

Die nächste große Herausforderung, die Integration der zugekauften und die Entwicklung und Integration der individuellen Softwarelösungen möglichst effizient zu gestalten, bleibt jedoch bestehen. Modellgetriebene Ansätze scheinen dafür geeignet zu sein. Sie zielen auf eine Steigerung der Anpassbarkeit von Software an sich wandelnde Anforderungen. Dies soll erreicht werden durch den Einsatz von Generortechnologie, wie er beispielsweise für den speziellen Bereich der Software-Produktlinien im BMBF-geförderten Projekt PESOA (Process Family Engineering in Service-Oriented Applications) von einem Partner bereits erforscht wurde (siehe [23]). Ergänzt wird dieser generative Ansatz durch die konsequente Trennung zwischen Modellierung der Architektur mit der fachlichen Logik (beschrieben durch das abstrakte, auf die Beschreibung der Anwendungsdomäne zielende „Computation Independent Model“ (CIM) sowie das konkretere Verhalten und Problemlösungsverfahren enthaltende „Platform Independent Model“ (PIM)) auf der einen Seite und den plattformspezifischen Eigenschaften auf der anderen Seite, wobei letztere durch einen Generator und seine Konfiguration verborgen werden. Bei der modellgetriebenen Entwicklung wird dem Softwareentwickler ein Vorgehensmodell an die Hand gegeben, welches beschreibt, wie (teilweise unter

Anwendung von Transformationsregeln) aus den Zielvorgaben eines Kunden ein CIM, dann ein ablauffähiges PIM sowie ablauffähige sog. „Platform Specific Models“ (PSM) erstellt werden können. Die PSM können dann anschließend in Code transformiert werden.

Im Bereich der Softwareentwicklung für betriebliche Informationssysteme werden die antizipierten Vorteile der modellgetriebenen Entwicklung heute noch nicht erreicht, da (a) sich die meisten bestehenden modellgetriebenen Ansätze auf die Neuentwicklung von Softwaresystemen konzentrieren und die gerade bei betrieblichen Softwareanwendungen wichtige Problematik der Evolution und Integration bestehender Systeme vernachlässigen und (b) heutige modellgetriebene Verfahren zwar technische Plattformspezifika kapseln, aber dennoch Eingabemodelle benötigen, die konzeptionell häufig kaum abstrakter sind als Programm-Code und damit nur ähnlich kostenintensiv zu erstellen sind. Daher werden statt UML-basierter MDA-Sprachen zunehmend domänenspezifische Sprachen diskutiert (siehe [17, 33, 66]), da diese Sprachen einfacher zu handhaben sind und stärker an den Besonderheiten ihrer branchenspezifischen Einsatzgebiete orientiert werden können. Gerade aber die CIM-Formulierung zur Darstellung kombinierter Geschäfts- und Problemlöseprozesse ist zurzeit mehr Kunst als ingenieurmäßig geplantes Vorgehen. Hier kann der Transfer musterbasierter Vorgehensweisen, wie sie z. B. im BMBF-geförderten Projekt InPULSE von einem der Partner erarbeitet wurden (siehe [43]), auf die CIM-Ebene eine erste verbessernde Maßnahme darstellen.

Das Projekt MINT ist neuartig in seiner Verknüpfung der Herausforderungen in den Bereichen (a) domänenspezifischer Sprachen und (b) der Integration bestehender Softwaresysteme mit der Fokussierung auf betriebliche Informationssysteme.

### **1.1.5. Zusammenarbeit mit anderen Stellen**

Im Verlauf des Projekts wurde mit zahlreichen Stellen außerhalb des Projekts zusammengearbeitet. So gab es eine Zusammenarbeit mit den ebenfalls im Rahmen der Forschungsoffensive „Software Engineering 2006“ geförderten Projekten OrViA und TransBS. Im Rahmen dieser Zusammenarbeit wurden Ergebnisse und Vorgehensweisen sowie Wissen über verwendete Werkzeuge ausgetauscht. Zudem wurde gemeinsam ein Workshop im Rahmen der Konferenz „Software Engineering 2008“ organisiert, um die Ergebnisse auch weiteren Interessierten aus Wissenschaft und Industrie vorzustellen und einen weiteren Erfahrungsaustausch herbeizuführen.

Weiterhin gab es Kontakte zu einer Reihe ebenfalls am Themenfeld „Modellgetriebene Softwareentwicklung“ interessierter Unternehmen und Forschungseinrichtungen im Rahmen von Arbeitskreisen der Gesellschaft für Informatik, insbesondere des Arbeitskreises Modellgetriebene Software-Architektur. Schwerpunkte dieser Arbeit waren Grundlagen modellgetriebener Softwareentwicklung und der Einsatz modellgetriebener Softwareentwicklung in der industriellen Praxis.

## 1.2. Übersicht des MINT-Ansatzes

Abbildung 1.2 zeigt eine Übersicht des in MINT entwickelten modellgetriebenen Integrationsansatzes. Im Folgenden werden die wichtigsten Punkte kurz beschrieben, um das Zusammenwirken der im Projekt bearbeiteten Aufgaben zu verdeutlichen. Die Abbildung zeigt die typischen Stufen des Vorgehens der Model-Driven-Architecture (MDA) der OMG. Dieses Vorgehen unterscheidet zwischen einem fachlichen Modell (Computation Independent Model - CIM) sowie zwei Stufen technischer Modelle. Diese bestehen aus plattformunabhängigen Modellen (Platform Independent Model - PIM) und plattformabhängigen Modellen (Platform Specific Model - PSM) der Anwendung. Aus letzteren lässt sich wiederum Code generieren. Dieses Vorgehen wurde auch im MINT-Ansatz umgesetzt und an entscheidenden Stellen an die Aufgabe der Integration angepasst.

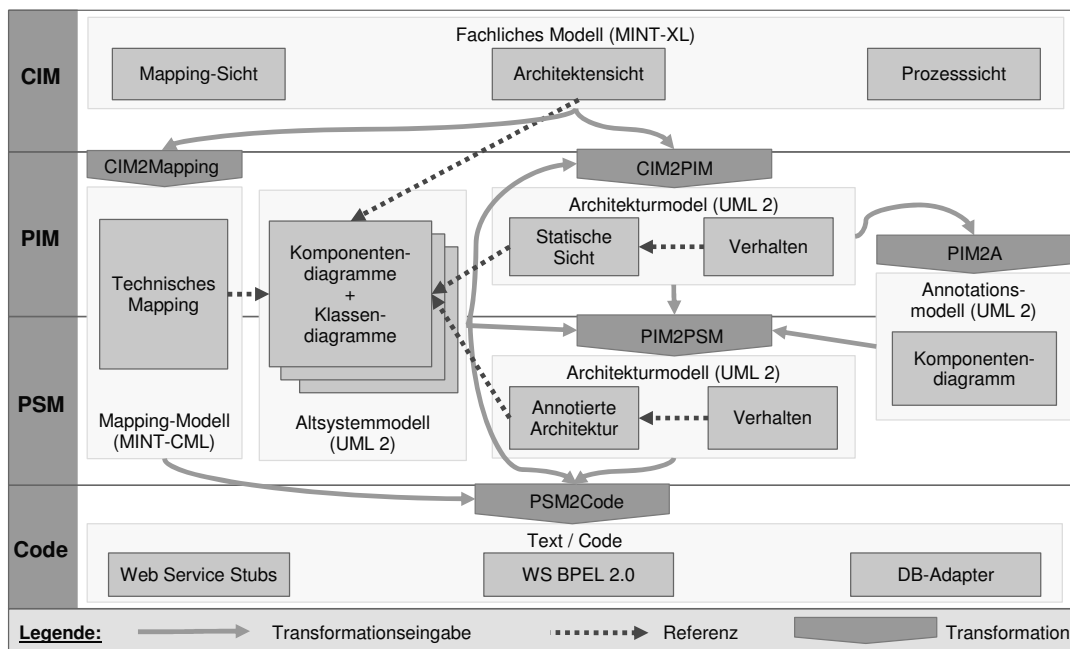


Abbildung 1.2.: Übersicht zum MINT-Ansatz

Auf der fachlichen Ebene wurde das klassische MDA-Vorgehen so erweitert, dass domänenspezifische Sprachen verwendet werden, die zum einen eine gemeinsame Grundlage für die Kommunikation mit Fachexperten legen und zum anderen aber auch darauf ausgerichtet sind, dass entsprechende Modelle automatisiert in technische Architekturmodelle überführt werden können. Die Arbeiten auf dieser Ebene waren das Kernstück des in 1.1.3 beschriebenen Arbeitspakets 1.

Auf der Ebene der technischen Architekturmodellierung wurden die Architekturmodelle aus dem Forward-Engineering, die auch Teil des MDA-Vorgehens sind, durch weitere Modelle ergänzt. Dabei sind vor allem das Altsystemmodell und das Mapping-

Modell zu nennen, die dazu dienen, die Prozesse und Datenmodelle des Forward-Engineering auf die Gegebenheiten der Altsysteme abzubilden. Das Altsystemmodell beschreibt dabei die Komponenten und Schnittstellen der Altsysteme, während das Mapping-Modell eine Abbildung zwischen diesen Schnittstellen beschreibt.

Aus den plattformspezifischen Architekturmodellen lassen sich wiederum direkt Artefakte generieren, die in diesem Fall zur Integration der bestehenden Altsysteme dienen. Auf der Seite der Prozessintegration sind dies Code für Webservices, die zur Anbindung von Altsystemen an moderne Systeme dienen und technische Prozessbeschreibungen, z. B. in der Prozessbeschreibungssprache BPEL, die direkt von Prozessmanagementsystemen ausgeführt werden können. Auf der Seite der Datenintegration lassen sich ebenfalls Datenservices zur Integration in die entsprechenden Prozesse, sowie Datenbankadapter, die die Abbildung zwischen den Datenmodellen der Services und den Schemata der bestehenden Datenbanken übernehmen, generieren. Die Modellierung und die Umsetzung der entsprechenden Transformationen und Generatoren war der Inhalt von Arbeitspaket 2. Dabei teilte sich das Arbeitspaket in die Szenarien Modellgetriebene Projektintegration und Modellgetriebene Datenintegration.

Aufgabe des Arbeitspakets 3 war die Evaluation des in Abbildung 1.2 illustrierten modellgetriebenen Ansatzes. Dabei wurde als Hauptevaluierungsszenario die modellgetriebene Datenintegration mit Schwerpunkt auf der Erstellung von Datenbankadaptern gewählt. Der MINT-Ansatz wurde mit verschiedenen anderen manuellen und generativen Verfahren verglichen. Aus den Ergebnissen lässt sich erkennen, unter welchen Kriterien welcher der untersuchten Ansätze bevorzugt einsetzbar ist. Aus diesen Erkenntnissen wurden Best Practices und Anti-Patterns zur Unterstützung von Entwurfsentscheidungen abgeleitet.

### **1.3. Aufbau des Dokuments**

Dieser Projektbericht ist wie folgt gegliedert. Kapitel 2 beschreibt Grundlagen des Projekts. Dabei werden sowohl konzeptuelle Grundlagen zu modellgetriebener Software-Entwicklung und Integration beschrieben als auch die verwendeten Werkzeuge und die in der Evaluierung verwendeten Szenarien. In Kapitel 3 wird mit der MINT-XL eine domänenspezifische Sprache für die Integration vorgestellt. Es folgen in Kapitel 4 Konzepte zur Erstellung eines fachlichen Modells am Beispiel wissensintensiver Prozesse. Die Kapitel 5 und 6 erläutern die im Projekt erarbeiteten Konzepte zur modellgetriebenen Integration auf Prozess- und Datenebene. Die Kapitel 7 und 8 geben einen Überblick über die Evaluierung der entwickelten Methoden und daraus resultierende Erkenntnisse zur Unterstützung von Entwurfsentscheidungen. Abschließend enthält Kapitel 9 eine Zusammenfassung und einen kurzen Ausblick auf zukünftige Arbeiten. Im Anhang A werden alle im Rahmen des Projekts entstandenen Veröffentlichungen aufgelistet.



## 2. Grundlagen

*Heiner Feislachen, Cord Giese, Thomas Kühn,  
Jing Shui, Niels Streekmann, Jochen Winzen*

Dieses Kapitel beschreibt allgemeine Grundlagen des Projekts sowie erste vorbereitende Arbeiten. Des Weiteren werden in den Abschnitten 2.7 und 2.8 die zur Evaluierung verwendeten Systeme kurz beschrieben.

### 2.1. Modellgetriebene Software-Entwicklung

In vielen in der Praxis üblichen Softwareentwicklungsprozessen dienen Modelle nur zur Spezifikation und Dokumentation. Jede Erweiterung oder Änderung der Software findet im Quelltext statt und findet selten genug den Weg in die Dokumentation. Von der Object Management Group (OMG) wurde mit der Model Driven Architecture (MDA) [40] ein Standardisierungsvorschlag zur Modellgetriebenen Softwareentwicklung vorgelegt. Die MDA sieht vor, dass die Modellierung einer Software nicht nur zur Spezifikation und Dokumentation genutzt wird, sondern dass auf einer fachlichen Ebene ein Modell erstellt wird aus welchem nach mehreren Transformationen Quelltext für eine Zielplattform generiert wird. Zu diesem Zweck werden in der MDA folgende Modellebenen beschrieben, die in Abbildung 2.1 dargestellt sind:

- Das CIM (Computation Independent Model) ist ein Modell, das die fachliche Sicht auf ein Softwaresystem beschreibt. Mit dem CIM wird von fachlichen Anwendern gemeinsam mit Softwarearchitekten definiert was eine Software leistet, aber nicht wie die Software dieses leistet. Modelliert wird in einer dem fachlichen Anwender verständlichen Sprache.
- Das PIM (Platform Independent Model) stellt plattformunabhängig die Funktionalität der zu erstellenden Software dar. In der MDA wird mit dem Begriff der Plattform eine weitere Terminologie eingeführt. Eine Plattform ist als abgeschlossene Softwarekomponente oder Technologie definiert, die Funktionalität über Schnittstellen bereitstellt, ohne dass Wissen über die Implementierung der Plattform vorliegt. Die Repräsentation einer Plattform auf Modellebene wird Plattformmodell genannt.
- Das PSM (Platform Specific Model) ist eine plattformabhängige Beschreibung des Softwaresystems und stellt damit eine Verfeinerung des PIMs dar.

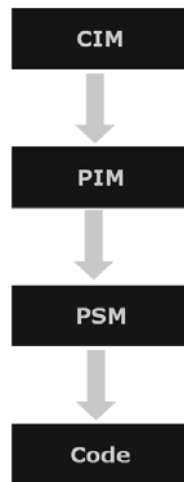


Abbildung 2.1.: MDA-Überblick

- Auf der untersten Ebene dieser Einordnung befindet sich der generierte Code der Anwendung.

Neben den Standardisierungsbestreben der OMG gibt es zahlreiche weitere Ansätze modellgetriebene Softwareentwicklung in der Praxis umzusetzen, die sich mehr oder weniger von den Vorgaben der MDA unterscheiden. Ein besonders im deutschsprachigen Raum viel beachteter Ansatz wird in [66] beschrieben. Einen Einstieg und Überblick über die wichtigsten Punkte, die beim Einsatz modellgetriebener Softwareentwicklung beachtet werden sollten, liefert [54, Kapitel 5].

In der Praxis ist diese Modellaufteilung nicht immer 1:1 wiederzufinden, die Grundprinzipien lassen sich aber übertragen. Neben unterschiedlichen Modellebenen werden auch zwei Arten von Modelltransformationen unterschieden. Zum einen Modell-zu-Modell-Transformationen, wie sie z. B. beim MDA-Vorgehen von CIM zu PIM und von PIM zu PSM durchgeführt werden. Dabei werden die Elemente des Ursprungsmodells in Elemente des Zielmodells überführt. Zum anderen Modell-zu-Text-Transformationen, die z.B. zur Generierung von Quellcode eingesetzt werden. Abbildung 2.1 zeigt neben den vier Ebenen der MDA auch die Transformationsrichtung zwischen den Ebenen. Weitere Details zu Modelltransformationen finden sich auch in [54, Kapitel 5.2]

Bei der modellgetriebenen Softwareentwicklung werden Änderungen an der Software im Idealfall nicht mehr im Quelltext durchgeführt, sondern auf Modellebene. Danach werden die Modelltransformationen durchlaufen und die Software erneut generiert. Es handelt sich also – auch wenn es in Abbildung 2.1 den Anschein hat – nicht um einen Wasserfallprozess sondern im Gegenteil um einen iterativen Prozess, der auch mit agilen Entwicklungsmethoden vereinbar ist.

Vorteile der modellgetriebenen Softwareentwicklung liegen u. a. darin, dass durch den hohen Automatisierungsgrad der Softwaregenerierung und die gute Wiederver-



wendbarkeit Aufwand und Kosten stark reduziert werden können. Einmal erstellte Modelle können auch Jahre später auf der Grundlage neuer Technologien für den Softwareherstellungsprozess genutzt werden. Die fachlichen Aspekte einer Software und die technische Realisierung können mit Hilfe modellgetriebener Softwareentwicklung getrennt voneinander entwickelt werden. Zudem wird durch diese Trennung und die Wiederverwendung von Modelltransformationen die Qualität der erstellten Software erhöht.

## 2.2. Integration von Softwaresystemen

Modellgetriebene Softwareentwicklung und auch der oben beschriebene MDA-Ansatz zielen bisher in erster Linie auf die Neuentwicklung von Software-Systemen ab. Die Integration von bestehenden Systemen wird in den modellgetriebenen Entwicklungsprozessen jedoch nicht berücksichtigt. Im Folgenden werden die Grundlagen für die in MINT vorgenommene Erweiterung des MDA-Ansatzes für die Integration bestehender Systeme und ihr Einfluss auf den Entwicklungsprozess beschrieben. Wie in Abbildung 2.2 zu sehen ist, ist es dazu notwendig auf allen Modellebenen die Einflüsse der bestehenden Systeme zu berücksichtigen. Diese grobgranulare Sicht wird in Kapitel 5 detaillierter beschrieben.

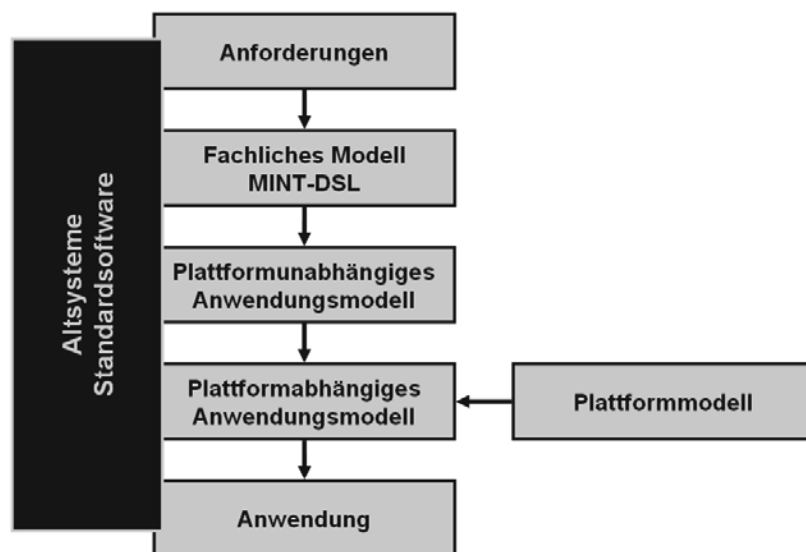


Abbildung 2.2.: Erweiterter MDA-Ansatz

Kern dieses Kapitels ist es die wissenschaftlichen Grundlagen, die von der Beschreibung des Dublo-Musters und vom BALES-Ansatz gebildet werden, in das MINT-Vorgehensmodell zu integrieren. In den folgenden Abschnitten werden diese Grundlagen zunächst erläutert und daraufhin ihre Relevanz für und ihre Einbettung in das Vorgehensmodell diskutiert.

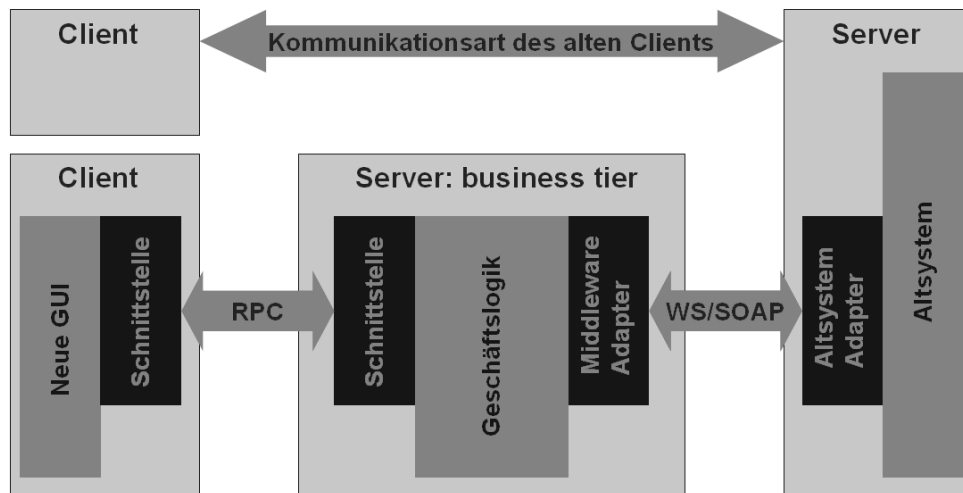


Abbildung 2.3.: Das Dublo-Muster [27]

### 2.2.1. Das Dublo-Muster

Das Dublo-Muster (DUal Business LOGic) wurde im Rahmen eines Projekts des OFFIS mit der Abteilung Software-Engineering der Universität Oldenburg und der KDO (Zweckverband Kommunale Datenverarbeitung Oldenburg) beschrieben [27]. Weitere Details zur technischen Umsetzung des Musters bei der KDO finden sich zudem in [68].

Kern des Dublo-Musters ist die Wiederverwendung von Altsystemen in modernen Mehrschichtenarchitekturen. Dazu wird der Weg der sogenannten sanften Migration verwendet. Sanfte Migration bedeutet, dass zunächst die Funktionen des Altsystems in die neue Architektur eingebunden und dann Schritt für Schritt migriert werden. Dieses Vorgehen bietet einige Vorteile im Gegensatz zu anderen Migrationsstrategien. Diese Vorteile werden in [27] näher erläutert.

Namensgeber des Dublo-Musters ist die Tatsache, dass beim Vorgehen der sanften Migration Erweiterungen des Systems nicht mehr im noch laufenden Altsystem vorgenommen werden müssen, sondern schon in der Zielumgebung der Migration unter Verwendung der bisherigen Funktionalität des Altsystems implementiert werden können. Das führt dazu, dass die Geschäftslogik des Gesamtsystems an zwei Stellen implementiert ist. Bei der schrittweisen Migration der Komponenten des Altsystems können diese dann wieder zusammengeführt werden, so dass das Ergebnis des gesamten Migrationsprozesses eine klassische Mehrschichtenarchitektur ist.

Die Kopplung zwischen dem Altsystem und der neuimplementierten Geschäftslogik kann dabei z. B. durch Webservices erfolgen. Abbildung 2.3 zeigt eine Übersicht des Dublo-Musters mit einer solchen Kopplung.

Da in MINT das Hauptaugenmerk auf der Integration von Altsystemen liegt, werden an dieser Stelle, wie auch in [68] nur die Integrationsaspekte des Musters berücksich-

sichtigt und der Migrationsanteil vernachlässigt. Bei der Integration auf Geschäftsprozessebene geht es darum, dass die Aktionen der Geschäftsprozesse, die von Softwaresystemen durchgeführt werden sollen, auf die entsprechenden Funktionen dieser Systeme abgebildet werden müssen.

In den letzten Jahren haben sich serviceorientierte Architekturen als eine wichtige Grundlagen für die Umsetzung von Geschäftsprozessen und die Integration von Softwaresystemen herausgestellt. Die technische Umsetzung geschieht dabei meistens über Webservices. In der Beschreibung des Dublo-Musters finden sich zahlreiche Grundlagen dafür, wie die Adaption von Altsystemen mittels Webservices umgesetzt werden kann und was dabei berücksichtigt werden muss. Außerdem zeigt das Muster einen Weg auf, wie verschiedenen Systeme zusammenarbeiten können, um eine bestimmte Funktionalität zu erfüllen.

Damit eignet sich das Dublo-Muster auch für die Integration im Kontext des MINT-Vorgehensmodells. Für die Verwendung dieser Techniken in MINT müssen allerdings noch weitere Punkte geklärt werden. Es fehlen z. B. Angaben dazu wie Aktionen eines Geschäftsprozesses auf von Softwaresystemen angebotene Webservices abgebildet werden können. Eine Methode zur Lösung dieser Aufgabe bietet der BALES-Ansatz, der im nächsten Abschnitt beschrieben wird. Zudem erfolgt die Erstellung der Webservices im Dublo-Kontext mit herkömmlichen Entwicklungsmethoden. In MINT werden auch Ansätze erforscht, wie hier modellgetriebene Methoden mehr Effizienz und mehr Flexibilität gegenüber sich ändernden Geschäftsprozessen bieten können. Dieser Punkt wird in Kapitel 5 vertieft.

### **2.2.2. Der BALES-Ansatz**

Der BALES-Ansatz (Business Application to LEgacy Systems) beschäftigt sich mit dem Zusammenspiel zwischen Geschäftsprozessen und Altsystemen bei der Integration der Altsysteme mit modernen Anwendungen, die aktuelle Geschäftsprozesse unterstützen. Dabei werden Reverse-Engineering von Services, die die Funktionalität, die die Altsysteme anbieten, kapseln, und Forward-Engineering der Services, die der Geschäftsprozess erfordert, miteinander verbunden. Das Reverse-Engineering entspricht dabei der Entwicklung von Adaptern für Altsysteme, wie sie bereits im vorigen Kapitel bei der Vorstellung des Dublo-Musters beschrieben wurden. Auch in der Beschreibung des BALES-Ansatzes werden dabei als technische Umsetzung Webservices verwendet [31, 29].

Das Forward-Engineering behandelt die Entwicklung von fachlichen Services, die aus der Beschreibung von Aufgaben in den Geschäftsprozessen, also aus den Anforderungen an das integrierte System heraus entstehen. Bei der Umsetzung dieses Systems müssen nun die fachlichen Services auf die durch die Altsysteme zur Verfügung stehenden technischen Services abgebildet werden. Abbildung 2.4 zeigt diese Abbildung als sogenanntes Linking. Dieses ist der wissenschaftliche Kernpunkt des BALES-Ansatzes. Die im Rahmen von Forward- und Reverse-Engineering entwickelten Services werden als gegeben angenommen. Hier besteht ein Unterschied zum MINT-Vorgehensmodell, in dem auch Fälle, in denen noch keine Services zur Verfügung stehen, behandelt

werden sollen.

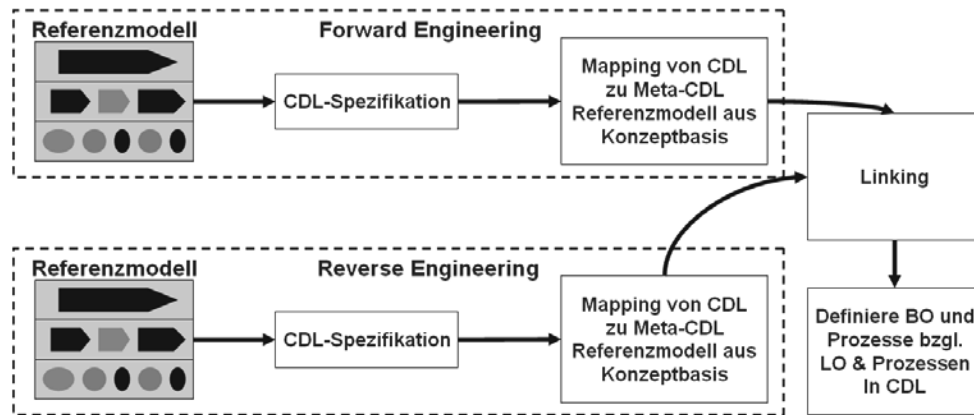


Abbildung 2.4.: Der BALES-Ansatz [30]

Das Linking im BALES-Ansatz besteht aus zwei Teilen. Das sind zum einen das Matching der Spezifikationen der fachlichen und der technischen Services und zum anderen die Entwicklung von Adaptern um Mismatches zwischen fachlichen und technischen Services auszugleichen. Detaillierte Ausführungen zu beiden Aktivitäten finden sich in [29].

In [28] wird bereits eine Einordnung des BALES-Ansatzes in das MDA-Vorgehensmodell vorgenommen. Dabei werden im Rahmen des Forward-Engineerings ebenfalls fachliche Geschäftsprozesse im CIM beschrieben, aus denen ein PIM erzeugt wird. Andersherum werden im Rahmen des Reverse-Engineering aus dem Altsystem ein PSM sowie ebenfalls ein PIM gewonnen. Das Linking findet dann auf PIM-Ebene statt. Abbildung 2.5 zeigt dieses Vorgehen.

Das Vorgehen im Forward-Engineering ähnelt dabei dem MINT-Vorgehensmodell. Ein großer Unterschied liegt darin, dass im MINT-Vorgehen bereits auf CIM-Ebene (entweder durch Fachexperten oder als technische Annotation durch den Software-Architekten) Informationen zur Zuordnung von Aktionen des Geschäftsprozesses zu Services der Altsysteme vorgenommen werden können. Auch im Reverse-Engineering gibt es einige Unterschiede, die in Kapitel 5 näher erläutert werden. In [28] werden allerdings über diese konzeptuellen Betrachtungen hinaus keine Details zur Umsetzung gegeben.

Der BALES-Ansatz behandelt sehr detailliert die Abbildung von Aktionen bzw. Services in Geschäftsprozessen auf Services der Altsysteme. Was jedoch nicht behandelt wird, ist die Umsetzung des Zusammenspiels bzw. der Orchestrierung dieser Services und damit die Erstellung eines lauffähigen Systems. In MINT wird auch die Orchestrierung auf technischer Ebene durch Modelltransformationen und Generierung entsprechenden Codes aus dem vom Fachexperten beschriebenen Modell auf CIM-Ebene heraus ablauffähig erzeugt. Im Bereich des Matchings wird hingegen im MINT-Vorgehensmodell weniger auf die semi-automatisierte formale Erkennung von

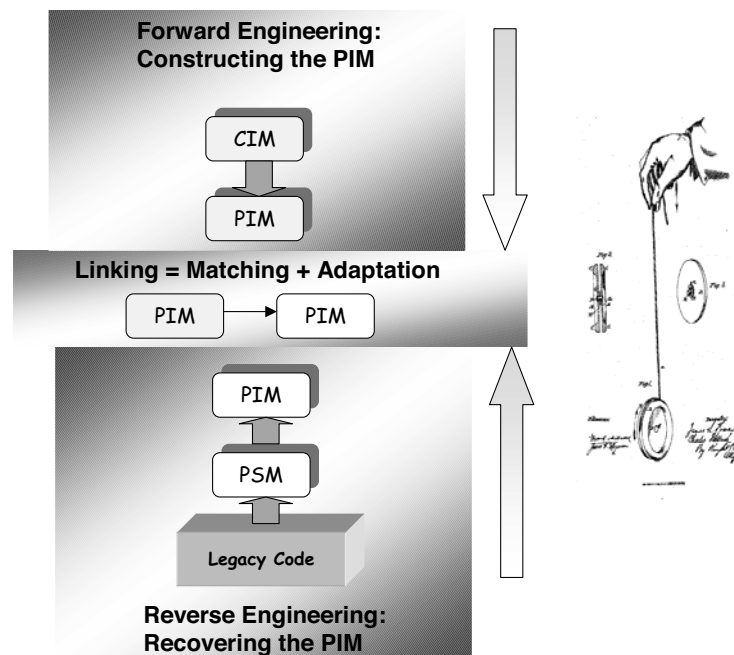


Abbildung 2.5.: Der BALES-Ansatz und MDA [28]

Ähnlichkeiten gesetzt, sondern schon auf CIM-Ebene durch die Hinzunahme entsprechender fachlicher Informationen die Abbildung von fachlichen auf technische Services unterstützt.

### 2.2.3. Serviceorientierte Architekturen

Serviceorientierte Architektur (SOA) ist ein Architekturkonzept, das die Geschäftsarchitektur eines Unternehmens mit der Softwarearchitektur verbindet. Dabei wird auf einem hohen Abstraktionsniveau die Integration von Softwaresystemen umgesetzt. Eine allgemeine Einführung zu SOA findet sich in [54, Kapitel 6].

In MINT liegt der Fokus auf der Umsetzung einer SOA und dem Zusammenkommen von fachlichen und technischen Sichten. Dabei sind serviceorientierte Architekturen nur ein Szenario für den MINT-Ansatz, das sich allerdings in der Praxis mehr und mehr verbreitet und bei dem die in MINT betrachtete fachliche Prozesssicht ebenfalls besonders im Vordergrund steht. Die fachliche Sicht in MINT unterstützt Fachexperten dabei, die Services einer SOA in Prozessen zu kombinieren und notwendige Anpassungen der Schnittstellen von Services in Form von Mappings vorzunehmen.

## 2.3. Modellierungswerkzeuge

Im Folgenden werden kurz die bei der prototypischen Umsetzung des MINT-Ansatzes für die Prozessintegration verwendeten Modellierungswerkzeuge vorgestellt.

### 2.3.1. Eclipse Modeling Framework (EMF)

Das Eclipse Modeling Framework ist ein Werkzeug zur Definition von Metamodellen für die modellgetriebene Softwareentwicklung. Aus den beschriebenen Metamodellen lassen sich direkt eine Java-Implementierung des Metamodells sowie ein einfacher Baumeditor für die Bearbeitung der auf den Metamodellen basierenden Modelle ableiten. Zur Metamodellierung stellt EMF eine Implementierung des Metametamodellstandards EMOF zur Verfügung.

EMF erlaubt damit die Definition domänenspezifischer Sprachen und stellt eine technische Infrastruktur zur Erstellung und Bearbeitung von Modellen in diesen Sprachen zur Verfügung. EMF setzt sich momentan in der Praxis mehr und mehr als Basis für Modellierungswerkzeuge und als Standard für den Austausch von Modellen zwischen Modellierungs- und Transformationswerkzeugen durch. Weiterführende Informationen finden sich unter: <http://www.eclipse.org/modeling/emf/> (Letzter Zugriff: 15.12.2008)

### 2.3.2. Graphical Modeling Framework (GMF)

Das Graphical Modeling Framework baut auf EMF auf und bietet die modellgetriebene Entwicklung graphischer Editoren für in EMF-Metamodellen beschriebene domänenspezifische Sprachen. Die Basis für die Generierung eines Editors mit GMF bilden das entsprechende EMF-Metamodell sowie weitere Modelle, mit denen sich die graphische Darstellung von Elementen sowie die zu verwendenden Bearbeitungswerkzeuge, z. B. für das Hinzufügen neuer Modellelemente, beschreiben lassen. Die Editoren lassen sich als Eclipse-Plugins oder auch als eigenständige Applikation mit Hilfe der Eclipse-Rich-Client-Plattform erstellen. Einfache Editoren lassen sich mit GMF zu 100% modellgetrieben erstellen. Für spezielle Anforderungen sowie graphische Aufwertungen der Editoren sind allerdings in der aktuellen Version noch manuelle Änderungen an Templates oder Ergänzungen zum generierten Code notwendig.

Bei der Erstellung eines graphischen Editors mit GMF werden die folgenden Modelle verwendet:

- Das **Domain Model** ist ein EMF-Metamodell, das die Sprache beschreibt, für die ein Editor erstellt werden soll.
- Das **Graphical Def Model** beschreibt die graphische Darstellung von Elementen aus dem Domain Model. Graphische Darstellungen können z.B. Kästen und Linien, aber auch selbstdefinierte Icons sein.
- Im **Tooling Def Model** werden Werkzeuge für die Erstellung und Bearbeitung von Elementen des Domain Models im Editor beschrieben. Hier wird z. B. fest-

gelegt welche Werkzeuge es für die Erstellung eines neuen Elements gibt und in welchen Menüs sie sich befinden.

- Das **Mapping Model** kombiniert die Informationen der zuvor beschriebenen Modelle, legt also fest welches Element aus dem Domain Model welche graphischen Darstellungen und Werkzeuge aus den anderen Modellen zugewiesen bekommt, und dient somit als Ausgangspunkt der weiteren Generierung.
- Aus dem Mapping Model wird durch eine Transformation das **Diagram Editor Gen Model** erstellt, das alle Informationen für die Erstellung des graphischen Editors enthält. Aus diesem Modell wird mit Hilfe von Modell-zu-Text-Transformationen der Quellcode des Editors erstellt.

Weitere Informationen finden sich unter: <http://www.eclipse.org/modeling/gmf/> (Letzter Zugriff: 15.12.2008). Die im Projekt gemachten Erfahrungen mit GMF werden in Abschnitt 3.5 beschrieben.

## 2.4. Transformationswerkzeuge

Seit der Veröffentlichung eines Vorschlags für die Model Driven Architecture (MDA) [40] durch die Object Management Group (OMG) Ende 2001 sind bis heute eine Vielzahl von Werkzeugen entstanden, die alle beteiligten Personen im Prozess einer modellgetriebenen Entwicklung unterschiedlich gut unterstützen. Unter diesen Werkzeugen befinden sich sowohl kommerzielle Produkte als auch Open Source Projekte mit zum Teil stark differierendem Leistungsumfang.

Zu Beginn des Projekts wurden verschiedene Werkzeuge für die prototypische Umsetzung des in Kapitel 5 beschriebenen Vorgehens evaluiert. Die Kriterien dieser Evaluierung sowie die Auswahl der drei hier betrachteten Werkzeuge beziehen sich auf die Anforderungen des spezifischen Kontextes der BTC AG in Bezug auf die Prozessintegration. Aus diesem Grund wurden auch die in Abschnitt 2.5 beschriebenen Werkzeuge nicht in die Vorauswahl aufgenommen.

### 2.4.1. Vorauswahl von MDA-Werkzeugen

Im Vorfeld wurden drei MDA-Werkzeuge ausgewählt, die ausführlicher untersucht werden sollten.

1. AndromDA<sup>1</sup>
2. openArchitectureWare<sup>2</sup>
3. Software Factories<sup>3</sup>

---

<sup>1</sup><http://www.andromda.org>, letzter Zugriff: 15.12.2008

<sup>2</sup><http://www.openarchitectureware.org>, letzter Zugriff: 15.12.2008

<sup>3</sup><http://msdn.microsoft.com/vstudio/teamsystem/workshop/sf/default.aspx>, letzter Zugriff: 15.12.2008

Diese Werkzeuge werden in den folgenden drei Abschnitten näher vorgestellt und die Ergebnisse der Evaluierung beschrieben.

### 2.4.2. AndroMDA

AndroMDA ist ein in Java geschriebener Codegenerator, der in einem Open-Source-Projekt unter der BSD-Lizenz veröffentlicht und über Sourceforge.net verwaltet wird. Eine kostenlose Dokumentation zu AndroMDA ist auf der Projektseite zu finden. Kostenpflichtiger Support und Schulungen werden von dem Gründer Matthias Bohlen auf einer weiteren Webseite angeboten. Getestet wurde im Rahmen dieser Evaluation die Version 3.2 SNAPSHOT vom 12.09.2006.

Bei AndroMDA handelt es sich um einen Codegenerator und nicht um ein komplettes MDA-Werkzeug. AndroMDA ist ein Glied in einer Reihe von mehreren Werkzeugen.

AndroMDA erwartet als Input ein Modell im XMI-Format. Jedoch unterstützt AndroMDA derzeit ausschließlich XMI in den Versionen „1.1“, „1.2“ und „2.0 EMF“. Der von der Object Management Group (OMG) verabschiedete Standard in der Version 2.1 wird noch nicht unterstützt. Im typischen Workflow handelt es sich dabei um ein UML-Modell, das mit bestimmten Stereotypen, die in einem speziellen UML-Profil definiert worden sind, angereichert ist. In diesem UML-Profil sind des Weiteren auch Stereotypen für Attribute enthalten, um so z. B. ein Attribute *unique* zu setzen. Ebenfalls kann AndroMDA auch mit Tagged Values umgehen. Eine Einschränkung des Wertebereichs und die Definition eines Primärschlüssels sind nicht möglich, da das Datenbankmodell automatisch mittels Hibernate erzeugt wird. Prinzipiell könnte man auch Modelle zur Codegenerierung verwenden, die einem eigenen Metamodell entsprechen, das in MOF XMI vorliegt.

Für die Erzeugung von UML-Modellen könnte im Prinzip jedes UML-Werkzeug eingesetzt werden, das XMI erzeugt. In der Praxis konnte das nicht bestätigt werden, da die Hersteller der UML-Werkzeuge den XMI-Standard nicht einwandfrei umsetzen. Das AndroMDA-Projekt empfiehlt als UML-Werkzeug die Community Edition von MagicDraw, die allerdings nur für nicht kommerzielle Zwecke kostenlos zur Verfügung steht. Im Test erwies sich die Zusammenarbeit zwischen AndroMDA und MagicDraw als weitestgehend problemlos. Dies gilt allerdings nur für die Version 9.5 von MagicDraw die sich noch dem UML-Standard 1.4 widmet. Die derzeit aktuelle Version hat sich hingegen vollständig dem UML 2 Metamodell verschrieben. AndroMDAs vollständige Unterstützung von UML 2 ist aktuell noch in der Entwicklung. Eine weitere positive Zusammenarbeit herrscht zwischen AndroMDA und Poseidon kleiner Version 4.0. Mit der im Test verwendeten Poseidonversion 4.2.1 in der Community Edition war eine Zusammenarbeit nicht möglich. Als dritter Kandidat aus der Menge von UML-Werkzeugen wurde noch StarUML als Vertreter der Open-Source-Gemeinde herangezogen. Auch hier stellte AndroMDA das als XMI exportierte Modell vor Probleme. StarUML war das einzige Werkzeug, das direkt über die Möglichkeit verfügt mdl-Dateien einzulesen und das Modell auch graphisch darzustellen. Wird dieses Modell von StarUML als XMI exportiert ist der Informationsverlust jedoch so hoch, dass eine Weiterverarbeitung nicht sinnvoll möglich ist. Im Hinblick auf Mehrbenutzertauglich-



keit bietet StarUML die Möglichkeit Modellfragmente abzuspeichern, um es mehreren Personen zu ermöglichen an verschiedenen Teilen des Modells gleichzeitig arbeiten zu können. Für den Projekteinsatz sind dann klare Regeln erforderlich um festzulegen wer wann welches Teilmodell bearbeiten darf. Oder aber es muss ein Versionierungswerkzeug zum Einsatz kommen, das ausgecheckte Dateien für einen weiteren Zugriff sperrt. Eine Übersicht über verschiedene UML-Werkzeuge und ihre Zusammenarbeit mit AndroMDA befindet sich ebenfalls auf der Projekt-Homepage.

Das Einlesen von XMI Light, einer auf das wesentliche reduzierten Form von XMI, das im Rahmen des UMT-QVT entwickelt wurde, wird von AndroMDA ebenfalls nicht unterstützt. Eventuell besteht die Möglichkeit für XMI Light ein MOF konformes Metamodell in XMI zu erstellen und so AndroMDA XMI Light „beizubringen“.

Die Modell-zu-Modell-Transformation ist in AndroMDA derzeit noch nicht möglich, ist aber für das nächste Major Release 4.0 geplant.

Für die Modell-zu-Text-Transformationen sind so genannte Cartridges verantwortlich. Für AndroMDA gibt es derzeit die folgenden ready-to-use Cartridges: (N)Spring, EJB 2 /3, Web Services, (N)Hibernate, Struts, JSF, Java, XSD und C#.

Im Praxiseinsatz zeigte sich, dass die Konfiguration aller benötigten Komponenten von AndroMDA sehr aufwändig ist. Einmal konfiguriert und mit dem Einsatz des sehr gut unterstützten UML-Werkzeugs MagicDraw ist das Nachvollziehen der Beispiele schnell möglich.

### 2.4.3. openArchitectureWare

In dieser Evaluation wurde die Version 4.1 von openArchitectureWare (oAW) untersucht. Das Entwicklungswerkzeug oAW ist eine Sammlung aufeinander abgestimmter Werkzeuge und Komponenten zur Unterstützung modellgetriebener Softwareentwicklung. Es baut auf einem modularen Generator-Framework auf, das weitgehend mit der MDA kompatibel ist und in Java programmiert wurde. Es genießt mittlerweile einen guten Ruf dank vieler Sponsoren, einer ausgebauten Online-Community, ausführlicher Dokumentation und einer guten Einbindung in die graphische Entwicklungsoberfläche Eclipse. Ab der Version 4.0 ist oAW Subprojekt innerhalb des Generative Modeling Tools (GMT) Projekts von Eclipse und steht deshalb unter der Eclipse Public License. Im Gegensatz zu AndroMDA, dessen Stärke eine „out-of-the-box“-Funktionalität durch eine Vielzahl bereits vorhandener Transformationen ist, setzt oAW vor allen Dingen auf die effizientere Entwicklung eigener Modell-Transformationen, denn oAW bietet Programmierern die volle Unterstützung einer modernen Entwicklungsoberfläche und sie müssen nicht ihre Metamodelle an die Besonderheiten vorgegebener Transformationen anpassen. Für die Definition eines Metamodells ist ein Metametamodell erforderlich. In oAW kann als Metametamodell das Eclipse-eigene Eclipse Modeling Framework (EMF) verwendet werden. EMF setzt auf Essential MOF (EMOF) auf, einem weitestgehend MOF-kompatiblen Standard von IBM. Alternativ kann ebenfalls das von früheren Versionen bekannte Metametamodell oAW Classic verwendet werden(vgl. [1]).

Abbildung 2.6 zeigt den grundlegenden Aufbau bzw. die verschiedenen Einsatzszenarien von oAW und die dabei eingesetzten Standards, Sprachen und Frameworks.

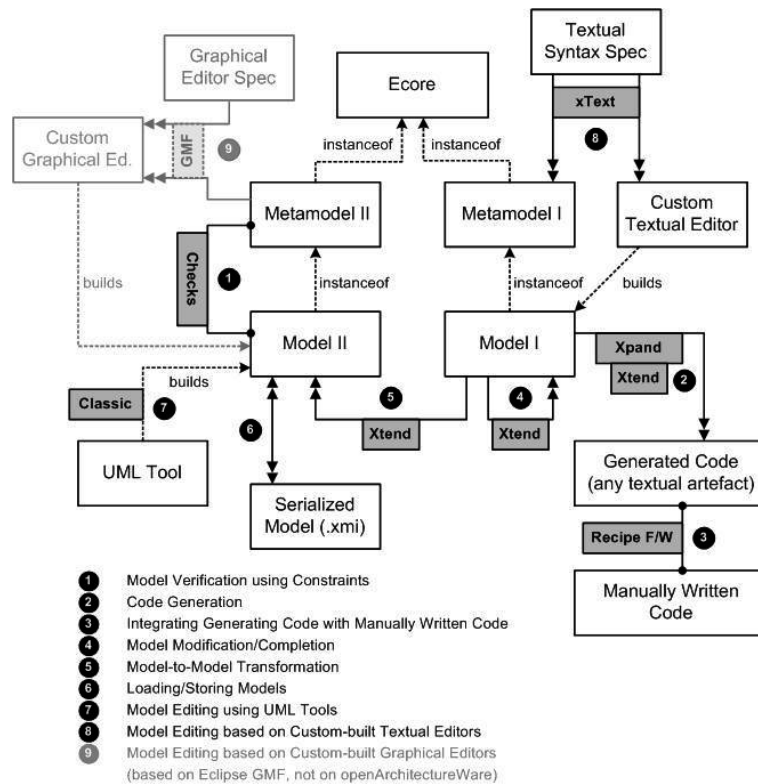


Abbildung 2.6.: Überblick über oAW

Beschränkt man sich allein auf die Codegenerierung, so wie es momentan noch bei AndroMDA der Fall ist, ist der Weg mit dem Startpunkt 7 interessant. oAW bietet über so genannte „Adapter“ die Möglichkeit verschiedene Modelle externer Quellen einzulesen und nutzbar zu machen. Allerdings wird in dem verwendeten Metamodell nur der, in den Augen der Entwickler, wichtigste Teil des UML-Standards abgebildet. Folgende Adapter stehen aktuell dafür zur Verfügung: Eclipse, EMF, UML2, UML-Werkzeuge, MagicDraw, Poseidon, Enterprise Architect, Rose, XDE, Innovator, StarUML, sowie andere Quellen wie XML und Visio.

Größte Herausforderung zu Beginn eines Projekts ist die Erstellung eines geeigneten Templates zur Codegenerierung. Die in oAW verwendete Sprache für Model-to-Code Transformationen ist XPand (siehe Punkt 2 in Abbildung 2.6). Dank der guten Integration in die EclipseIDE verfügt der Editor auch für XPand über Syntax-Highlight und Autovervollständigung welche den Umgang mit den Templates erleichtern.

Templates bestehen aus den Platzhaltern der XPand-Sprache und statischem Text. Zur Unterscheidung von Platzhaltern und Text werden die Platzhalter zwischen einem öffnenden und schließenden Guillemot geschrieben: «Xpand-Ausdruck». Platzhalter werden beim Ausführen des Templates vom Codegenerator umgesetzt. Der außerhalb

stehende Text wird direkt in die Zielfeile ausgegeben. Die Aufgabe von XPand ist es, ein Instrumentarium zur Verfügung zu stellen, mit dem man auf die Elemente des Modells zugreifen und ihre Eigenschaften abfragen kann. Templates werden in einer Datei mit der Endung .xpt gespeichert.

Die Erstellung von eigenen Templates erfordert viel Zeit, bevor der erste Code erstellt wird. Aber es wurde bewusst auf die Integration vorgefertigter Templates verzichtet, da in den Augen der leitenden Entwickler des Projekts in der Praxis vorgefertigte Templates nie das gewünschte Ergebnis liefern. Ein Projekt, das dennoch „ready-to-use“ Templates anbieten möchte, nennt sich Fornax und wurde Mitte Mai 2006 gegründet. Zur Zeit der Untersuchung stand aber noch kein Template zum Download zur Verfügung.

Wie bei allen Codegeneratoren entstehen beim reinen Forward-Engineering keine Probleme zwischen handgeschriebenem und generiertem Code. Um Konflikte beim Round-Trip-Engineering zu vermeiden besitzt oAW zwei Möglichkeiten. Innerhalb der Templates können geschützte Bereiche definiert werden, die bei einer erneuten Generierung nicht überschrieben werden. Diese Methode stammt aus der Vergangenheit von oAW. Der neue, zusätzliche Ansatz besteht in dem Recipe Framework (vgl. Punkt 3, Abbildung 2.6). Dieses sorgt für eine bessere Integration von generiertem und manuellem Code, in dem es nach einer erneuten Generierung definierte Checks durchführt. Ein weiterer wichtiger Bereich bei modernen Softwareprojekten ist das Testen. Das mittlerweile etablierte Unit-Testing wird von oAW unterstützt indem die entsprechenden Testklassen gleich miterzeugt werden können.

Der Ablauf einer Modell-zu-Text-Transformation wird in oAW über einen Workflow in einer Workflow-Engine konfiguriert. In der XML-basierten Workflow-Konfiguration wird unter anderem angegeben:

- welches Metamodell geladen wird,
- welches Modell geladen wird,
- welches Template verwendet wird,
- an welchen von oAW mitgelieferten Generator das Template übergeben wird,
- das Encoding des generierten Codes und den
- Pfad für das Schreiben des generierten Code ins Dateisystem

In der Workflow-Datei werden alle benötigten Modelle und Metamodelle geparkt und geladen. Über die von oAW mitgelieferte Metamodell-Implementierung kann auf die Elemente der Modelle zugegriffen und deren Eigenschaften ausgelesen werden. Im Anschluss wird ein instanziiertes Model des Metamodells an ein in XPand implementiertes Template übergeben. Dieses generiert dann aus dem übergebenen Model den Code in der Zielsprache und schreibt das Ergebnis ins Dateisystem.

oAW besitzt aber nicht nur die Möglichkeit ein PIM oder PSM in Code umzuwandeln, sondern bietet auch ein Framework, um zunächst eine DSL auf der CIM-Ebene

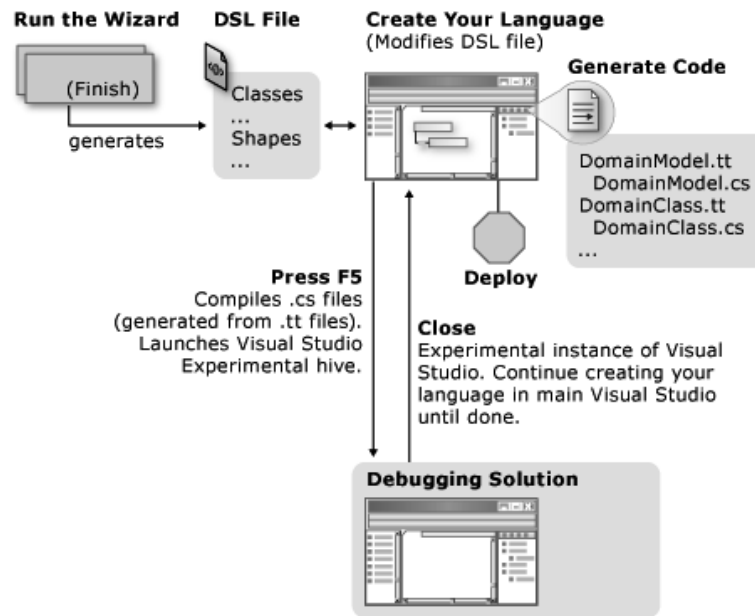


Abbildung 2.7.: Workflow DSL Tools [39]

zu entwerfen, für diese Sprache einen graphischen oder textuellen Editor zu erstellen und die damit erstellten Modelle weiter zu transformieren, entweder in andere Modelle oder in Code.

Bei oAW handelt es sich um ein sehr flexibles Framework, das sich dank der Integration in das Eclipse-Projekt bzw. in das Subprojekt GMT gut in die Eclipse IDE einfügt. Der Vorteil der hohen Flexibilität geht zu Lasten der Einarbeitungszeit und der Zeit, die benötigt wird, um fertigen Code generieren zu können. Für das MINT-Projekt bietet oAW aber ggf. genau die Flexibilität und Mächtigkeit, die benötigt wird.

#### 2.4.4. Software Factories

Der Begriff „Software Factories“ bezeichnet eine von Microsoft entwickelte Methodologie zur industriellen Softwareentwicklung, insbesondere im Kontext von Software-Produktlinien. In diesem Abschnitt sollen die dafür bereitgestellten DSL-Tools untersucht und näher vorgestellt werden. Für diese Untersuchung wurde das Release v1 der DSL-Tools vom 15.09.2006 aus dem VS2005 SDK v3 verwendet. Mit dieser Version, die gleichzeitig das erste finale Release ist, wurden noch einige Änderungen gegenüber den Versionen älter als März 2006 vorgenommen.

Die DSL-Tools wählen einen ähnlichen Ansatz wie oAW4. Der reguläre Workflow (vgl. Abbildung 2.7) besteht darin, eine Solution mit dem Projekt-Wizard anzulegen. Die DSL-Tools bieten dazu vier verschiedene Templates:

1. Class Diagram - An example demonstrating CompartmentShapes and shape inheritance.

2. Component Model - Languages with interconnected ports
3. Minimal Language - A complete small example
4. Task Flow - A basis for workflow, activity and state language

Muss eine komplett unabhängige DSL erzeugt werden, wird das Template „Minimal Language“ ausgewählt. Wurde eine Domäne abgegrenzt, kann für diese Domäne ein Metamodell modelliert werden. Im Kontext von den DSL-Tools wird das Metamodell Domänen-Modell genannt. Als Metametamodell wählt Microsoft nicht den OMG Standard MOF sondern verwendet ein eigenes, nicht näher spezifiziertes Modell das ähnlich EMF sein soll.

Beim Erstellen eines Domänen-Modells werden gleichzeitig auch die Informationen für einen graphischen Editor eingefügt. So können sowohl vorhandene Formen und Linien als auch eigene Graphiken, z.B. für Zustände eines Automaten, verwendet werden. Dieser Vorgang wird wiederum komfortabel mit einem graphischen Editor durchgeführt.

Zusätzlich zum reinen Editor können Constraints definiert werden. Wird ein Constraint verletzt, zeigt der DSL-Editor dies direkt während der Modellierung an. Ist die DSL funktionsfähig, kann sie deployed werden. Dies hat den Vorteil, dass sie auch in der Visual Studio Standard Edition verwendet werden kann, wohingegen die Erstellung nur mit der Professional Edition oder größer durchgeführt werden kann. Im Hinblick auf die formalen Kriterien sieht es so aus, dass eine Zusammenarbeit mit gängigen UML-Werkzeugen nicht möglich ist. Die einzige Unterstützung, die noch angeboten wird, besteht für Visio. Da die UML-Unterstützung aber auch in Visio nicht weiterentwickelt wird und bei UML 1.3 stehen geblieben ist, ist dies keine Alternative. In einem Blog eines Microsoft Mitarbeiters wurde auf die Frage nach XMI Unterstützung geantwortet:

XMI interchange is clearly on our roadmap for the DSL tools, although I can't say when. We support it today with Visio. Of course, there are many different versions of XMI, and we have to choose which to support.<sup>4</sup>

Die einzige weitere Möglichkeit der Interoperabilität besteht durch ein Bridging-Verfahren, das an der Universität von Nantes beschrieben wurde [8]. Dieses Verfahren versucht jede Modell-Ebene verschiedener Tools aufeinander abzubilden. Der Workflow der DSL-Tools lässt sich auf die CIM- und die Code-Ebene von MDA abbilden. Die PIM- und PSM-Ebene werden nicht behandelt.

Mit den DSL-Tools werden, wie auch bei oAW, keine vorgefertigten Templates ausgeliefert. Was also mit den DSL-Tools realisiert werden kann, liegt ganz am verwendeten Domänen-Modell und den selbst erstellten Templates.

Dabei unterstützen die DSL-Tools primär das Forward-Engineering. Generierter Code wird bei jeder erneuten Generierung überschrieben. Es gibt innerhalb des generier-

---

<sup>4</sup>Quelle: <http://blogs.msdn.com/stevecook/archive/2005/01/05/346838.aspx>, letzter Zugriff: 11.12.2008

ten Codes keine geschützten Bereiche. Anpassungen an Klassen müssen über Vererbungen realisiert werden. Wird jedoch der Class Designer von Visual Studio genutzt, sind Code und Klassendiagramm stets synchron. Jedoch ist der Class Designer auf Klassendiagramme beschränkt und unterstützt keine individuelle Codegenerierung. Für den Import von bestehenden UML-Klassendiagrammen in Visual Studio empfiehlt Microsoft durch das ursprüngliche UML-Werkzeug Code generieren zu lassen und diesen zu importieren. Daraufhin kann der Class Designer das entsprechende Klassendiagramm darstellen.

Die Unterstützung für Unit-Testing ist vorgesehen aber noch in keinem Beispiel durchgeführt worden. Die Dokumentation im MSDN ist gut. Im Weiteren sind aber noch nicht viele Informationen und Tutorials zu den DSL-Tools vorhanden, was u. a. an der erst kurzen Verfügbarkeit der finalen Version liegen kann.

Insgesamt war das Arbeiten mit den DSL-Tools angenehm, da sie sich nahtlos in Visual Studio integrieren. Der Nachteil an dieser Lösung ist jedoch die feste Anbindung an die .NET Welt und die Verwendung von proprietären Techniken statt offener Standards.

#### 2.4.5. Bewertung der untersuchten Werkzeuge

In der Tabelle 2.1 werden die drei Transformationswerkzeuge anhand ausgewählter Kriterien gegenübergestellt. Die getesteten Werkzeuge lassen sich nur schwer miteinander vergleichen. Auf das Wesentliche reduziert kann man folgende Ergebnisse festhalten:

- AndromDA ist ein gutes Werkzeug wenn aus UML-Modellen schnell Code generiert werden soll, sowohl für Java als auch für .NET. Hier liegt der Vorteil in den bereits vorhandenen Cartridges, die schnell Ergebnisse liefern. Die Arbeit mit UML-Modellen ist vielen Entwicklern geläufig. Allerdings werden in den meisten Fällen Änderungen an den Cartridges notwendig sein.
- openArchitectureWare ist das flexibelste Werkzeug im Testfeld und erfüllt alle drei Auswahlkriterien. Es bietet ein Framework um beliebige Schritte aus MDA durchführen zu können. Durch die Eingliederung in die Eclipse Community, wird oAW sicherlich weiterhin gut unterstützt und noch besser in die Eclipse IDE integriert werden. Der kostenlose Support im Internet war sehr gut. So wurde ein Fehler im StarUML-Mapping in Zusammenarbeit mit einem Entwickler innerhalb einer halben Woche behoben.
- Die Microsoft DSL-Tools sind eine benutzerfreundliche aber speziell auf die Microsoft .NET Welt zugeschnittene Lösung. Eine Zusammenarbeit mit den offenen Standards im MDA-Umfeld besteht aktuell noch nicht. Microsoft bestreitet hier einen eigenwilligen Weg (siehe dazu VS2005TS Modeling Strategy<sup>5</sup>), so wird direkt aus der DSL Code generiert und so die PIM- und PSM-Ebene übersprungen.

---

<sup>5</sup>[http://msdn.microsoft.com/en-us/library/ms379623\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/ms379623(VS.80).aspx), letzter Zugriff: 11.12.2008

Kriterien	AndroMda	oAW	Software Factories
MOF	Ja	Ja	Nein
UML	1.4, (2.0 in Arbeit)	Eclipse UML2, EMF, UML nicht vollständig .	nur 1.3 von Visio
XMI	1.1, 1.2, (2.0 EMF)	siehe folgende Zelle	nein, aber geplant
XMI Tools	MagicDraw++, Poseidon <4.0+, StarUML-	MagicDraw++, Poseidon, Enterprise Architect, Rose, XDE, Innovator, StarUML, XML, Visio	Nein
XMI	Light	Nein	Nein
MDA Ebenen	PIM,(PSM), Code	CIM, PIM, PSM, Code	CIM, Code
Wertebereiche	Nein	Def im Metam.	Def im Metam.
Primärschlüssel	Nein	Def im Metam.	Def im Metam.
Unique Attribute	Ja	Def im Metam.	Def im Metam.
Tagged Values	Ja	Def im Metam.	Def im Metam.
Model2Model	nein, erst ab 4.0	ja, xTend, ATL	nein
Graphische DSL	Nein	Ja	Nein
Model2Code	ja, Velocity	ja, XPand	ja, ASP.NET ähnlich
Zielsprachen	(N)Spring, EJB2/3, WS, (N)Hibernate, Struts, JFS, Java, XSD	keine fertigen Templates	keine fertigen Templates
Templates anpassbar	Ja	Ja	Ja
Eigene Templates	Ja	Ja	Ja
Template Sprache	Velocity	XPand	ASP.NET ähnlich
Mehrfachvererbung	Nein	Nein	Nein
Forward Engineering	Ja++	Ja++	Ja++
Round-Trip	Möglich	Möglich	Nein
Reverse Engineering	nur Schema2XMI	Nein	Nein
Unit Testing	Ja	Ja	Ja
Mehrbenutzerf.	Nein	Nein	Nein
Versionierung	Nein	Nein	Nein
Preis	Kostenlos	Kostenlos	800 €
Dokumentation	Vorhanden	Vorhanden	Vorhanden

Tabelle 2.1.: Bewertung der MDA-Werkzeuge

Die erste finale Version der DSL-Tools ist noch sehr jung, weshalb sich in Bezug auf Unterstützung, Tutorials und Templates bestimmt noch einiges in den nächsten Monaten entwickeln wird.

oAW deckt den kompletten MDA-Ansatz ab, ermöglicht als einziges Tool Modell-zu-Modell-Transformationen, bietet die größte Flexibilität, orientiert sich an offenen Standards und ist somit die erste Wahl für die Umsetzung der Konzepte zur Prozessintegration im MINT-Projekt.

## 2.5. Werkzeuge für die modellgetriebene Datenintegration

Von der Delta Software Technology GmbH (DSTG) wurde im Rahmen des MINT-Projekts das Produkt SCORE<sup>®</sup> Adaptive Bridges – Data Architecture Integration<sup>™</sup> eingesetzt und weiterentwickelt. Ein Schwerpunkt dieser Weiterentwicklungen betraf die dort eingesetzte Basistechnik HyperSenses<sup>™</sup>.

### 2.5.1. SCORE Adaptive Bridges – Data Architecture Integration

Der Schwerpunkt der Aktivitäten von DSTG in MINT lag auf der Untersuchung der Anbindung relationaler Legacy-Datenbanken. Auch unabhängig von MINT sieht DSTG die Datenintegration als eine zentrale Integrationsaufgabe an, für die ein eigenes Produkt existiert: SCORE Adaptive Bridges – Data Architecture Integration (kurz: SCORE). Dessen Schwerpunkt lag bis zum Start des MINT-Projekts auf nicht-objektorientierten Umgebungen. Waren in Kundenprojekten objektorientierte Clients erforderlich, wurden sie durch eine zusätzliche generierte Schicht („Proxies“) angebunden. Im Rahmen von MINT fanden zahlreiche Erweiterungen statt, um eine umfassendere OO-Unterstützung in SCORE zu erhalten.

SCORE basiert auf der in Abb. 2.8 skizzierten Schichtenarchitektur [13]. In Datenobjekten sind Datenzugriffe auf Basis des DB-Schemas definiert. Sie bilden die Datenzugriffsschicht, die unmittelbar auf dem DBMS bzw. der eingesetzten Middleware aufsetzt. Das „Service Interface“ beschreibt die Schnittstelle der gesamten Datenhaltungskomponente gegenüber der Client-Applikation. Seine Struktur orientiert sich an den Client-seitigen Zugriffen. Sie kann inhaltlich völlig anders strukturiert sein als die Datenzugriffsschicht. Die Operationen des Service Interface besitzen Implementierungen, die Methoden der Datenobjekte verwenden. Diese Abbildung wird im Folgenden als „Mapping“ bezeichnet. Im Detail findet beispielsweise ein Mapping von Operationsparametern auf Methodenparameter statt und vice versa. Eine feste Eins-zu-eins-Zuordnung gibt es dabei nicht, einzelne Mappings können z. B. auch nur Anwendungscoding umfassen. Im Folgenden fassen wir begrifflich das Service Interface und die Implementierungs- und Mapping-Schicht als „Service-Schicht“ zusammen.

Die Datenzugriffs- und die Service-Schicht bilden gemeinsam den Anwendungsadapter. Die zentrale Idee von SCORE besteht darin, diesen Anwendungsadapter aus neutralen Definitionen zu generieren. Der SCORE-Workflow lässt sich in folgende Schritte unterteilen:



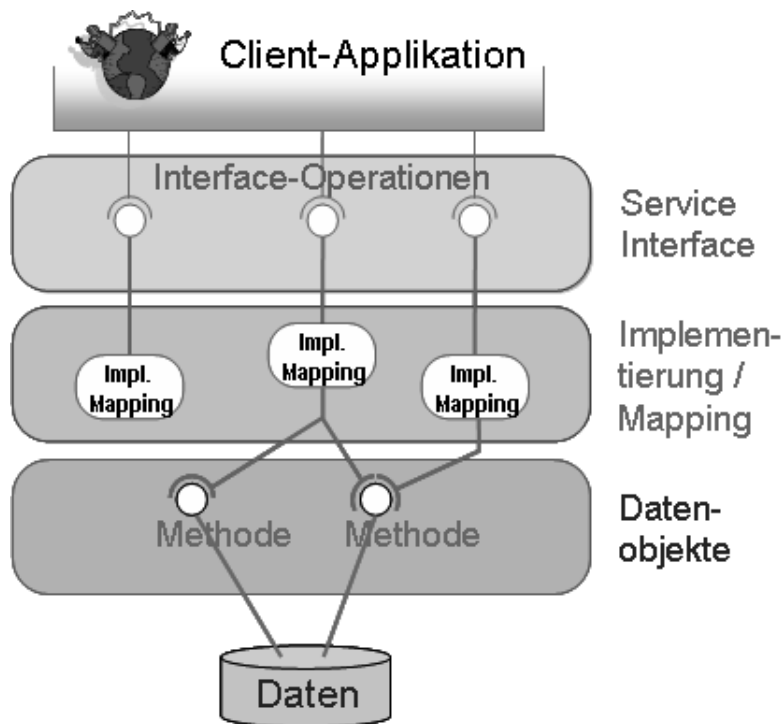


Abbildung 2.8.: Schichtenarchitektur von SCORE

- Schema-Import
- Adapter-Definition
- Adapter-Generierung

Diese Arbeitsschritte werden in den folgenden Abschnitten genauer erläutert.

### Schema-Import

Der Ausgangspunkt des Schema-Imports ist ein vorhandenes DB-Schema (s. Abb. 2.9). Ein DB-Schema ist – technisch wie inhaltlich – DBMS-spezifisch. In einem ersten Schritt überführt ein DataDef-Generator das DB-Schema in eine neutrale, plattform-unabhängige Form, das „Data Definition File“. Dieses Dateiformat ist ein menschlich lesbares Textformat, welches manuelle Ergänzungen in Form von Include-Dateien ermöglicht. Solche Ergänzungen sind dann notwendig, wenn das DB-Schema unvollständig ist – ein Fall, der in der Praxis häufig anzutreffen ist. Das ggf. vervollständigte Data Definition File wird von einem Templates-Generator in ein „Templates Repository“ überführt, d. h. aus dem lesbaren Textformat wird ein technisches<sup>6</sup>. Das Templates Repository ist ein PIM im Sinne der MDA.

<sup>6</sup>In diesem Fall handelt es sich um die DSTG-eigene Implementierung von MOF.

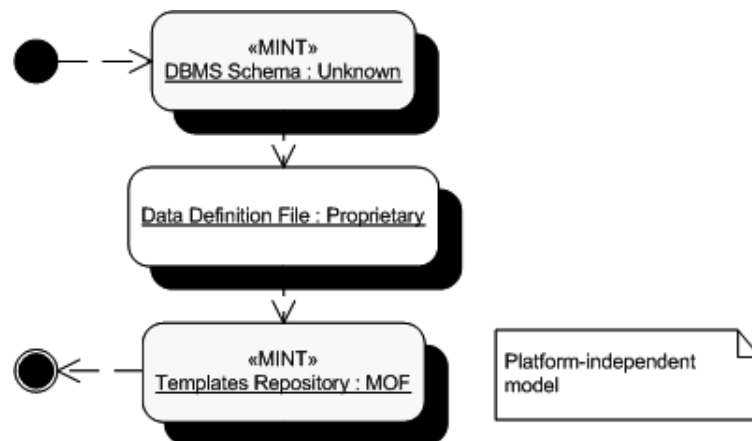


Abbildung 2.9.: Datenfluss des Schema-Imports

### Adapter-Definition

Die Definition der Datenobjekte, des Service Interface und der Mappings findet in dem Werkzeug SCORE<sup>®</sup> Composition Manager<sup>™</sup> statt. Es ist die grafisch-interaktive Benutzerschnittstelle von SCORE. Für die Definition der Datenobjekte im SCORE Composition Manager werden selektiv Templates aus dem Templates Repository importiert. Diese werden an die Erfordernisse der konkreten Anwendung angepasst. Das Service Interface kann in sehr einfachen Fällen aus den Datenobjekten abgeleitet werden. Ist es allerdings völlig anders strukturiert, so ist es manuell zu erfassen. Die abschließende Definition der Mappings erfolgt ebenfalls manuell<sup>7</sup>. Alle Definitionen für die Service- und die Datenzugriffsschicht werden schließlich als Composition Repository gespeichert (s. Abb. 2.10).

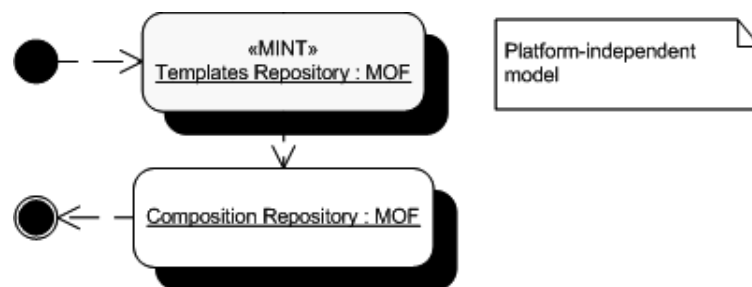


Abbildung 2.10.: Datenfluss der Adapter-Definition

Insgesamt stellen die Definitionen im SCORE Composition Manager einen hochgradig iterativen Prozess dar: Es ist möglich, die Datenobjekte zuerst zu definieren, wenn

<sup>7</sup>Beide „manuelle“ Arbeitsschritte werden durch SCORE Composition Manager unterstützt. Da es sich um kreative Entwurfsentscheidungen handelt, ist hier das Potential für eine weitergehende Automatisierung gering.

der verwendete DB-Ausschnitt bereits bekannt ist. Dann kann bei der Definition des Service Interface bereits das Mapping berücksichtigt und eingebaut werden. Umgekehrt ist es möglich, dass das Service Interface fest definiert ist, und dann – abhängig vom Mapping – der passende Ausschnitt des DB-Schemas in Form von Templates importiert wird. Es gibt hier keine durch das Werkzeug vorgegebene Reihenfolge. So ist im SCORE Composition Manager eine automatische Konsistenzprüfung ein separater, jederzeit durch den Benutzer zu aktivierender, Arbeitsschritt.

### Adapter-Generierung

Die Adapter-Generierung erhält das Composition Repository als Input und erzeugt daraus den Zielcode, der die zuvor definierten Schichten für Datenobjekte, Service Interface und Mappings implementiert. Ein temporäres Zwischenprodukt ist dabei das „Extract Model“ (s. Abb. 2.11).

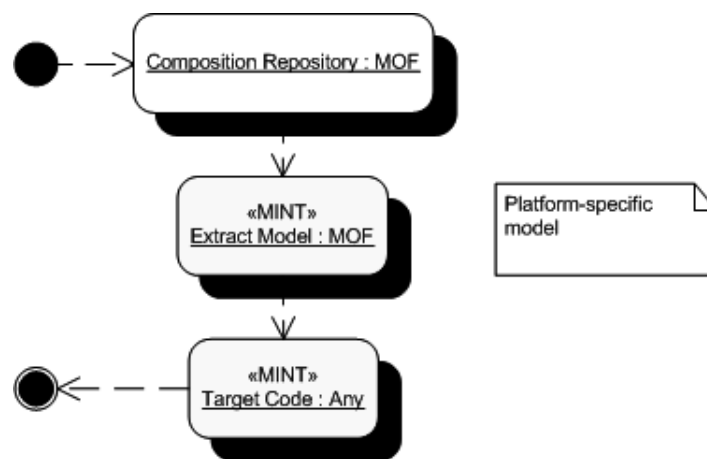


Abbildung 2.11.: Datenfluss der Adapter-Generierung

Das Extract Model, erzeugt von einem Extract Model-Generator, liefert den Input für die Generierung von Zielcode („Target Code“) durch einen Application Adapter-Generator. Es enthält die für die jeweilige Generierung benötigten Daten in Form eines HyperSenses-Modells (s. 2.5.2) und ermöglicht so die Implementierung des Application Adapter-Generators mit HyperSenses. Das Extract Model ist ein PSM im Sinne der MDA (s. 2.1). Insgesamt ist SCORE eine hochgradig automatisierte Werkzeugkette, die an zwei Stellen, nämlich vor der Erzeugung des PIM und des PSM, manuelle Ergänzungen erlaubt (im Data Definition File bzw. im Composition Repository).

Während die generierte Service-Schicht von ihrer Struktur her sehr plattformspezifisch ist, gibt es für die generierte Datenzugriffsschicht ein allgemeines Datenmodell, das SCORE-Datenmodell (s. Abb. 2.12) [13].

- Mit dem Navigator-Objekt<sup>8</sup> legt man die Fundstellenmenge der Daten fest, auf

<sup>8</sup>In seiner allgemeinen Form ist das Datenmodell objektbasiert, d. h. es gilt hier Klasse gleich Objekt.

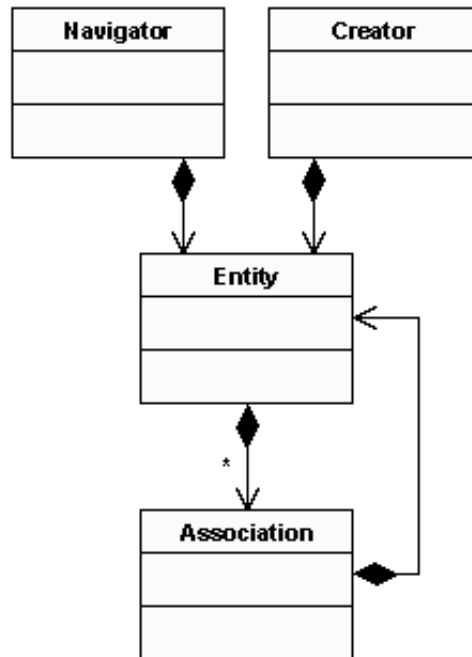


Abbildung 2.12.: Das SCORE-Datenmodell

die man zugreifen möchte. Es enthält außerdem die Methoden, um auf dieser Fundstellenmenge zu navigieren. Der Zugriff auf die Daten erfolgt jeweils über ein Entity-Objekt. Der Aufruf einer Navigationsmethode ändert jeweils das zugehörige Entity-Objekt.

- Pro Navigator-, Creator- oder Association-Objekt gibt es jeweils ein Entity-Objekt. Das Entity-Objekt repräsentiert einen Datensatz im Datenhaltungssystem. Über Methoden des Entity-Objekts kann man die Daten aus der Datenbank lesen und verändern.
- Wenn man Beziehungen („Relationships“) zwischen Entitäten definiert hat, so benötigt man eine Möglichkeit, um von einer Entität zur anderen navigieren zu können. Relationships entsprechen in SCORE Assoziationen. Zu einem Entity-Objekt kann es mehrere Association-Objekte geben, die jeweils ein Entity-Objekt enthalten.
- Mit einem Creator-Objekt kann man nur neue Daten erzeugen. Eine Navigation ist damit nicht möglich. Das zugehörige Entity-Objekt enthält nur Methoden zum Einfügen neuer Datensätze.

Für MINT wurde die Semantik dieses Konzepts beibehalten. Die konkrete Ausgestaltung in Form von Klassen wurde jedoch an OO-Erfordernisse angepasst, s. Kapitel 6.6.1 (Abschnitt „Zugriffsklassen“). Für die generierte Service-Schicht gibt es zwar

kein allgemeines Konzept, wohl aber ein (im Rahmen von MINT definiertes) Konzept für die Abbildung der Schnittstelle objektorientierter Clients auf Zugriffe der Datenzugriffsschicht, vgl. dazu Kapitel 6.6.1 (Abschnitt „Adapterklassen“).

### 2.5.2. HyperSenses

HyperSenses ist ein System zur Entwicklung und Anwendung von Software-Generatoren. Es bietet für beide Aufgaben eine interaktive Werkzeugunterstützung. Diese ermöglicht es bei der Generatorentwicklung, eine manuelle Programmierung zu vermeiden, s. dazu Abschnitt 2.5.2. Bei der Anwendung eines mit HyperSenses erstellten Generators gibt es eine Unterstützung für domänenspezifische Konfigurationssichten bzw. DSLs. HyperSenses verfolgt dabei einen strikt modellbasierten Ansatz und basiert auf offenen Standards (XML, MOF).

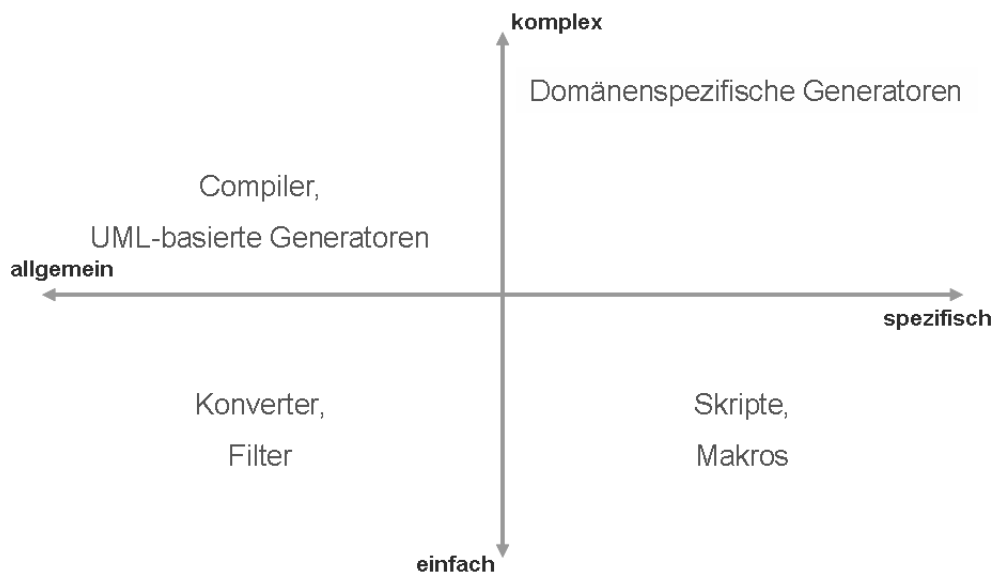


Abbildung 2.13.: Abstraktion und Scope von Generatoren

Software-Generatoren lassen sich in Bezug auf ihren Einsatzbereich (Scope) und die Komplexität der durch sie implementierten Transformation unterscheiden (siehe Abb. 2.13): Im weitesten Sinne sind auch einfache Filter und Konverter Generatoren, sie führen einfachste Transformationen durch und haben ein breites Einsatzspektrum. Ähnlich verhält es sich mit Skripten und Makros, nur sind diese in der Regel auf ein bestimmtes, eng gefasstes Einsatzszenario abgestimmt. Compiler und UML-basierte Generatoren, etwa solche die aus UML-Klassendiagrammen Java-Klassen erzeugen, führen anspruchsvollere Transformationen durch, haben aber ein nahezu universelles Einsatzgebiet. Im Gegensatz dazu bieten domänenspezifische Generatoren ein Höchstmaß an Abstraktion und eine größtmögliche Anpassung an das jeweilige Einsatzgebiet. Beide Kriterien bedingen einander. HyperSenses ist ein System zur Erstellung und Anwendung solcher domänenspezifischer Generatoren [23].

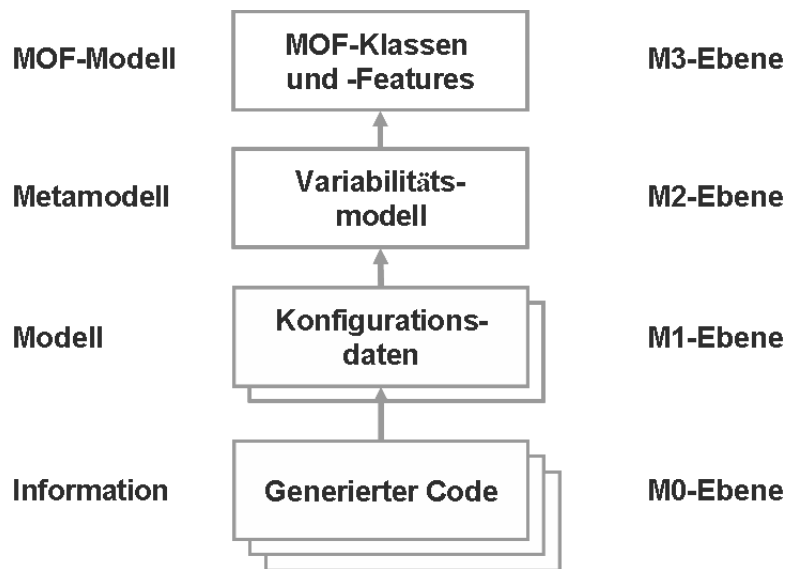


Abbildung 2.14.: Codegenerierung und MOF

HyperSenses basiert konzeptionell auf dem OMG-Standard MOF. Wendet man die MOF-Modellebenen auf den Fall der Codegenerierung an, so ergibt sich die in Abb. 2.14 gezeigte Struktur. Hier sind besonders die in HyperSenses immer wiederkehrenden Begriffe Modell und Metamodell wichtig, die den Konfigurationsdaten bzw. dem Variabilitätsmodell entsprechen.

Die modellbasierte Philosophie von HyperSenses markiert zugleich eine Top-Down-Methodik: Bevor ein Modell erstellt werden kann, muss das passende Metamodell definiert worden sein. Während letzteres einen Bestandteil des Generators darstellt, wird das Modell erst bei der Anwendung des Generators erstellt bzw. verarbeitet. HyperSenses kombiniert Top-down- mit Bottom-up-Methoden. Für letzteres steht insbesondere die Pattern By Example<sup>TM</sup>-Methode.

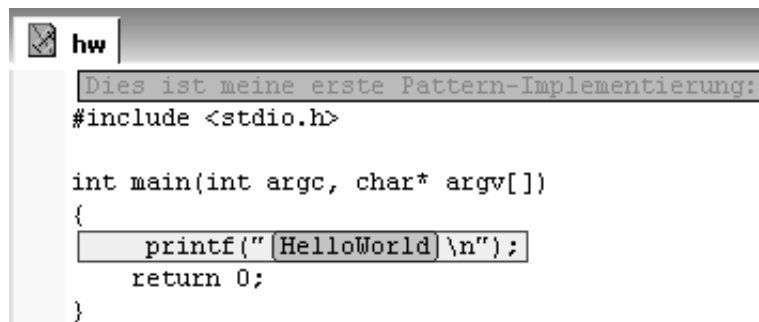
### Pattern By Example

Eine gängige Programmier-Technik ist das sogenannte „Copy-Paste-Adapt“: Man möchte etwas programmieren und entsinnt sich, so etwas ähnliches bereits an anderer Stelle realisiert zu haben. Der entsprechende Quellcode wird kopiert und dann im neuen Kontext angepasst. Es ist die einfachste und häufigste Form der Wiederverwendung. Leider ist sie auch unsystematisch und fehlerträchtig. Die Idee der Pattern By Example (PBE)-Methode besteht darin, für Copy-Paste-Adapt eine Werkzeugunterstützung anzubieten, um diese Nachteile zu beseitigen [11].

Der Ausgangspunkt für PBE ist ein existierendes oder prototypisches Software-Artefakt. Dieses wird in den HyperSenses Pattern-Editor importiert und dort neutralisiert: Variationspunkte werden identifiziert und mit Bedingungen und Berechnungen versehen. Diese Definitionen werden in einer interaktiven grafischen Oberfläche vorge-

nommen. Auf diese Weise entsteht aus einem spezifischen Codefragment ein wiederverwendbares, parametrisierbares Codefragment – ein sogenanntes „Code Pattern“<sup>9</sup>. Bzgl. der Zielsprache oder des Inhalts gibt es für Code Patterns keinerlei Beschränkungen.

Code Patterns bestehen aus einem Pattern-Interface und mindestens einer Pattern-Implementierung. Das Interface enthält die Parameter des Code Patterns. Eine Pattern-Implementierung ist ein zusammenhängendes Fragment Zielcode, welches das Interface verwendet. Mit der PBE-Methode wird daher nicht immer ein neues Code Pattern angelegt, sondern eine Pattern-Implementierung. Betrachten wir als Beispiel einen Generator, der die C-Version von „Hello, World!“ erzeugen soll. Die entsprechende Pattern-Implementierung zeigt Abb. 2.15.



```
hw
Dies ist meine erste Pattern-Implementierung:
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("HelloWorld\n");
    return 0;
}
```

Abbildung 2.15.: HelloWorld-Implementierung für C

Der in der *printf*-Anweisung ausgegebene String soll variabel sein. Er ist daher als Slot markiert worden. Slots bezeichnen Variationspunkte im Zielcode und können sehr verschiedenartig definiert werden. In diesem Fall geht es um einen Slot für einen konkreten Wert, einen String. Solche Slots werden auch als Expression-Slots bezeichnet, da der jeweilige Inhalt durch eine Expression aus Generierungsparametern berechnet wird. Im einfachsten Fall bedeutet dies eine Eins-zu-eins-Übernahme des Parameters. Der Slot wird umschlossen von einem optionalen Codeblock, welcher hier genau eine Zeile umfasst. Die Expansion eines optionalen Codeblocks ist davon abhängig, ob die enthaltenen Slots mit Werten gefüllt sind. Optionale Codeblöcke können mit Zusatzbedingungen versehen und geschachtelt werden. Sie sind ein sehr mächtiges Mittel, um auch sehr komplexe Abhängigkeiten zwischen Code-Abschnitten auf deklarative Weise zu definieren. Eine Sonderrolle nimmt der Kommentarblock in der obersten Zeile der Pattern-Implementierung ein. Er kommentiert die Pattern-Implementierung und wird nicht generiert.

## HyperSenses in SCORE

DSTG setzt HyperSenses als Basistechnik für Generator-Software dann ein, wenn diese projektspezifisch hochgradig anpassbar sein soll oder wenn eine DSL eingesetzt

<sup>9</sup>Um Verwechslungen mit anderen Patterns, insbesondere Design Patterns, zu vermeiden, sprechen wir bei HyperSenses von Code Patterns.

werden soll. Im Zusammenhang mit SCORE spielen beide Aspekte eine Rolle: Zum einen ist HyperSenses in SCORE die Basistechnik für die Implementierung des jeweiligen Adapter-Generators, zum anderen werden im SCORE Composition Manager bestimmte Mapping-Definitionen durch HyperSenses-Patterns vorgenommen.

Der jeweilige Adapter-Generator ist spezifisch für eine technisch definierte Zielplattform. Für MINT wurde ein solcher Adapter-Generator für die Zielplattform C# / ADO.NET / Oracle neu entwickelt. Der generierte Quellcode ist jedoch nicht nur spezifisch für die Zielplattform, sondern auch anwendungsspezifisch, da die Definitionen im Composition Repository zum Teil anwendungsspezifisch sind. Sowohl die technische Zielplattform, die im Detail stark variieren kann, als auch projektspezifische Anforderungen – besonders in Bezug auf die Service-Schicht – erfordern hier eine hochgradige Anpassbarkeit des Adapter-Generators. Aus diesen Gründen wurde hier HyperSenses als Implementierungstechnik gewählt.

Im SCORE Composition Manager können Mapping-Definitionen auf verschiedene Weisen vorgenommen werden, unter anderem durch sogenannte Mapping-Patterns. Im Rahmen von MINT wurden mehrere neue Standard-Mapping-Patterns definiert, s. dazu den Abschnitt 6.5.1. Die Auswahl und Konfiguration dieser Patterns erfolgt mittels HyperSenses-Techniken. Dabei handelt es sich um eine in die grafische Oberfläche des Composition Manager integrierte DSL, wobei im Fall von Standard-Mapping-Patterns SCORE die Domäne ist. Mit HyperSenses können ggf. projektspezifische Mapping-Patterns hinzugefügt werden.

## 2.6. Visual Composer

SAP bietet mit dem Visual Composer ein Werkzeug an, mit dem es Domänenexperten ohne Programmierkenntnisse ermöglicht werden soll, Daten verschiedener Datenquellen in einem Prozess zu integrieren und die Ergebnisse webbasiert darzustellen. Da die Integration von Standardsoftware zu berücksichtigen ein Teil von MINT ist und SAP ein Werkzeug bewirbt, das über eine eigene domänenspezifische Sprache verfügt und Integrationsaspekte behandelt, wurde der SAP Visual Composer im Rahmen von MINT näher untersucht.

### 2.6.1. Motivation

SAP Visual Composer ist ein kommerzielles Produkt von SAP und ein wichtiger Bestandteil von SAP NetWeaver. Als ein plattformunabhängiges, webbasiertes MDA-Werkzeug bietet SAP Visual Composer die Möglichkeit, modellbasierte Anwendungen flexibel zu entwickeln, ohne zu programmieren. Ziel des MINT-Projekts ist es, eine modellgetriebene Integration von Standard-Software-Systemen mit individuellen Software-Systemen auf der Service- und Portal-Ebene zu ermöglichen. Mit dieser Evaluation soll die Betrachtung des Werkzeugs Visual Composer aus der SAP NetWeaver Plattform dargestellt und anhand der in Abschnitt 2.4.5 verwendeten Bewertungskriterien untersucht werden. Auf Basis der Ergebnisse kann entschieden werden, ob



eine Integration von Visual Composer mit dem MINT-Ansatz möglich beziehungsweise sinnvoll ist. Dazu wurde eine Evaluation an einem Szenario, bei dem ein Zugriff auf Kundendaten erfolgt, durchgeführt und im Folgenden beschrieben.

## 2.6.2. Einführung in SAP Visual Composer

**Visual Composer und SAP NetWeaver** SAP NetWeaver stellt Tools für ein neues Architekturkonzept mit dem Name Enterprise Service-Oriented Architecture (Enterprise SOA) bereit. Eines dieser neuen und viel versprechenden Entwicklungstools für die Erstellung von Anwendungen zur Nutzung von Enterprise Services ist der Visual Composer. [9] Der Visual Composer soll eine schnelle und einfache Entwicklung von Anwendungen und Prototypen über eine grafische Benutzeroberfläche ermöglichen, ohne dass Code manuell geschrieben werden muss. Anstatt sich auf technische Fähigkeiten und Programmierkenntnisse konzentrieren zu müssen, verknüpfen die Anwender einfach grafische Bausteine in einem Flussdiagramm miteinander. Als Ausgabemedium wird Flash verwendet. Der Visual Composer richtet sich somit an einen neuen Typ von Anwendungsentwickler, nämlich an Geschäftsanalysten und Geschäftsprozessexperten. Sie können mit ihren Kenntnissen über Geschäftsprozesse und Unternehmensanforderungen Visual Composer-Anwendungen modellieren, ohne programmieren zu müssen. Diese und vergleichbare Domänenexperten stehen auch im Fokus des MINT-Ansatzes.

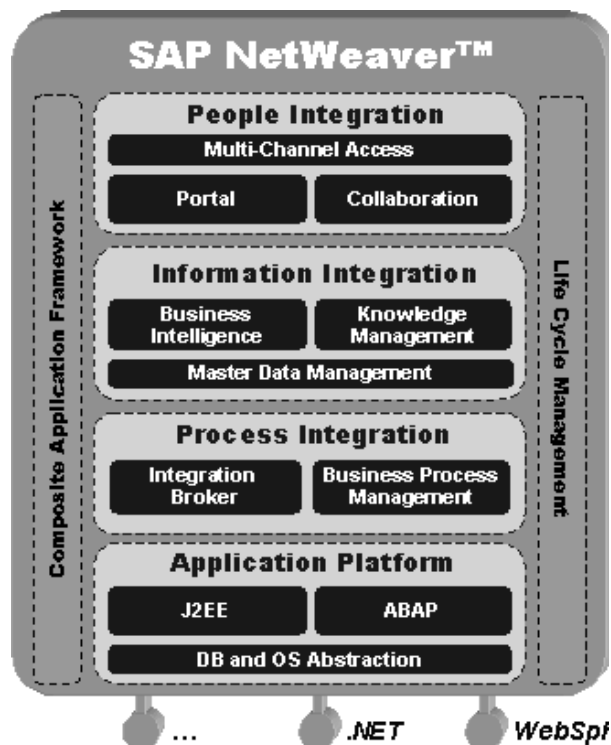


Abbildung 2.16.: SAP Netweaver Komponentenansicht (Quelle: SAP AG)

SAP NetWeaver umfasst verschiedene Schichten, drei Integrationsebenen und die Schicht „Application Platform“ als Basis von SAP NetWeaver. Die Schicht Application Platform wird durch den SAP NetWeaver Application Server realisiert. Anwendungen können entweder auf dem Java-Stack oder auf dem ABAP-Stack ausgeführt werden. Die Schicht Process Integration ermöglicht die system- und anwendungsübergreifende Anpassung und Integration von Prozessen. Diese Ebene besteht aus zwei Komponenten: Dem Integration Broker und dem Business Process Management. Der Integration Broker ermöglicht eine Infrastruktur für den XML-basierten Nachrichtenaustausch zwischen SAP und anderen Systemen. Über das Business Process Management werden Prozesse gesteuert. Beide Komponenten werden durch SAP NetWeaver Exchange Infrastruktur (SAP XI) implementiert. [9] Die zweite Integrationsebene von SAP NetWeaver ist die Schicht Information Integration, über die auf alle relevanten Daten des Unternehmens zugegriffen werden kann, unabhängig von deren Speicherort. In dieser Schicht sind drei Komponenten verfügbar:

- Mit der Komponente Master Data Management ist es möglich, Daten einheitlich und standortübergreifend in einer heterogenen IT-Landschaft zu speichern.
- Business Intelligence stellt Informationen zentral bereit, um Benutzer bei Kontroll- und Entscheidungsfindung zu unterstützen.
- Knowledge Management bietet Zugriff auf unstrukturierte Informationen und Dokumente aus verschiedenen Datenquellen und wird durch das Knowledge Management des SAP NetWeaver Portals implementiert.

In der Schicht People Integration sind drei Komponenten verfügbar: Portal, Collaboration-Funktionen und Multi-Channel-Zugriff. Der Multi-Channel-Zugriff wird durch SAP NetWeaver Mobile implementiert, wodurch von unterschiedlichen Geräten (z. B. Mobil-, Funk- und Sprachgeräten) über verschiedene Kanäle auf SAP-Anwendungen in Unternehmenssystemen zugegriffen werden kann. Die anderen zwei Komponenten werden durch SAP NetWeaver Portal implementiert. Das SAP NetWeaver Portal ist die strategische Plattform von SAP, bietet einen zentralen Einstieg zu allen Anwendung in Unternehmen über einen Webbrowser. Der Visual Composer wird neben dem SAP Composite Application Framework (CAF) als eines der wichtigsten Werkzeuge gesehen, denn er kann die Komposition und Orchestrierung von Enterprise Services zur Erstellung zusammengesetzter Applikationen gut unterstützen. Das CAF ist fester Bestandteil von SAP NetWeaver und bietet Entwicklern eine Entwicklungsumgebung für die Erstellung von Composite Applications. Composite Applications sind Anwendungen, die typischerweise nicht vollständig neu erstellt, sondern aus vorhandenen Komponenten zu einer neuen Anwendung zusammengestellt werden. Während mit dem SAP Visual Composer einfache Composite Applications bzw. zusammengestellte Sichten erstellt werden können, verfügt das SAP Composite Application Framework über eine größere Reichweite, die sich auf Geschäftsfälle erstreckt, die über den Bereich des Visual Composer hinausgehen. [58]

## Architektur von SAP Visual Composer

Die Architektur von SAP Visual Composer besteht aus drei Hauptkomponenten:

- Visual-Composer-Storyboard
- Visual-Composer-Server
- SAP Netweaver Portal

In der Abb. 2.17 werden die wichtigsten Komponenten der Architektur von SAP Visual Composer for NetWeaver 2004s dargestellt. Die Pfeilrichtung beschreibt nicht den Informationsfluss, sondern die Richtung der Bereitstellung.

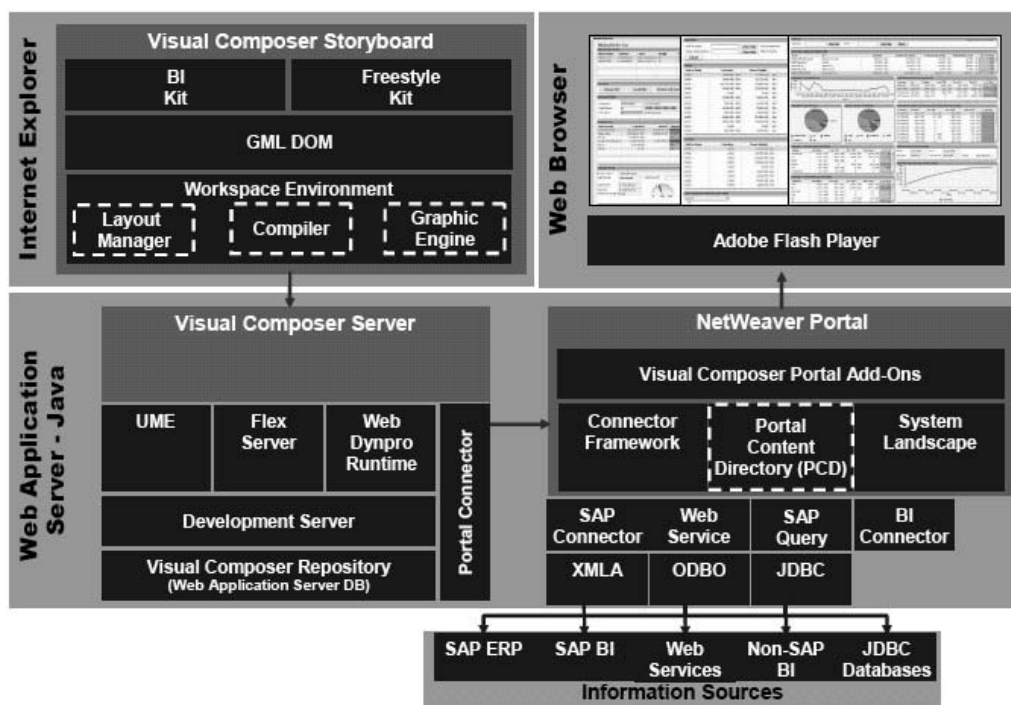


Abbildung 2.17.: Architektur von SAP Visual Composer for NetWeaver (Quelle: [9])

Bei dem Visual-Composer-Storyboard handelt es sich um den Visual-Composer-Client für die Design Time. Er enthält sämtliche notwendige Tools für die Modellierung von Anwendungen mit dem Visual Composer. Seine zwei Hauptkomponenten sind die Workspace-Umgebung und das Generic Modeling Language Document Object Model (GML DOM). In der Workspace-Umgebung sind weitere Komponenten wie der Layout-Manager, der Compiler und die Grafik-Engine enthalten. Die Darstellungen aller Elemente, die zum Modellieren einer Anwendung im Visual Composer verwendet werden können, sind im GML DOM in zwei Teilen gespeichert, eine Beschreibung des Elements

in Form von Metadaten im GML DOM und eine grafische Darstellung des Elements in skalierbarer Vektorgrafik (SVG). [9] Der Visual-Composer-Server stellt den einzigen Zugriffspunkt des Visual-Composer-Storyboard auf den SAP NetWeaver Application Server dar. Über diesen Server werden Anforderungen wie z.B. Benutzerauthentifizierung an die entsprechenden Komponenten gesendet. Der Entwicklungsserver stellt eine plattformunabhängige Schnittstelle zum Visual-Composer-Repository bereit. Dadurch werden im Visual Composer sämtliche Datenbanktypen unterstützt, die auch vom SAP NetWeaver Application Server unterstützt werden. Im Visual-Composer-Repository werden die im GML beschriebenen Modelle gespeichert [9].

Über den Portal Connector kann der Visual Composer mit dem Portal Connector Framework verbunden werden. Über die Portal-Konnektoren kann auf sämtliche Informationsquellen zugegriffen werden, für die ein Konnektor vorhanden ist. Zusätzlich stellt der Visual Composer einen Webservice-Konnektor zur Verfügung, so dass die gewünschten externen Webservices auch im Visual Composer verwendet werden können.

Die durch SAP NetWeaver Portal bereitgestellten Funktionen können noch über Visual-Composer-Port-Add-Ons eine direkte Verbindung mit dem SAP NetWeaver Portal herstellen. Das SAP NetWeaver Portal ermöglicht die Bereitstellung der Visual-Composer-Anwendungen für Endbenutzer. Nach dem Deployment im SAP NetWeaver Portal werden Anwendungen im Portal Content Directory (PCD) gespeichert. Auf die Services der SAP NetWeaver Portal-Systemlandschaft kann der Visual Composer auch zugreifen. Nur die Systeme, für die der angemeldete Benutzer über die entsprechenden Zugriffsberechtigungen verfügt, können im Visual Composer angezeigt werden. [9] Nach der Modellierung kann die Anwendung im Webbrowser bereitgestellt (Deploy Application) werden.

### 2.6.3. Anwendungsszenario mit Visual Composer

#### Anforderung

Das Anwendungsszenario gliedert sich in vier Teile:

1. Customer: Suche nach Kundendaten anhand Adressdaten
2. Address: Ausgabe der Adressdaten eines Kunden und seiner Bestellungen
3. Product: Tabellarische und grafische Darstellung der Produktdaten
4. Nested iView: Anwendung eines „Nested iView“

Diese vier Anwendungsteile sind parallel und unabhängig voneinander. Sie werden in Form von Tabreitern im SAP NetWeaver Portal bereitgestellt. Diese analytische Beispielanwendung dient hauptsächlich dazu, die wichtigsten Funktionen vom SAP Visual Composer zu testen bzw. zu untersuchen. Zwischen diesen vier Teilen gibt es keinen Zusammenhang.

Teil	Verwendung	Name	Quelle
1	Liste der Kunden	ISA_CUSTOMER_SEARCH	IDES_CORE
2	Abfrage der Kundenadresse	BAPLCUSTOMER_GETDETAIL	IDES_CORE
2	Abfrage der Kundenbestellungen	BAPLSALESORDER_GETLIST	IDES_CORE
3	Abfrage Produktdaten	Query: Product Group	BW_800
4	Abfrage des Verkaufsreiches der Kunden	BAPLCUSTOMER_GETSALESAREAS	IDES_CORE
4	Liste der Kunden	BAPLCUSTOMER_DISPLAY	IDES_CORE

Tabelle 2.2.: Informationsquellen

### Analyse der Informationsquelle

Im nächsten Schritt wird die Datenherkunft geklärt. Die notwendigen Informationen sind in mehreren BAPIs aus dem IDES\_CORE-System und in BW-Queries zu finden. Tabelle 2.2 gibt einen Überblick über alle identifizierten Informationsquellen für die Erstellung der analytischen Anwendung. An dieser Stelle wird auf eine ausführliche Definition der einzelnen Felder, die aus den Informationsquellen verwendet werden, nicht eingegangen.

### Design der Ablauflogik

Es werden zwei iViews angelegt, ein Source iView der Anwendung und ein Nested iView. Der Source iView enthält vier Layer, damit die vier Anwendungsteile auf verschiedene Layer aufgeteilt werden können. Ein Nested iView hat einen Eingangsparameter „Kundennummer“ und wird in dem Source iView angewendet. Abb. 2.18 zeigt die komplette Ablauflogik der Anwendung im Design Workspace. Das Navigation Control dieses iView wird auf „Tabstrip“ gesetzt, so dass die Gestaltung des iView in den vier unterschiedlichen Bereichen „Customer“, „Address“, „Product“ und „Nested iView“ erfolgt.

Im Default-Layer „Customer“ wird die Suchfunktion der Kundendaten modelliert. Wird eine Adresse eingegeben, werden die entsprechenden Kundendaten mit Sortierungsfunktion in der Tabelle aufgelistet.

In dem zweiten Layer „Address“ werden zwei unterschiedliche Datendienste mit dem Eingabeformular verbunden, wobei ein Data Mapping aufgrund verschiedener Daten-

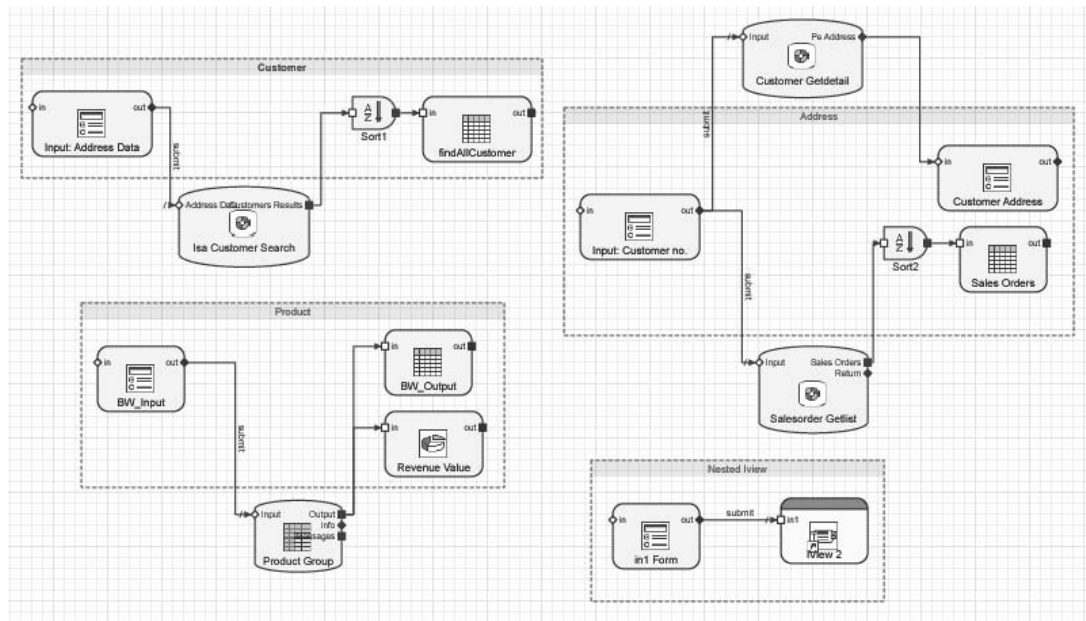


Abbildung 2.18.: Source iView

bezeichnungen definiert wird.

Des Weiteren wird ein Datendienst im Layer „Product“ mit zwei verschiedenen Ausgabekomponenten angebinden. Gibt der Benutzer Suchkriterien der Produktdaten im Eingabeformular ein und betätigt den Button „Submit“, werden die Produkterlöse sowohl tabellarisch als auch grafisch dargestellt.

Das Layer „Nested iView“ enthält ein Eingabeformular und einen Nested iView. Ein Nested iView wird zur Modularisierung von Informationsblöcken verwendet und aus einem Source iView aufgerufen. Mit dem Nested iView können Daten empfangen, dargestellt oder wieder zurückgegeben werden. Der Nested iView wird im Layout Workspace als ein eigener Container angelegt, ohne dass die darin enthaltenen Komponenten und deren Größe ersichtlich sind. Die Elemente in einem Container werden jeweils innerhalb ihres eigenen Nested iView gestaltet (siehe Abb. 2.19). In diesem Nested iView werden zwei Layer mit den Namen „Salesareas“ und „Contactaddressdata“ definiert. Als Navigation Control wird „Tabstrip“ ausgewählt.

### Gestaltung des Layouts und Konfiguration der Elemente

Das Layout von kleinen Anwendungen lässt sich schnell und intuitiv direkt im Layout Workspace erstellen. Bei der Ausgabe werden die Elemente in der Größe aneinander angepasst und ausgerichtet. Am unteren linken Bildschirmrand im Layout Workspace werden die Koordinaten angezeigt. Wer ein exaktes Layout haben möchte, muss selbst die Komponentenposition pixelgenau vorberechnen und per Maus im Layout Workspace die Komponenten positionieren und die Größen festlegen.

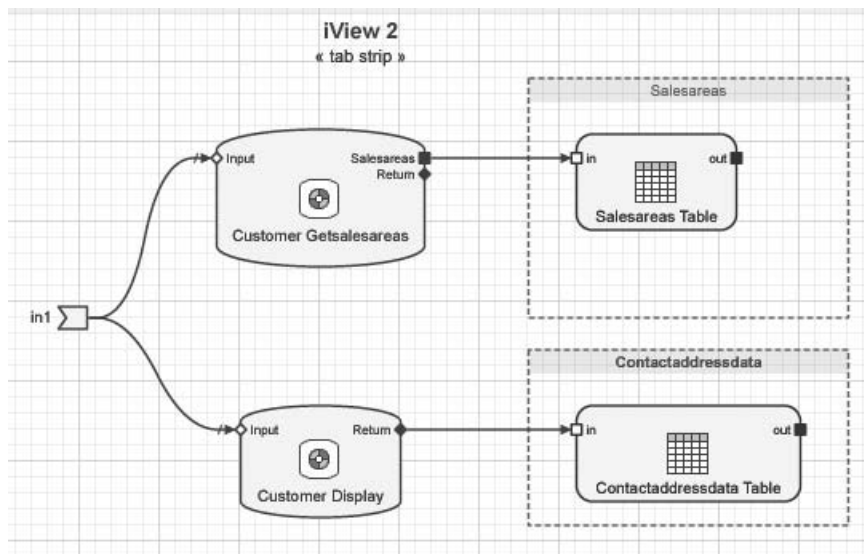


Abbildung 2.19.: Nested iView

Abb. 2.20 ist ein Screenshot der Ansicht „Customer“. Gibt der Anwender zu dem Feld „Post Code“ „190\*“ ein, werden alle Kunden, deren Postleitzahl mit „190“ anfängt, zurückgeliefert und in der Tabelle aufgelistet. Die einzelnen Spaltengrößen können verändert werden. Die Position der einzelnen Komponenten ist im Webbrowser nicht veränderbar. Mit dem Visual Composer erstellte iViews bieten an sich keine Personalisierungsfunktionalität. Eine Personalisierung ermöglicht dem Endbenutzer eine moderate Änderung des Layouts, von Stylesheets und andere Einstellungen. Das SAP NetWeaver ist ein Beispiel für ein System, das eine Personalisierung unterstützt [9].

In der Abb. 2.21 wird die zweite Ansicht „Address“ dargestellt. Für die zwei Pflichtfelder werden Default-Werte definiert. Mit dem Data Mapping lassen sich zwei Datenquellen mit einem Eingabeformular verbinden. Wird der Button „Submit“ betätigt, werden entsprechende Daten unten angezeigt. Auf die einzelnen Spalten der Tabelle „Sales Orders“ werden Sortierungsfunktionen definiert. Beim Klicken auf die Spaltenüberschrift in der Tabelle, werden Dateneinträge aufsteigend oder absteigend sortiert. Hierbei handelt es sich um eine clientseitige Sortierung.

#### 2.6.4. Ergebnisse

SAP Visual Composer ist ein interessantes Modellierungswerkzeug sowie ein fertiges Produkt für einen eng umrissenen Einsatzzweck. Dieser eng umrissene Einsatzzweck ermöglicht es ein Werkzeug zu erstellen, das auch von Domänenexperten zu bedienen ist und für viele Anwendungsfälle keine Programmierung mehr erforderlich macht. Gleichzeitig beschränkt der eng umrissene Einsatzzweck sowie die alleinige Fokussierung auf die SAP-Welt eine Integration des Visual Composers in das in diesem Dokument beschriebene MINT-Vorgehen. Für die SAP-spezifische Modellierungssprache GML ist

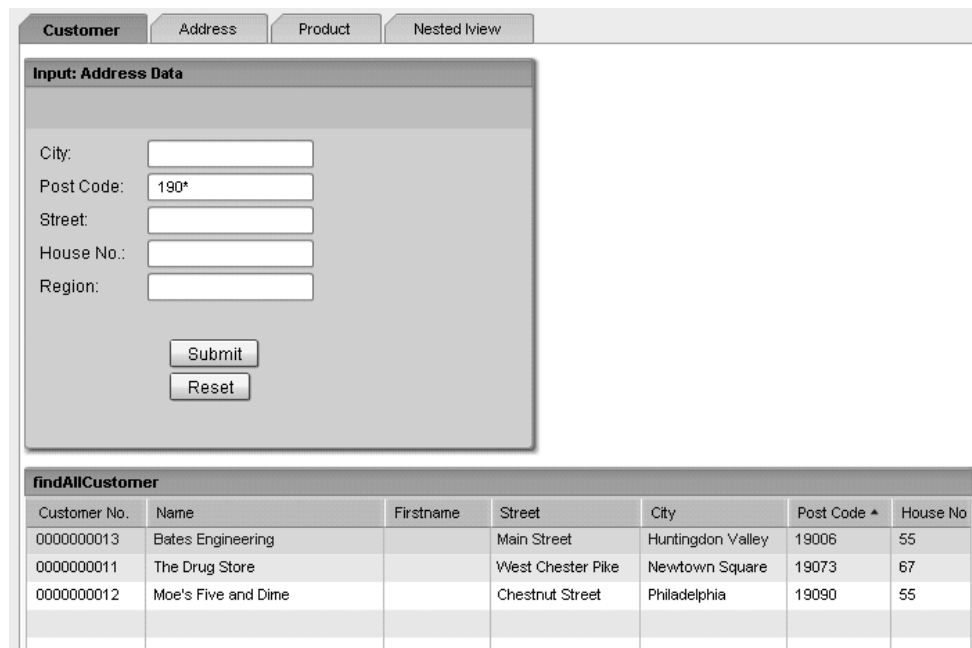


Abbildung 2.20.: Ansicht „Customer“

kaum Dokumentation verfügbar. Die verschiedenen Versionen des Visual Composers weichen an mehreren Stellen, wie z. B. in Bezug auf die GML-Sprache, voneinander ab. Für die nächste Version des Visual Composers plant die Firma SAP, GML durch GML+ zu ersetzen. Deshalb ist die Wahrscheinlichkeit einer Inkompatibilität sehr hoch und für diese Fälle bietet SAP keinen Support an, da ein Upgrade eines Modells offiziell nicht unterstützt wird.

Als ein fast vollständig modellbasiertes Werkzeug ermöglicht SAP Visual Composer eine kurze Entwicklungszeit einer Standardanwendung. Trotzdem kann unter bestimmten Umständen auch im Rahmen von Visual-Composer-Projekten Programmierung notwendig sein. Der Visual Composer basiert auf bereits vorhandenen Funktionalitäten, die durch Entwickler definiert werden müssen. Was in einem Projekt den größeren Teil der Entwicklungszeit in Anspruch nehmen wird, ist die Analyse der Geschäftsprozesse, die Ausarbeitung der Anwenderprofile und Benutzerrollen sowie die Identifikation der Datenquellen. In künftigen Versionen wird sich der Funktionsumfang des Visual Composers erweitern. Entwicklungsschwerpunkte gibt es zu folgenden Themen: [14]

**Enterprise Services** Der Visual Composer dient als Integrationsframework für Enterprise Services, mit dem sich die zusammengesetzten Applikationen realisieren lassen.

**Composite Application Framework und Guided Procedures** Es wird eine stärkere Integration des Visual Composer in das Composite Application Framework sowie einen Ausbau der Integration in Richtung Guided Procedure Framework geben.



The screenshot shows a web application interface with a navigation bar at the top containing tabs for 'Customer', 'Address', 'Product', and 'Nested View'. The 'Address' tab is active. Below the navigation bar, there are two main sections:

**Input: Customer no.**

Customerno:  \*

Salesorganisation:  \*

**Customer Address**

First Name:

Name:

Date Birth:

Street:

Postl Code:

City:

Region:

Country:

Telephone:

**Sales Orders**

Currency	Doc Date	Exchg Rate	Purch No	Division
USD	Aug 04, 2006	1.00	111	01
USD	Aug 04, 2006	1.00	112	01
USD	Aug 02, 2006	1.00	123	01
USD	Aug 03, 2006	1.00	123	01
USD	Aug 01, 2006	1.00	123	01
USD	Aug 01, 2006	1.00	123	01
USD	Aug 01, 2006	1.00	123	01
USD	Aug 02, 2006	1.00	123	01
USD	Aug 03, 2006	1.00	123	01
USD	Aug 01, 2006	1.00	123	01
USD	Aug 15, 2006	1.00	1123	01
USD	Aug 07, 2006	1.00	1234	01
USD	Aug 02, 2006	1.00	1234	01
USD	Aug 03, 2006	1.00	1235	00

Abbildung 2.21.: Ansicht „Address“

**Lifecycle Management** Bislang unterstützt der Visual Composer nicht die Standards von SAP für das Lifecycle Management, z. B. Versionierung, Transportwesen usw. Die entsprechende Realisierung ist aktuell sehr eingeschränkt und teilweise mit großem Aufwand verbunden. Einer der aktuellen Entwicklungsschwerpunkte ist die Behebung dieser funktionalen Lücken.

**Adobe Flex 2.0** Es soll eine Portierung auf die neue Flex 2.0-Technologie stattfinden. Dies wird einige der großen Probleme beseitigen, u. a. das Problem der 32K-Grenze beim Deployment von iViews.

**SDK (Kit Development)** Es soll ein eigenes Software Development Kit zur Entwicklung eigener Erweiterungs-Kits für den Visual Composer bereitgestellt werden.

Tabelle 2.3 fasst die Ergebnisse analog zu Tabelle 2.1 zusammen, obwohl die in Abschnitt 2.4 beschriebenen Transformationswerkzeuge nicht direkt mit Visual Composer zu vergleichen sind.

## 2.7. Anwendungsszenarien für die Prozessintegration

In diesem Abschnitt werden mögliche Anwendungsfälle skizziert, die zwischen real existierenden Systemen eine Integration notwendig machen. Bei den enthaltenen Systemen kann es sich sowohl um spezialisierte Eigenentwicklungen als auch um angepasste Standardsoftware handeln.

Für die notwendige Integration zwischen diesen Systemen soll das im Rahmen des MINT-Projekts entwickelte Vorgehensmodell (siehe Abschnitt 5) eingesetzt werden.

<b>Kriterium</b>	<b>SAP Visual Composer</b>
MOF	Unklar
UML	Nein
XMI	Nein
XMI Tools	Nein
XMI Light	Nein
MDA Ebenen	PSM (DSL), Code
Wertebereiche	Nicht relevant
Primärschlüssel	Nicht relevant
Unique Attribute	Nicht relevant
Tagged Values	Ja
Model2Model	Nein
Model2Code	Ja
Zielsprachen / Frameworks	- Die Beschreibung des Elements in Form von Metadaten im GML DOM. - Die grafische Darstellung des Elements in SVG
Templates anpassbar	Nein
Eigene Templates	Nein
Mehrfachvererbung	Nein
Forward Engineering	Ja
Round-Trip	Nein
Reverse Engineering	Nicht empfohlen von SAP; Fehlende Dokumente
Unit Testing	Noch nicht
Mehrbenutzerfähig	Nein
Versionierung	- Versionverwaltung mit einem Quellcodeverwaltungstool möglich; - Inkompatibilität zwischen beiden Versionen von Visual Composer
Preis	Enthalten in der Lizenz für „SAP Netweaver 4.0s“ oder der „SAP Business Suite Professional User Lizenz“
Dokumentation	vorhanden

Tabelle 2.3.: Evaluationsergebnis

Die Ergebnisse dieses Integrationsvorhabens dienen als Basis für die Evaluation des umgesetzten Konzepts des MINT-Projekts.

### 2.7.1. Störungserfassungsdomäne

Ein Callcenter-Mitarbeiter eines Energieversorgers empfängt einen Telefonanruf eines Kunden, der eine Störung in seiner Stromversorgung meldet. Der Callcenter-Mitarbeiter fragt die Störungssparte (in diesem Fall Strom), die Daten des Anrufers (Namen, Telefon) und den Störungsort (Wohnort, Plz, Strasse, Hausnr, usw.) beim Kunden ab und trägt sie in das Störungserfassungssystem ein. Im Anschluss überprüft der Callcenter-Mitarbeiter, ob sich der Störungsort im Versorgungsgebiet des Energieversorgers befindet. Ist dieses nicht der Fall, wird der Kunde an den zuständigen Versorger verwiesen. Liegt die Störung im Versorgungsgebiet, werden die Störungsdaten an die GIS-Verarbeitung gesendet. Diese zeigt auf einer Karte alle Störungen die in der Peripherie des Störungsorts des anrufenden Kunden liegen. Anhand der Karte kann der Callcenter-Mitarbeiter entscheiden, ob die Störung bereits von einem anderem Kunden gemeldet wurde. Ist dieses der Fall, teilt er dem Kunden mit, dass an der Störungsbehebung bereits gearbeitet wird. Meldet der Kunde eine neue Störung wird diese in einem neuen Störungsfall im Störungserfassungssystem angelegt und die Außenstelle zur Störungsbehebung per Email und per SMS über die Störung benachrichtigt.

Die zu integrierenden Systeme für dieses Szenario sind:

- **Störungserfassungssystem** Die Software wurde als komplette Neuentwicklung erstellt und ist nicht ausschließlich auf die Erfassung von Störungen in Stromnetzen ausgerichtet, sondern wird auch für Störungen im Gas- und Wasserbereich eingesetzt. Bei der Entwicklung des Systems wurde ein modellgetriebener Ansatz verfolgt bei dem die Modellierung auf UML-Basis und die automatische Generierung des Quellcode einen zentralen Punkt in der Entwicklung des Systems einnahmen. Neben der Schnittstelle zur Annahme von Störungen mittels einer grafischen webbasierten Oberfläche bestehen Webservice -Schnittstellen zur Erweiterung und Abfrage des Systems.
- System zur Darstellung der Störfälle auf einer Karte (Standardsoftware)
- System für den SMS-Versand und den Email-Versand (Standardsoftware)

Dieses Integrationsszenario wird als Beispielszenario für das in Kapitel 5 beschriebene Vorgehensmodell verwendet.

### 2.7.2. Instandhaltungsdomäne

Der Standpunkt eines Offshore-Windparks befindet sich im Spannungsfeld zwischen maximalem Ertrag, Aufbaukosten und möglichst minimalen Instandhaltungskosten. Dennoch wird es möglich sein, dass die Anlage etliche Kilometer vor der Küste liegt

und eine großflächige Lagerhaltung von Ersatz- und Verschleißteilen auf See nicht rentabel möglich ist. Das bedeutet, dass bei den meisten Wartungsintervallen Ersatz- und Verschleißteile von den Wartungsteams mit zu den Anlagen gebracht werden müssen. Insbesondere der Transport stellt sich als Herausforderung an die Planung heraus. Zum einen kommen nur sehr wenige Verkehrsmittel für den Transport in Frage. Der übliche Transport von Teilen und Wartungsteams wird per Schiff durchgeführt. Das ist jedoch von unterschiedlichen Faktoren abhängig. Offshore-Windparks sind unter Umständen nur an 20 % aller Tage im Jahr per Schiff zu erreichen. An den übrigen Tagen machen starker Wind und hoher Wellengang das Erreichen der Plattform über den Wasserweg nicht möglich. Das bedeutet, dass für die Zeit, in der das Wetter für Schiffe ungeeignet ist, ein Helikopter zum Einsatz kommt. Dieses ist jedoch auch nur innerhalb der zulässigen Wetterbedingungen möglich. Des Weiteren können nicht alle Ersatzteile mit dem Helikopter transportiert bzw. bei schlechtem Wetter an die Windparkanlage geliefert werden. Zum anderen stellt sich die Wartungs- und Instandhaltungsplanung als komplexes Problem heraus. Aus versicherungstechnischen Gründen sind bestimmte Wartungsintervalle nicht überschreitbar. Zum jeweiligen Wartungstermin müssen alle Wartungsteile vor Ort sein, außerdem müssen für die Instandhaltung bzw. Instandsetzung der Anlage Teile vor Ort sein. Aus Kostengründen werden nur solche Schiffsgrößen eingesetzt die für den jeweiligen Auftrag auch ausgelastet sind und nicht nur zum Teil beladen sind. Die Zusammensetzung der Wartungsteams unterliegt bestimmten gesetzlichen Vorschriften, beispielsweise muß einem Team immer ein Erste-Hilfe-Kundiger angehören. Des Weiteren sind für den Einsatz in Offshore-Anlagen weitere Lehrgänge und Kurse vorgeschrieben. Beinahe unabhängig vom Wetter können bestimmte Störfälle auch ein sehr schnelles Eingreifen erfordern. So gestaltet sich die Wartung und Instandhaltung als ein Thema, das von vielen auch nicht vorhersehbaren Faktoren abhängt. Während die Wartung einer Anlage bestimmten Intervallen unterliegt, die aus witterungstechnischen Gründen zum größten Teil in den Sommermonaten liegen und somit zumindest mit Ausnahme des Wetters planbar sind, ist die Instandsetzung im Vorfeld nur bedingt planbar. Im weiteren Verlauf wird ein Szenario skizziert, das im MINT-Kontext untersucht wurde.

Eine Offshore-Anlage meldet auf der Konsole eines Überwachungstechnikers einen Fehler. Dieser entscheidet, ob der Fehler bis zur nächsten routinemäßigen Wartung tolerierbar ist, sofort an einen der Hersteller der Anlage weitergeleitet wird oder ob er selbstständig eine Fehlerbehebung durchführen kann. Sollte der Fehler schwerwiegender und nur durch menschlichen Einsatz vor Ort zu beheben sein, wird ein Instandsetzungseinsatz geplant. Zuerst wird ermittelt ob der Fehler den Austausch von Anlagenteilen erforderlich macht und ob diese Ersatzteile vor Ort sind. Sind die benötigten Teile nicht vor Ort, wird in der Planung nachgeschaut, ob bei der nächsten Routine-Wartung die erforderlichen Teile mit transportiert werden können und der aufgetretene Fehler diese möglicherweise mehrtägige Verzögerung erlaubt. Sollte das der Fall sein, kann der Fehlerbehebungsauftrag bezüglich des benötigten Teams untersucht werden. Unter Umständen werden Spezialisten benötigt, die im Normalfall nicht den Instandsetzungsteams angehören. Zum Abschluss können alle benötigten Teile bestellt, die Teams zusammengestellt, das benötigte Verkehrsmittel bestellt und ein Ausweichplan

erstellt werden. Für die im Szenario beschriebenen Aufgaben werden unterschiedliche Systeme verwendet. Neben der Windparkanlage wird ein Windparkmanagementsystem und ein Netzleitsystem für die Überwachung der Anlage verwendet. Das Off- und Onshore-Lager wird mit einem Lagerverwaltungssystem gemanagt. Für die Team- und Personalplanung ist ein Personalplanungssystem zuständig. Die Durchführung der unterschiedlichen Diagnose-, Wartungs- und Instandhaltungsaufgaben wird nach Bedarf an externe Dienstleister delegiert.

## 2.8. Die andrena Testumgebung

Eine der ersten Entscheidungen, die getroffen werden musste, ist die Frage nach der Testumgebung. Einerseits sollte die Implementierung und die Architektur so einfach gehalten sein, dass ein einfaches Austauschen der Persistenzschicht möglich ist und sich diverse Messungen vornehmen lassen. Auf der anderen Seite sollte es möglichst auch eine *real world* Applikation sein, die die große Menge an sich im Einsatz befindlichen Altsystemen repräsentiert.

Die Testumgebung basiert auf *MESCOR*, einer Programmsuite zur Analyse von Unternehmen und deren Wertpapieren. Diese Programmsuite besteht aus mehreren, eigenständigen Applikationen, die eine gemeinsame Server-Mittelschicht benutzen. Diese Mittelschicht stellt die Kommunikation mit dem Datenbanksystem bereit.

Die gesamten Anwendungsdaten werden in einer zentralen Oracle-Datenbank gespeichert. Diese enthält mehr als 100 Tabellen. Die gespeicherten Informationen kommen aus verschiedenen Domänen der Finanzindustrie, bezogen beispielsweise auf Wertpapiere, Analysten, Industriesektoren und davon abhängigen Daten.

*MESCOR* ist seit einem Jahrzehnt im täglichen Einsatz bei verschiedenen Kunden aus der Finanzindustrie. Dieses System wird von der andrena objects ag gewartet und weiterentwickelt. Die verwendeten Clients sowie der Server sind in Borland Delphi implementiert. Die Kommunikation findet mittels DCOM und Sockets statt. Seit dem Jahr 2005 wurden einige Teile des Systems auf die Microsoft .NET Plattform migriert. Als Vorarbeit zu diesem Forschungsvorhaben wurde eine Analyse der relevanten Anwendungsfälle sowie der Datenbank durchgeführt um die Teile zu identifizieren und auszuwählen, die für das Forschungsvorhaben relevant sein könnten.

Die ersten Messungen werden für die *ChartProduction* durchgeführt. Dieses Werkzeug erstellt Verlaufsdiagramme (Charts) für die in der Datenbank hinterlegten Aktienkurse. Es ist einfach zu benutzen, jedoch erfordert es eine hochgradig performante Datenzugriffsschicht. Daher ist dieses Werkzeug ein geeignetes Beispiel um die Geschwindigkeit für die verschiedenen Mappingansätze zu messen.

Die anderen betrachteten Applikationen, *Accounting* und *ImportTool*, setzen ihren Fokus auf komplexe Lese- und Schreiboperationen (Objekt-Bäume anzeigen, Listen aktualisieren, etc.) oder auf Benutzerinteraktionen (kurze Antwortzeiten).

Die Bewertung wird sich darauf beschränken, die verschiedenen Persistenzlösungen zu vergleichen. Es wurde eine allgemeine Testumgebungsarchitektur erstellt (siehe Abbildung 2.22), um eine gemeinsame Methode zur Verfügung zu stellen, das Persis-

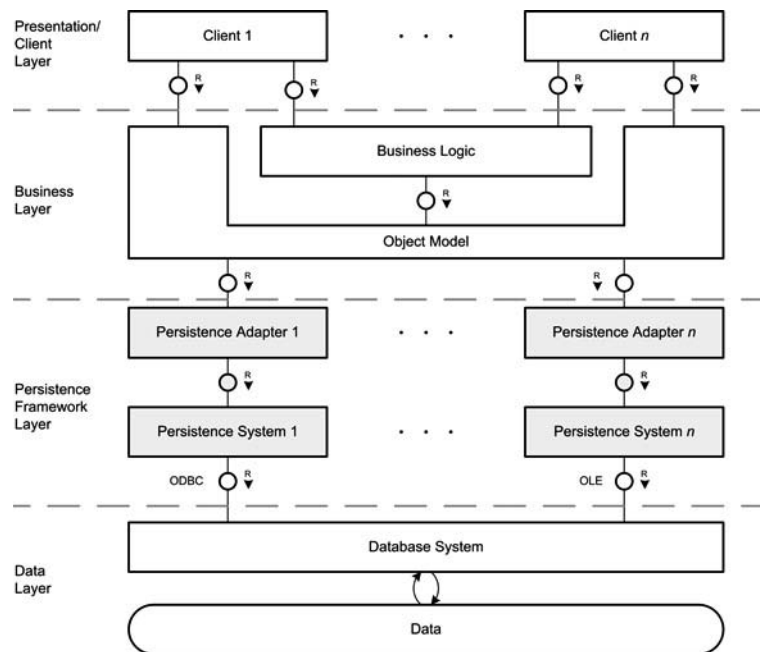


Abbildung 2.22.: Architektur der Testumgebung

tenzsystem auszutauschen und die jeweiligen Metriken zusammenzustellen.

Die Applikationen der Testumgebung sind in einer 4-Schichten-Architektur untergebracht. Es gibt eine Datenschicht (data layer), eine Persistenzframeworkschicht (persistence framework layer), eine Businessschicht (business layer) und eine Clientschicht (client layer). Diese werden nachfolgend kurz erläutert.

**Datenschicht (data layer)** Für alle Applikationen aus *MESCOR* wird eine gemeinsame Oracle-Datenbank benutzt.

**Persistenzframeworkschicht (persistence framework layer)** Dies ist die wichtigste Schicht in der Testumgebung. In dieser Schicht werden die verschiedenen Persistenzsysteme (z. B. ADO.NET, NHibernate, MINT-Ansatz) gemessen und verglichen. Jedes Persistenzsystem muss gemeinsame Schnittstellen implementieren, damit die Verbindung zur Businessschicht hergestellt werden kann. Die Vermittlung zwischen den Schnittstellen und dem Persistenzsystem wird durch einen individuellen Persistenzadapter für das jeweilige System hergestellt. Es wurde das Fassaden-Muster für die Schnittstellendefinition und Implementierung in dieser Schicht herangezogen.

**Die Business- und Clientschicht (business and client layer)** Diese Schichten sind von der Persistenzschicht getrennt und kennen nur die allgemeine Schnittstelle. Daher ist es nicht notwendig diese Schichten beim Wechsel des Persistenzsystems auszutauschen.

Alle Schichten tauschen ihre Informationen über ein gemeinsames Objektmodell aus. Dieses Objektmodell wurde in UML erstellt und orientiert sich nahe an der Struktur der Altdatenbank (bottom up design). Das Objektmodell besteht hauptsächlich aus einfachen DTOs (data transfer objects). Seitdem die Testumgebung in C# implementiert wurde, sind diese Objekte als so genannte POCOs (plain old c# objects) implementiert.

Die Persistenzframeworkschicht muss die Abfrageergebnisse also auch in Objekten des Objektmodells zurückgeben. Alle Schichten kennen nur die Schnittstellendefinitionen des Objektmodells. Daher kann, falls notwendig, jede eingesetzte Persistenztechnologie eine eigene Implementierung des Objektmodells verwenden.

Zusammenfassend kann man sagen, die Testumgebung ist eine Ansammlung von Applikationen, die alle die gleiche verallgemeinerte Testumgebungsarchitektur verwenden. Die Beispielapplikationen konzentrieren sich auf verschiedene Nutzungsszenarien, um die konkurrierenden Persistenzsysteme vergleichen zu können. Die Veränderung liegt einzig in der Persistenzschicht, die die bestimmte Persistenztechnologie und den spezifischen Persistenzadapter enthalten. Die anderen drei Schichten bleiben für die Applikationen unverändert. Jede Technologie muss die gleiche Funktionalität zur Verfügung stellen, um vergleichbare Merkmale zu bekommen.





# 3. Domänenspezifische Modellierung mit der MINT-XL

*Steffen Kruse, Ulrike Steffens, Niels Streekmann*

Dieses Kapitel beschreibt die im Projekt entwickelte Sprache zur domänenspezifischen Modellierung von Integrationsaspekten, MINT-XL (Extensible Language for Model-Driven Integration). Der Hauptbestandteil der Sprache sind Konstrukte der Prozessmodellierung. Zusätzlich lässt sich Datenintegration mit Hilfe sogenannter Mappings beschreiben. Die MINT-XL wird zur fachlichen Prozessmodellierung im in Kapitel 5 beschriebenen Vorgehensmodell verwendet. Die Konstrukte zur Datenintegration werden dabei als Abbildung der involvierten Schnittstellen der beteiligten Systeme verwendet. Diese lassen sich auch im Kontext der in Kapitel 6 beschriebenen Datenintegration zur Abbildung von Daten auf Datenschemaebene für Datenbanken nutzen. Zudem wird die MINT-XL wie in Kapitel 4 beschrieben zur Modellierung kognitiver Muster im Zusammenhang mit der Umsetzung wissensintensiver Prozesse verwendet.

Die MINT-XL ist eine graphische Sprache von hohem Abstraktionsgrad und eignet sich für die berechnungsunabhängige Modellierung von Geschäftsprozessen im Sinne der *Model Driven Architecture* (siehe 2.1). Im Gegensatz zu früheren (rein textuellen) Sprachen zur Domänenprogrammierung [32], bietet die MINT-XL eine modulare, einfach zu erweiternde Basis zur Generierung von plattformunabhängigen Architekturmodellen nach dem MINT-Vorgehen. Dabei kommen die typischen Vorteile zum Tragen, die üblicherweise DSLs zugesprochen werden: die erhöhte Verwendbarkeit durch Domänenexperten durch die Nähe zur Domäne und die flexible Erweiterbarkeit. Beide sind von besonderer Bedeutung im Bereich der Integration betrieblicher Informationssysteme.

## 3.1. Domänenspezifische Modellierung von Integrationsaspekten

Der Fokus des MINT-Projekts liegt auf der Integration von prozessorientierten, geschäftlichen Informationssystemen. Deshalb hat die MINT-XL eine starke prozessorientierte Ausprägung. Jedoch sind Integrationsaufgaben immer eingebettet in die jeweiligen Geschäftsdomänen des durchführenden Unternehmens. Es müssen also Anforderungen sowohl technischer Natur als auch aus diversen fachlichen Domänen berücksichtigt werden um Integrationsaufgaben erfolgreich zu bewältigen. Dabei sind statische, monolithische Modellierungssprachen aus mehreren Gründen als ungeeignet

zu bewerten um alle Aspekte einer Integration mit angemessenem Aufwand abzubilden.

Die Entwicklung einer DSL ist eine umfassende und zeitintensive Aufgabe, die die Identifizierung und Eingrenzung des Problemraums, den Transfer von Domänenwissen in geeignete semantische und syntaktische Sprachkonstrukte und die Erstellung von Abbildungen zwischen der Sprache und mindestens einer technischen Zielplattform beinhaltet [10]. Diese Plattform kann eine Implementierungsbibliothek (wie oft im Falle der domänenspezifischen Programmierung) oder wie im MINT-Projekt im Kontext der modellgetriebenen Entwicklung eine Modellebene mit technischer statt fachlicher Ausprägung sein. Im letzteren Fall dienen (semi-)automatische Transformationen aus der Sprache zu technischen Modellen häufig als Abbildung.

Da die Entwicklung einer DSL zusätzlichen Entwicklungsaufwand bedeutet, ist die Eingrenzung der geeigneten Problemdomäne von besonderer Bedeutung. [65] empfiehlt die Wahl von ausgereiften, stabilen und zweckmäßigen Domänen. [10] hingegen beachtet, dass die meisten Domänen Veränderlichkeiten unterliegen, die bei der Konstruktion der jeweiligen DSL möglichst berücksichtigt werden müssen. Diese Anforderungen motivieren die Modularisierung der MINT-XL nach Domänen und Anwendungsbereichen. Unter der Annahme, dass die Integration von Informationssystemen eine bekannte und sich wiederholende Aufgabe im Gegensatz zur Erfassung spezifischer Geschäftsprozessen darstellt, wird erwartet, dass Integrationsanteile der MINT-XL generisch und längerfristig stabil bleiben, während domänenspezifische Erweiterungen sich heterogen darstellen und Veränderungen der Geschäftsmodelle unterliegen. Aus diesem Grund verbessern die MINT-XL Erweiterungsmechanismen und die Modularisierung die Eignung der Sprache für geschäftsspezifische Integrationsaufgaben, wobei die Wiederverwendung gleichbleibender Aufgabenbereiche den Entwicklungsaufwand gegenüber einer monolithischen, hoch spezialisierten Sprache deutlich verringert.

Grundlegend bei der Entwicklung der MINT-XL ist die Annahme, dass die Integration von betrieblichen Informationssystemen heutzutage basierend auf Geschäftsprozessen und nicht Objekten durchgeführt wird. Dazu lassen sich diverse prozessorientierte Integrationsansätze finden. Am prominentesten ist sicherlich Service-Orchestrierung in serviceorientierten Architekturen (SOA) [47]. Das Konzept der prozessbasierten Integration ist jedoch älter [6]. Außerdem existieren diverse Prozessmodellierungssprachen, [59, 57, 44, 2, 41] welche in spezifischen Kontexten und Aufgabenbereichen entwickelt wurden.

Integrationsprojekte auf fachlicher Ebene erfordern eine enge Zusammenarbeit zwischen Domänenexperten und IT-Experten. Zusätzliche Interessengruppen müssen unter Umständen an dem Integrationsprozess beteiligt werden um alle Geschäftsinteressen zu wahren, da Informationssysteme häufig viele Geschäftsbereiche und Organisationseinheiten berühren und mit Unternehmenszielen abgestimmt werden müssen. Es ist davon auszugehen, dass nicht alle beteiligten Personen über ausreichende Kenntnisse der technischen Domäne verfügen um mit klassischen IT-Prozessmodellierungssprachen umgehen zu können. Aus diesem Grund ist eine Modellierungssprache gefordert, die einerseits von technischen Details abstrahiert und andererseits auf fachlicher Ebene semantisch eindeutige Konstrukte bietet um eine nachfolgende Transformation zu einer

technischen Sicht zu ermöglichen.

### 3.2. MINT-XL

Die MINT-XL (Extensible Language for Model Driven Integration) ist eine erweiterbare Sprache zur Modellierung der fachlichen Sicht von integrationsrelevanten Aspekten. Sie ist mittels der Complete Meta Object Facility (CMOF) [45] spezifiziert und ist somit vom Abstraktionsgrad auf einer Ebene mit der UML anzusiedeln. Verschiedene Erweiterungsmechanismen erlauben die Anpassung der MINT-XL, um den fachlichen Eigenschaften unterschiedlicher Domänen Rechnung zu tragen.

In diesem Abschnitt wird die grundlegende Struktur der Sprache vorgestellt. Darauf folgen die Beschreibungen der gebotenen Mechanismen zur Integration eigener, domänenspezifischer Erweiterungen und umgesetzter Erweiterungen.

Im Folgenden werden die Struktur der MINT-XL und die grundlegenden Sprachelemente vorgestellt. Die Sprache ist bezogen auf Abstraktionsgrad und Domäne in Pakete unterteilt. Pakete importieren nach Bedarf andere Pakete um Elemente und Eigenschaften per Generalisierung wieder zu verwenden. Assoziationen zwischen Elementen können über Subklassen verfeinert oder überschrieben werden um die zulässigen Assoziationspartner nach Bedarf einzuschränken. Dabei werden die von der MOF gebotenen Mechanismen zur Sprachdefinition genutzt. Abbildung 3.1 zeigt eine Übersicht der MINT-XL Struktur. Im Folgenden werden die einzelnen Pakete und die enthaltenen Elemente vorgestellt.

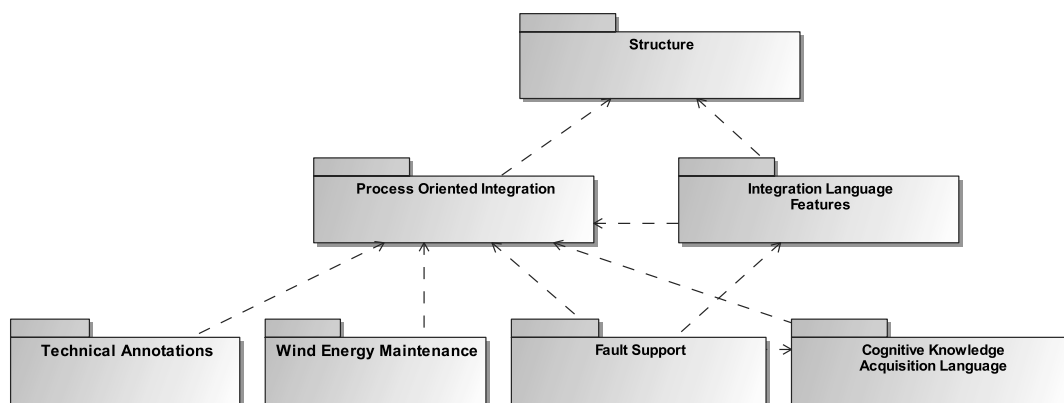


Abbildung 3.1.: MINT-XL Sprachstruktur

Im *Structure*-Paket wird der grundlegende Zusammenhang zwischen Sprachelementen, deren Modellen und natürlichsprachlichen Annotationen (Kommentare) und dem primären Erweiterungsmechanismus (Features) geschaffen (siehe Abbildung 3.2).

Das abstrakte *Container*-Element dient als Behälter für Modellelemente. Jedes Modellelement ist genau einem *Container* zugeordnet, wobei Modellelemente von *Container* erben und die *containment*-Referenz verfeinern können um selbst als Behälter

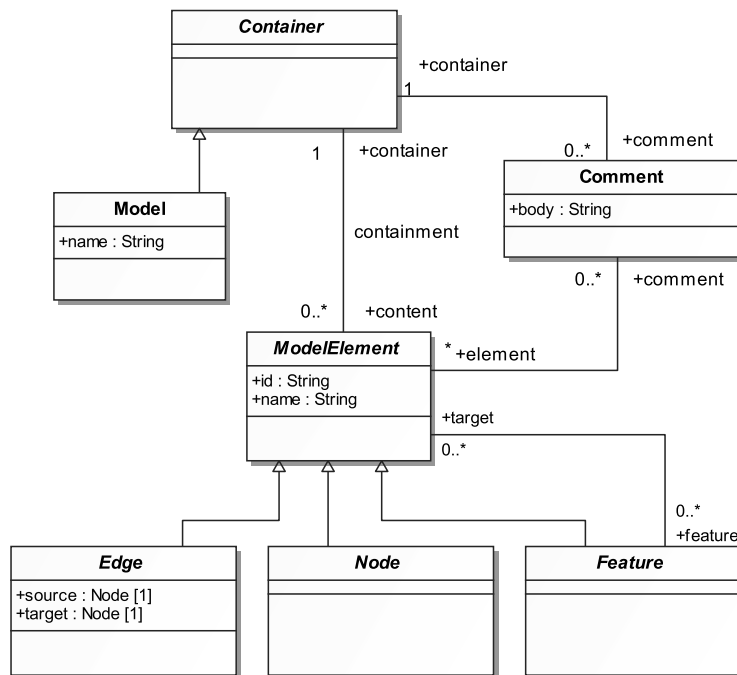


Abbildung 3.2.: MINT-XL Struktur-Paket

für andere Elemente zu dienen. Das bedeutendste *Container*-Element ist das Modell selber. Modellelemente (*ModelElement*) sind unterteilt in Knoten (*Node*) und Kanten (*Edge*), wobei jede Kante gerichtet ist und einen Ausgangs- und Zielknoten besitzt. Diese Beziehung spannt die grundlegende Graphstruktur der DSL auf. Das ebenfalls von *ModelElement* ererbende Element *Feature* bildet die Grundlage für einen Erweiterungsmechanismus, der im Detail in Abschnitt 3.3 erläutert wird. Schließlich bietet das Element *Comment* die Möglichkeit, alle Elemente mit natürlichsprachlichen Kommentaren zu versehen.

Das *Process Oriented Integration*-Paket (siehe Abbildung 3.3) importiert das *Structure*-Paket und definiert Elemente für (Geschäfts-) Prozessmodelle. Diese bestehen aus *ProcessNode*-Elementen, die untereinander mit gerichteten *Flow*-Kanten verbunden sind. Die Reihenfolge der Prozesselemente über die gerichteten Kanten eines Modells ergibt die Ausführungsreihenfolge der Knoten in einem Prozess. *Start* und *End*-Elemente markieren Anfang und Ende des Gesamtprozesses.

*Action*-Elemente sind die zentralen Knoten eines Prozesses. Jeder *Action*-Knoten beschreibt eine Aktivität die während der Ausführung des Prozesses ausgeführt wird. Assoziierte *Actor*-Elemente beschreiben die ausführenden Akteure einer Aktivität. Ein *Action*-Knoten wird ausgeführt (wie alle *ProcessNode*-Knoten im Allgemeinen) wenn alle vorhergehenden Prozesspfade beendet wurden. Dieses Verhalten kann durch die Weitergabe von Marken entlang des Prozesses repräsentiert werden. Jeder Knoten wird ausgeführt wenn alle eingehenden Kanten mit Marken belegt sind und reicht eine

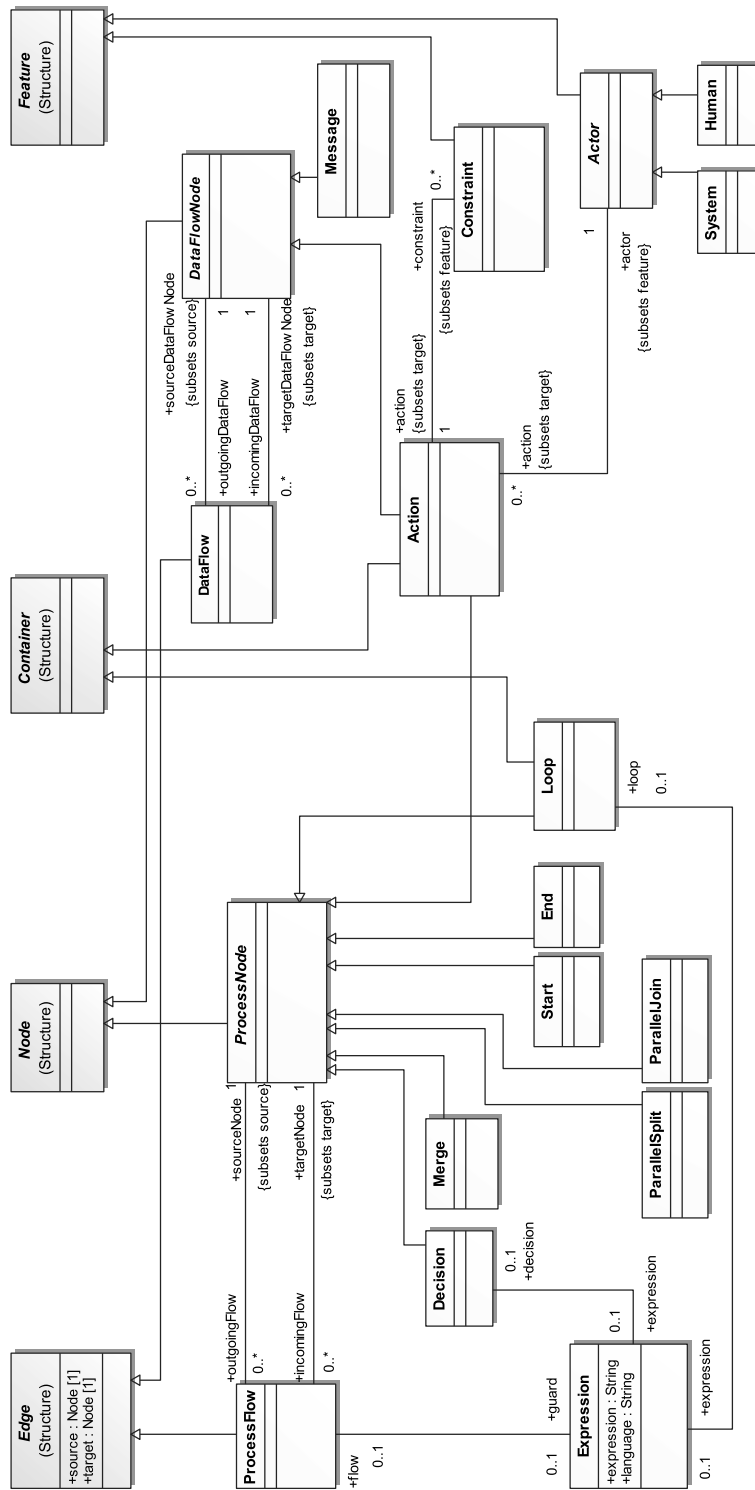


Abbildung 3.3.: *Process Oriented Integration Package*

oder mehrere Marken je nach dem eigenen Verhalten an ausgehende Kanten weiter, wenn die enthaltene Aktivität beendet ist. Die Ausführung jedes *Action*-Knotens kann zusätzlich durch ein *Constraint*-Element beeinflusst werden. So kann zum Beispiel eine Ableitung von *Action* mit einer zeitlichen Einschränkung versehen werden, die den Abbruch der Aktivität nach einer gewissen Zeitspanne auslöst.

Die Abarbeitungsreihenfolge eines Prozesses wird weiter von Knoten beeinflusst, die Verzweigungen oder parallele Bearbeitung einleiten bzw. beenden. *Decision*-Knoten wählen einen von mehreren nachfolgenden Pfaden zur Bearbeitung aus, abhängig von assoziierten Ausdrücken (*Expression*), die ausgewertet werden (XOR-Verhalten). *Decision*-Verzweigungen werden durch *Merge*-Knoten wieder zusammengeführt.

Parallele Verarbeitung wird durch die Knoten *ParallelSplit* und *ParallelJoin* ausgedrückt. Dabei hängt die Art der Parallelität von der Implementierung der zu Grunde liegenden Plattform ab, da diese parallele Verarbeitung durch Mehrprozessormaschinen unterstützen kann oder nicht. Im Prozesskontext wird jedoch angenommen, dass die einzelnen Pfade erst mit einem *ParallelJoin*-Knoten wieder synchronisiert werden.

Schließlich enthalten Prozessmodelle *Message*-Elemente, die den Informationsfluss zwischen *Action*-Knoten darstellen. Wenn eine *DataFlow*-Kante von einem *Action*-Knoten zu einem *Message*-Knoten führt, produziert dieser *Action*-Knoten die gegebene Nachricht. Führt eine *DataFlow*-Kante aus einem *Message*-Knoten zu einem *Action*-Knoten, dient die gegebene Nachricht der Aktivität als Eingabe.

### 3.3. Erweiterungsmechanismen

Wie in Abschnitt 3.1 beschrieben, wird im MINT-Projekt zwischen Prozessmodellierung als technische Domäne und zusätzlichen Fachdomänen unterschieden. Diese Unterscheidung ist im Design der MINT-XL reflektiert. Der Kern der MINT-XL besteht aus Elementen zur Prozessmodellierung. Dazu wurde eine konkrete Syntax zur Verwendung von Fachexperten entwickelt, zusammen mit einem entsprechenden Editor und Transformationen zu unterschiedlichen Architekturmodellen (siehe Abschnitt 5.2).

Die MINT-XL definiert Erweiterungspunkte, um die Sprache auf unterschiedliche (Fach-)Domänen und Entwicklungsprozesse anzupassen. Abbildung 3.4 zeigt eine schematische Darstellung dieses Erweiterungsmechanismus.

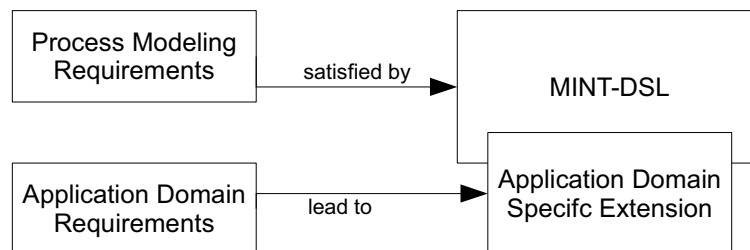


Abbildung 3.4.: Erweiterungsmechanismus der MINT-XL

Die MINT-XL bietet zwei Erweiterungsmechanismen. Der erste ist implizit, durch

die Verwendung des CMOF-Metamodells, gegeben. Zur Erstellung von Erweiterungen können die durch die CMOF bereitgestellten Mechanismen zum Importieren, Kombinieren und Vereinen von (MINT-XL) Paketen genutzt werden (siehe [45]). Dabei können domänenspezifische Elemente existierende Elemente erweitern, neue Attribute einführen oder bestehende undefinieren oder von diesen erben. Dieser Mechanismus ist mächtig, da alle Möglichkeiten zur Erstellung und Anpassung einer Sprache mittels CMOF erhalten bleiben. Jedoch gehen solche Änderungen auf Kosten der einfachen Verwendung bestehender Werkzeuge und Prozesse, da diese auf neue bzw. veränderte Elemente und Konstrukte angepasst werden müssen. Eine breite Anpassung der MINT-XL auf weitere Domänen kann so eine aufwendige Anpassung der zugehörigen Artefakte nach sich ziehen.

Der zweite, explizite Erweiterungsmechanismus erlaubt es, Verfeinerungen der MINT-XL mittels Annotationen ohne größeren zusätzlichen Aufwand durchzuführen. Dieser ist durch das *Feature*-Element im *Structure*-Paket gegeben (siehe Abbildung 3.2). Im Allgemeinen können alle Modellelemente mit einer beliebigen Anzahl von *Feature*-Subklassen annotiert werden. Diese reichern Modellelemente mit zusätzlicher (domänenspezifischer) Information an. Zum Beispiel wird der *Feature*-Mechanismus im *Process Oriented Integration*-Paket benutzt um die Möglichkeit zu schaffen, Aktivitäten (*Action*) mit Akteuren (*Actor*) zu versehen, die die gegebene Aktivität ausführen. Das *Actor*-Element erweitert *Feature* und die Assoziationen *source* und *target* von *Feature*.

### 3.3.1. Allgemeine Erweiterung: Mappings

Die Mappings im Rahmen der MINT-XL dienen der konzeptuellen Modellierung der Abbildung von Daten. Dabei steht auf der betrachteten fachlichen Ebene zunächst nur die Semantik der Daten im Vordergrund. Technische Details wie Datentypen oder Zeichensatzkodierungen spielen auf dieser Ebene keine Rolle. Eine konzeptuelle Abbildung von Daten wird vor allem in folgenden Fällen notwendig:

- Verwendung verschiedener Namen für das gleiche Konzept in unterschiedlichen Systemen (Synonyme).
- Verwendung gleicher Namen für verschiedene Konzepte in unterschiedlichen Systemen (Homonyme).
- Verschiedene Strukturen des gleichen Konzepts (z. B. hervorgerufen durch verschiedene Sichten auf das gleiche Konzept). Ein Beispiel wäre ein Konzept *Kunde*, das in verschiedenen Systemen verwendet wird, aber in den Systemen über verschiedene Attribute verfügt.
- Benennung von Konzepten in verschiedenen natürlichen Sprachen (z. B. Deutsch und Englisch) in unterschiedlichen Systemen.

Im Rahmen der Prozessmodellierung steht die Integration der Schnittstellendaten zweier kommunizierender Systeme im Vordergrund. Dabei werden z. B. Ausgabeparameter von Services auf Eingabeparameter später aufgerufener Services abgebildet.

Abbildung 3.5 zeigt das MINT-XL-Paket in dem das Metamodell der Mappings beschrieben wird.

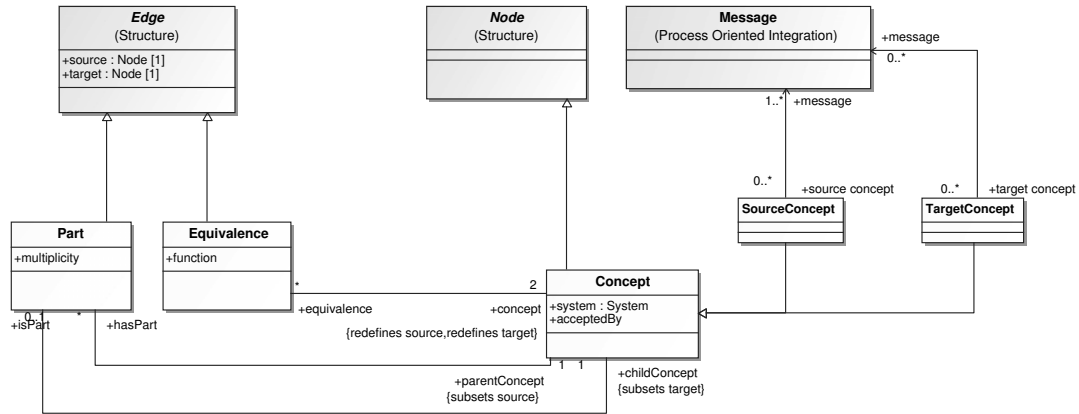


Abbildung 3.5.: Mappings im MINT-XL Metamodell

Im Folgenden wird das Metamodell am Beispiel der Prozessintegration beschrieben. Die Kommunikation zweier Systeme wird in der Prozesssicht der MINT-XL mit Nachrichten modelliert. Diese Nachrichten lassen sich durch die Elemente im gezeigten MINT-XL-Paket mit Quell- und Zielkonzepten annotieren. Die Konzepte entsprechen in der Praxis den Parametern der Schnittstellen der beteiligten Systeme. Dabei stellt *SourceConcept* eine konzeptuelle Sicht auf Daten auf Seite des aufrufenden Systems und *TargetConcept* eine konzeptuelle Sicht auf Daten auf Seite des aufgerufenen Systems dar.

Zwischen den Konzepten auf Quell- oder Zielseite lassen sich mit Hilfe der *Part*-Beziehung Konzepte verfeinern und somit eine Konzepthierarchie aufbauen. Auf die Praxis bezogen heißt das, dass auch strukturierte Datentypen modelliert werden können. Z. B. ließe sich auf diese Weise, wie in Abbildung 3.6 zu sehen, ein strukturierter Datentyp *Maildaten* modellieren, der die Attribute *Mailadresse*, *Betreff* und *Inhalt* hat. Zum Mapping zwischen Konzepten auf der Quellseite und Konzepten auf der Zielseite lässt sich die *Equivalence*-Beziehung verwenden. Das entspricht z. B. der Beziehung zwischen den Konzepten *E-Mail* und *Mailadresse* in Abbildung 3.6.

### 3.3.2. Domänenspezifische Erweiterung: Störungsmanagement

Die Domäne *Störungsmanagement* (engl. *Fault Support*) umfasst Elemente für die Modellierung von Prozessen für das Störungsmanagement und die Systemintegration bei Versorgern (Energie, Gas, Wasser). Diese sind in dem *Fault Support*-Paket zusammengefasst (siehe Abbildung 3.7). Abschnitt 3.4 beschreibt beispielhaft einen Prozess aus dem Störungsmanagement in dem die hier beschriebenen Spracherweiterungen genutzt werden.

Das *Fault Support*-Paket importiert die Pakete *Structure* und *Process Oriented Integration* (siehe Abschnitt 3.2) und bietet folgende domänenspezifische Subklassen von



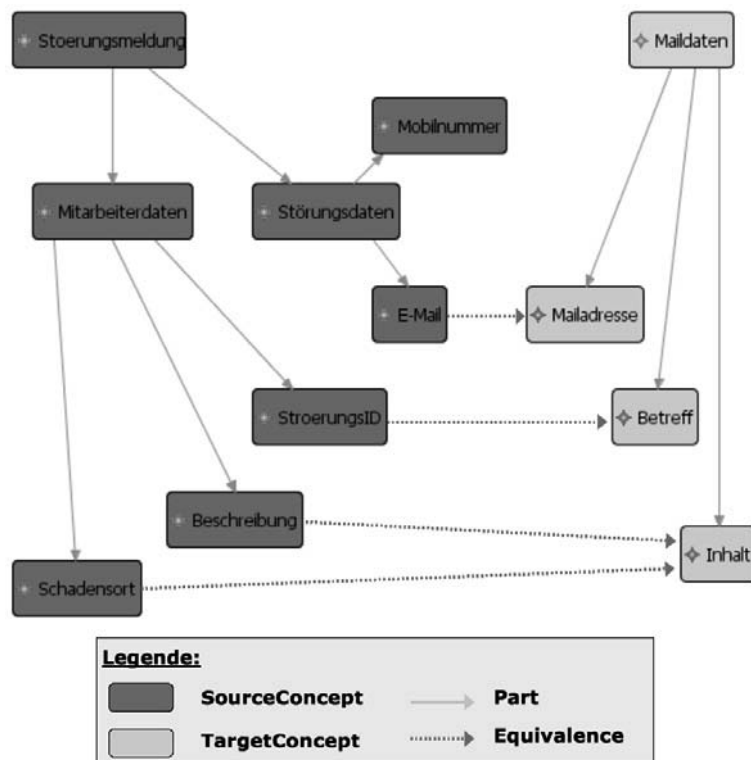


Abbildung 3.6.: Beispiel für ein Mapping in der MINT-XL

*Action:*

- *PhoneCall* stellt einen Telefonanruf dar, und führt das Attribut *phoneCallID* ein
- *EmergencyCall* erweitert *PhoneCall* und repräsentiert einen Notruf
- *Shutdown* ist die Abschaltung einer Anlage
- *Emergency Shutdown* ist eine Abschaltung im Notfall als Erweiterung von *Shutdown*
- *Deposit* stellt das Ablegen von Informationen dar
- *Interaction* ist die Interaktion zwischen zwei Entitäten
- *HumanSystemInteraction* ist die Interaktion ausgehend von einem Menschen mit einem System (menschliche Eingaben in ein System)
- *SystemHumanInteraction* ist die Interaktion von einem System zu einem Menschen (Ausgaben eines Systems für menschliche Benutzer)
- *Notification* stellt eine Benachrichtigung dar

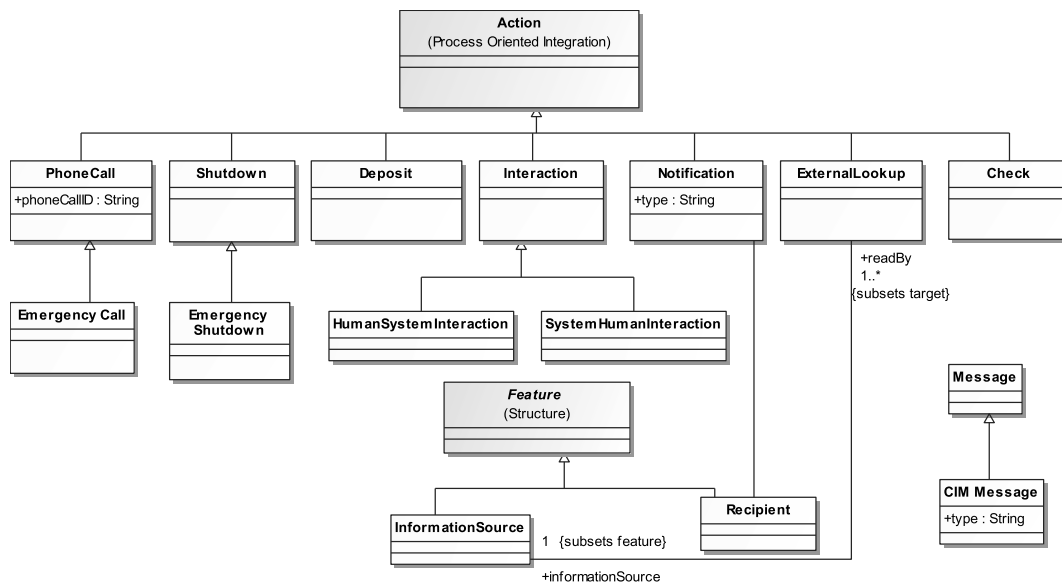


Abbildung 3.7.: Das *Fault Support*-Paket

- *ExternalLookup* ist ein externer Aufruf von Informationen
- *Check* ist eine Prüfung eines Sachverhalts

Zusätzlich sind die *Features*

- *InformationSource* als Quelle zu einem externen Aufruf (*ExternalLookup*), und
- *Recipient* als Empfänger einer Benachrichtigung (*Notification*)

und als Erweiterung von *Message* die Nachricht

- *CIM Message*, die eine Nachricht nach dem Common Information Model (CIM) darstellt,

definiert.

### 3.4. Anwendungsbeispiel MINT-XL

Als Anwendungsbeispiel der MINT-XL dient ein fachlicher Prozess, der den in Abschnitt 2.7.1 beschriebenen Anwendungsfall abbildet (siehe Abbildungen 3.8, 3.9 und 3.10). Für die Modellierung wurden die Basiselemente der MINT-XL um die domänenspezifische Erweiterung *Störungsmanagement* (3.3.2) erweitert.

Der Prozess fasst das Vorgehen eines Mitarbeiters in einem Callcenter eines Energieversorgers (Strom, Gas, Wasser) zur Bearbeitung einer telefonischen Störungsmeldung. Der Prozess beginnt mit einem eingehenden Anruf eines Kunden, bei dem eine Störung

einer Dienstleistung des Versorgers aufgetreten ist. Der Callcenter-Mitarbeiter erfragt die wichtigsten Informationen zur Störung: die Sparte, die Daten des Meldenden und den Ort an dem die Störung aufgetreten ist. Diese Daten werden in das Störungserfassungssystem (SES) eingegeben. Die Schritte zur Datenerfassung unterliegen keiner festen Reihenfolge und sind so im Prozessmodell als parallele Verarbeitung dargestellt.

Im Anschluss an die Dateneingabe prüft das SES ob die Störung im Versorgungsgebiet des Unternehmens liegt. Ist dies nicht der Fall, endet der Prozess im dargestellten Beispiel, um die Darstellung kurz zu halten. In einer vollständigen Darstellung folgen weitere Schritte, um den Kunden an einen geeigneten Ansprechpartner weiterzuleiten.

Ist die Zuständigkeit des Versorgers gegeben, führt das SES eine Benachrichtigung eines geographischen Informationssystems (GIS) durch. Die Nachricht entspricht dem Common Information Model (CIM). Das GIS erstellt eine Karte mit der geographischen Einordnung des Störungsorts und sendet diese an das SES zurück. Dieses stellt die Karte dem Mitarbeiter zusammen mit anderen bereits gemeldeten Störungen dar, damit dieser feststellen kann, ob die aktuelle Störung schon gemeldet wurde oder im Zusammenhang mit bereits gemeldeten Störungen steht. Ist die Störung schon bekannt, endet in der verkürzten Darstellung der Prozess.

Wird die Störung als neu eingeordnet, erfragt der Mitarbeiter alle relevanten Details zur Störung. Diese werden in einer neuen Störungsmeldung im SES angelegt. Nach Eintrag der Störungsdaten werden parallel Nachrichten zur weiteren Bearbeitung an zuständige Außenstellen versandt. Dafür werden zwei prozesseexterne Dienste angesprochen, die jeweils eine für sie passende, detaillierte Störungsmeldung erhalten. Dies sind ein SMS-Dienst, der zuständigen Mitarbeitern der Außenstelle eine SMS schickt und ein E-Mail-Dienst, der Details über die Störung per E-Mail weiterleitet. Nach der Benachrichtigung der Außenstelle endet der Prozess.

## **3.5. Umsetzung**

Dieser Abschnitt beschreibt die Implementierung der MINT-XL und eines entsprechenden Editors mit Hilfe von Werkzeugen aus dem Eclipse-Umfeld.

### **3.5.1. Umsetzung von MINT-XL mit EMF**

Um die im Eclipse-Umfeld bestehenden Generierungswerkzeuge nutzen zu können, wurde die MINT-XL mit Hilfe des Ecore-Metametamodells des Eclipse-Modelling-Frameworks (EMF) beschrieben. Da Ecore eine Implementierung des Essential-MOF-Standards (EMOF) ist, der eine Teilmenge von CMOF darstellt, mussten einige Details leicht verändert werden. Bei der Umsetzung der MINT-XL (s. Abschnitt 3.2) mit Ecore wurde die Einteilung in Pakete beibehalten. Auch sonst entsprechen die Klassen der EMOF- und der CMOF-Variante einander. In Ecore können Relationen einer erbenden Klasse jedoch nicht, wie in CMOF, Relationen einer Elternklasse redefinieren oder partitionieren. In diesen Fällen wurden in den jeweiligen konkreten Klassen eigene Relationen geschaffen.

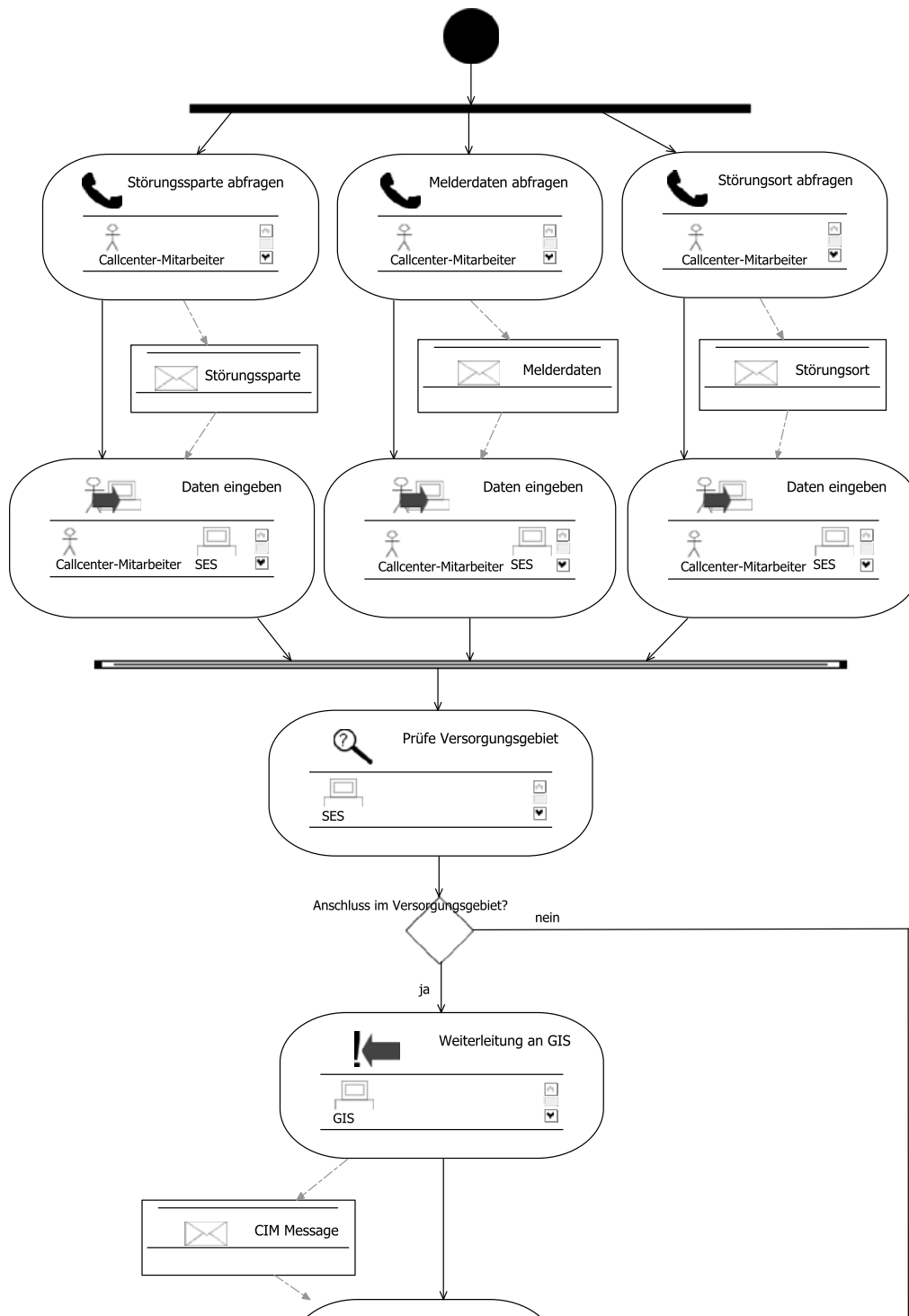


Abbildung 3.8.: Anwendungsbeispiel Störungsmanagement I

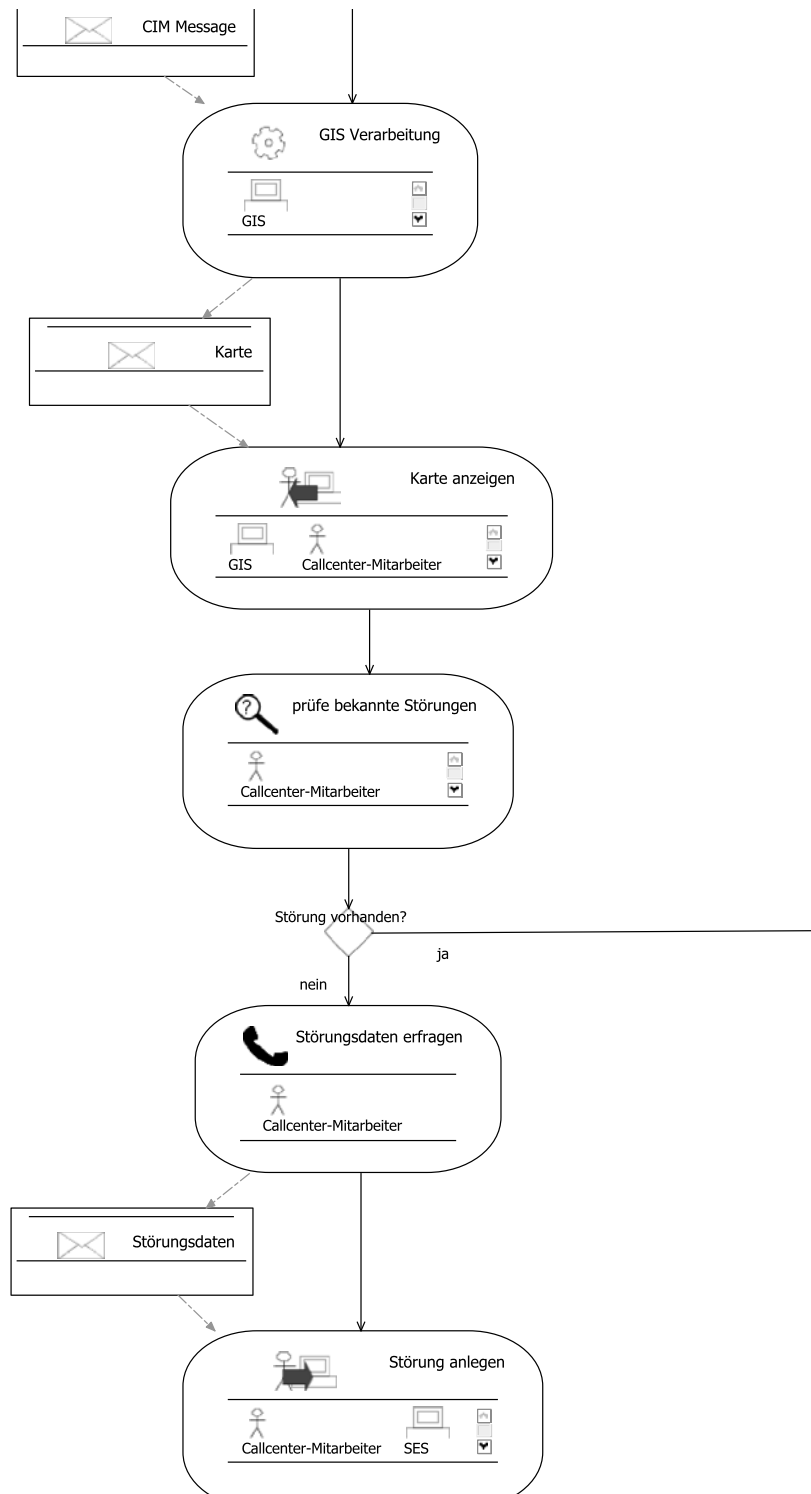


Abbildung 3.9.: Anwendungsbeispiel Störungsmanagement II

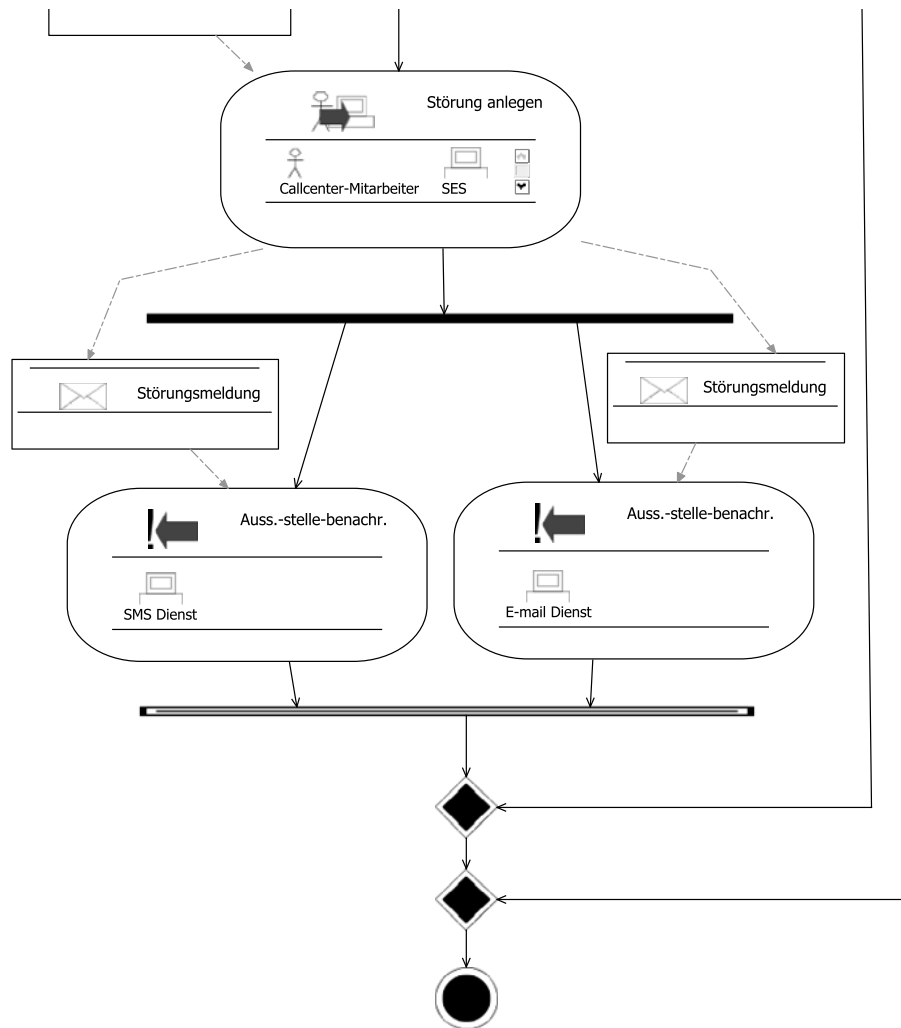


Abbildung 3.10.: Anwendungsbeispiel Störungsmanagement III

### 3.5.2. Erstellung des MINT-PIE-Editors mit GMF

Bei der Erstellung des MINT Process Integration Editors (MINT-PIE) wurde zunächst dem Vorgehensmodell des Graphical Modelling Frameworks (GMF, s. Abschnitt 2.3.2) entsprechend vorgegangen, d. h. es wurde ein Graph-Modell erstellt, welches mittels primitiver Figuren die Elemente darstellt, sowie ein Tooling-Modell, um die Werkzeuge zu definieren. Beide Modelle sind miteinander und mit dem MINT-XL-Metamodell über ein Mapping-Modell verknüpft. Dadurch läßt sich ein GenModel erstellen, durch welches mit openArchitectureWare (s. Abschnitt 2.4.3) der vollständige Editor in lauffähigem Java-Code generiert wird.

Beschränkt man sich jedoch auf die eigenen Mittel von GMF, ist das Ergebnis für den Einsatz des Editors im produktiven Einsatz und die Anwendung durch Domänenexperten wenig befriedigend. Zum einen sind die primitiven Symbole der Knoten wenig ansprechend, zum anderen sind die Verbindungslinien nicht direkt mit diesen verknüpft, handelt es sich nicht um Rechtecke als Figuren.

Zwar ermöglicht GMF die Erstellung von Custom-Figuren direkt im Editor des Graph-Modells, jedoch ist dies zum einen sehr mühsam und wenig intuitiv, zum anderen sind diese Figuren nicht skalierbar. Es besteht auch die Möglichkeit, Bilder als Figuren zu laden. Dies ist sogar mit Vektorgrafiken möglich. Diese werden jedoch unmittelbar in Bitmap-Grafiken umgewandelt, so dass sie, wenn sie skaliert werden, an Qualität verlieren.

#### Erweiterung von GMF

Abhilfe schafft die Erstellung einer eigenen Figurenklasse `SVGFigure`, welche generisch Vektorgrafiken im Scalable Vector Graphics (SVG) Format lädt und darstellt. Die Klasse kann über den Editor für das Graph-Modell als Figur eingebunden werden, wobei der Pfad durch ein Attribut übergeben wird. Die Definition jeder Vektorgrafik wird dabei nur einmal geladen und in einen Objektbaum geparkt und wird für jede Figur mit derselben Vektorgrafik verwendet. Dies ist sehr effizient und performanter als die Einbindung von Grafiken, wie sie GMF zur Verfügung stellt. Alle SVG-Figuren sind frei skalierbar, sowohl mit als auch ohne festem Seitenverhältnis.

Ein weiterer Vorteil der Verwendung von `SVGFigure` sind generische *Anker*, welche die Figuren mit den Kanten verbinden. Dabei kann es sich um markante Punkte der Figuren selbst handeln, aber auch um Linien oder Kreise. Ein Anker kann direkt in der SVG-Datei definiert werden.

Um solch einen Anker zu nutzen, muss der Quellcode des Editors modifiziert werden. Dies ist jedoch hinderlich, wenn Änderungen am Metamodell oder an den Definitionsmodellen des Editors erfolgen und eine neue Generierung des Quellcodes erforderlich machen. Jedoch kann der Generierungsprozess über die Xpand-Templates (s. Abschnitt 2.4.3) von GMF beeinflusst werden. Ohne die GMF-eigenen Templates ändern zu müssen, können aspektorientierte Zusatztemplates geschrieben werden, so dass die gewünschten Änderungen im Quellcode bereits im Generierungsprozess entstehen.

Mit Templates wurden auch noch eine Reihe anderer Modifikationen vorgenommen.

Zum einen musste die korrekte Einfügung neuer Objekte in die Objekthierarchie des Modells hergestellt werden, zum anderen sollte die Speicherung der Modelldateien so vorgenommen werden, dass das MINT-XL-Metamodell als Datei in der Modelldatei referenziert wird. Auch die Speicherung der Ecore-IDs konnte nur über Templates erreicht werden.

Auf diese Weise konnte auch ein Fehler in der Codegenerierung von GMF behoben werden, der verhinderte, dass mehr als ein Subeditor aus dem Editor gestartet werden konnte.

### **Beschreibung des Editors**

Bei dem fertigen Editor handelt es sich um ein Eclipse Plug-In, welches durch das Erstellen oder das Öffnen einer Datei vom Typ *MINT-XL Diagramm* gestartet wird. Mit der linken Werkzeugliste können Modellelemente der mittigen Zeichenfläche hinzugefügt werden (s. Abbildung 3.11). Alle Modellelemente können frei positioniert und skaliert werden. Ihre Eigenschaften können in einer Property-View editiert werden. Im Kontext der Actions können Subprozesse editiert werden, hierzu wird eine neue Instanz des Editors geöffnet. Über die Messages kann der ConceptMapping-Editor (s. Abschnitt 5.1.1) geöffnet werden. Weiterhin verfügt der Editor über die Möglichkeit, Diagramme als Bilddateien zu exportieren und auszudrucken.

Der Editor muss nicht ausschließlich als Plug-In für Eclipse genutzt werden. Er kann auch wahlweise als Rich-Client Applikation auf Basis der Eclipse Rich-Client Plattform erstellt werden, so dass eine eigenständige Anwendung genutzt werden kann.



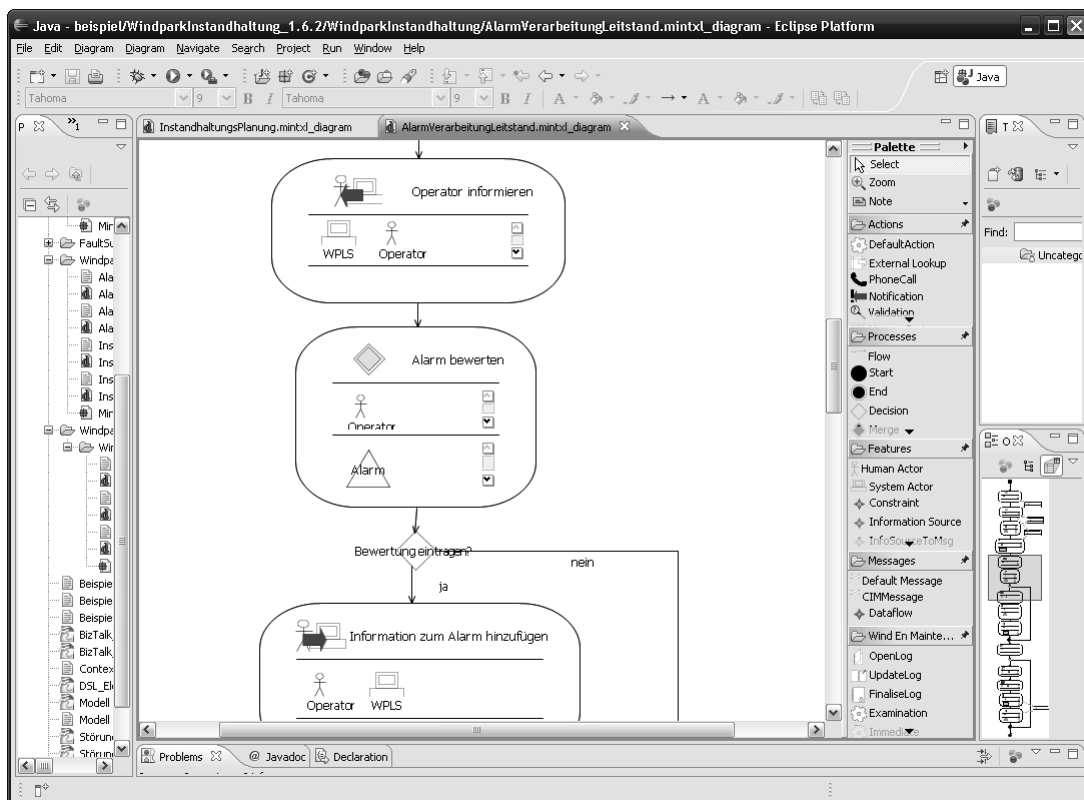


Abbildung 3.11.: MINT-PIE Editor



## 4. Umsetzung wissensintensiver Prozesse

*Malte Zilinski*

Dieses Kapitel geht auf die notwendigen Besonderheiten der Modellierung von wissensintensiven Prozessen im Kontext der *Model Driven Architecture* (MDA, siehe [40]) ein. Es wird ein Verfahren vorgestellt, das domänenspezifisch wissensintensive Prozesse erfasst und diese für einen Softwareentwicklungsprozess nutzbar macht. Für die Strukturierung des Wissens werden dem Domänenexperten kognitive Muster bereitgestellt. Diese werden in der MINT-XL (siehe Kapitel 3) modelliert und für die entsprechenden domänenspezifischen Anforderungen spezialisiert. Es wird gezeigt, dass sie sich als *Computation Independent Models* (CIM) für die Softwareentwicklung nach MDA nutzen lassen. Die Zielplattformen für den MDA-Prozess sind in diesem Kontext Methoden bzw. Sprachen aus dem Bereich der Künstlichen Intelligenz (KI). Der hier vorgestellte Ansatz wird an der Beispieldomäne Studienplanung illustriert, dabei wird das Planungsproblem auf eine ausführbare Planungssprache abgebildet. Der beschriebene Ansatz ist innerhalb der Problemklasse generisch und kann auf verwandte Probleme in anderen Anwendungsdomänen übertragen werden.

Am Anfang dieses Kapitels werden die Verwendung und die Anpassung von bestehenden Mustern für die Modellierung wissensintensiver Prozesse beschrieben. Im Weiteren wird gezeigt, wie diese Muster in die MINT-XL übertragen werden können. Die Muster müssen anschließend verfeinert werden, um den domänenspezifischen Anforderungen zu entsprechen und sie für die weitere Nutzung im MINT-Prozess verwenden zu können. Im Folgenden wird für einen Planungsprozess gezeigt, wie aus dem domänenspezifischen Modell über verschiedene Transformationen ein ablauffähiges System generiert werden kann.

### 4.1. Wissensintensive Prozesse

Die Modellierung von wissensintensiven Prozessen ist durch ihre speziellen Anforderungen anders geartet als klassische Prozessmodellierung. Die einfache Definition von Kontrollstrukturen mit zugehörigen Datenstrukturen, wie man es von Geschäftsprozessen gewohnt ist, lässt sich im Allgemeinen nicht auf die Klasse wissensintensiver Prozesse übertragen. Daher stellt sie sich besonders im Bereich der MDA als nicht trivial dar. Das Kernproblem besteht darin, dass die Geschäftsprozesse teils aufwändige Berechnungsalgorithmen beschreiben müssten, diese aber auf den oberen Modellierungsebenen ohne Bedeutung sind. Abstrahiert man von diesen Algorithmen auf

höheren Ebenen, stellt sich das Problem, das keine Informationen definiert sind, die für die Generierung der weiteren MDA-Ebenen verwendet werden könnten.

CommonKADS (siehe Knowledge Engineering and Management [64]) ist eine Methodologie zur Erfassung, Strukturierung, Formalisierung und Operationalisierung von Wissen. Für die Modellierung wissensintensiver Prozesse stellt sie eine deklarative und für Domänenexperten verständliche Sprache bereit. Es werden vorgefertigte, generische Problemlösemuster, die „Template Knowledge Models“ (TKM), für verschiedene wissensintensive Prozesse angeboten, welche von Experten für die Domänenmodellierung verwendet werden sollen. Wissensintensive Prozesse lassen sich in zwei verschiedene Klassen aufteilen: analytische und synthetische Prozesse. Für die analytischen gilt, dass das System, auf das der Prozess angewendet wird, bereits existiert. Hingegen erstellen synthetische Prozesse ein neues System. Die Eingaben definieren dabei die Bedingungen, die für das zu erstellende System gelten müssen. Die Abbildung 4.1 zeigt verschiedene wissensintensive Prozesse entsprechend der oben vorgestellten Klassifizierung.

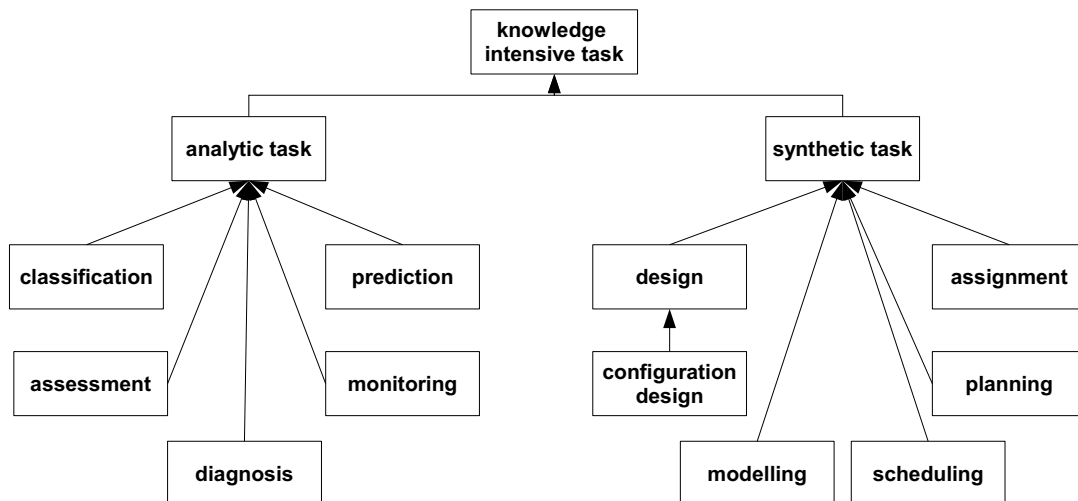


Abbildung 4.1.: Übersicht CommonKADS: wissensintensive Prozesse [64, S. 125]

## 4.2. MINT Vorgehensmodell

Die Betrachtung wissensintensiver Prozesse im Kontext der MDA ist interessant, da sie in vielen Anwendungen auftreten, ihre Lösung aber nicht trivial ist. Im Bereich der KI existieren spezialisierte Sprachen, die sich für die Beschreibung und Lösung solcher Probleme bewährt haben, deren Anwendung bisher aber Expertenwissen erforderte. Das im Folgenden beschriebene MINT-Vorgehensmodell ermöglicht es Domänenexperten, Planungsprobleme fachlich zu beschreiben und diese für die Softwareentwicklung von spezialisierten planungsspezifischen Details zu abstrahieren.

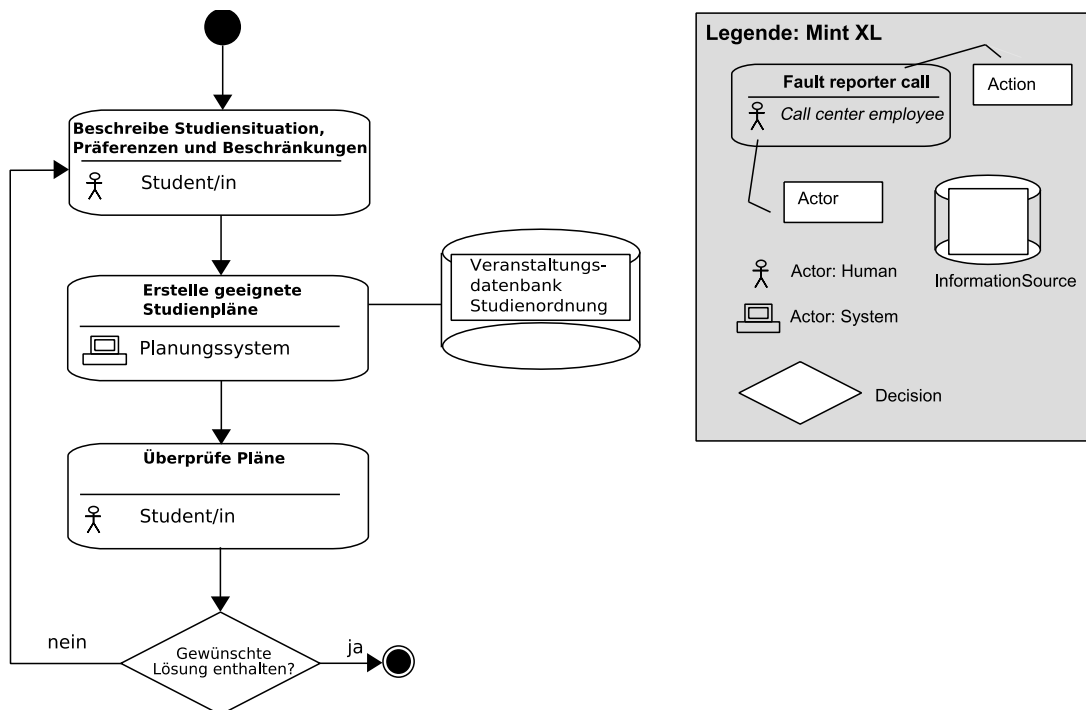


Abbildung 4.2.: Geschäftsprozess der Studienplanung [34]

Im Rahmen des MINT-Projekts wurde dieses Vorgehensmodell auf die Domäne der Studienplanung angewendet. Dabei wurde ein Softwaresystem entwickelt, welches Studierende entsprechend ihrer Studiensituation bei der Studienplanung unterstützen kann. Abbildung 4.2 zeigt diesen Prozess in der Syntax der MINT-XL.

Der planungsspezifische Prozess wird auf CIM-Ebene mit einem adaptierten und instanziierten TKM Planning für Planungsprozesse in der MINT-XL dargestellt (Abbildung 4.3). Für die Definition der fachlichen Details werden die Elemente des Musters mit klassischen MDA-Methoden verfeinert und entsprechend im MDA-Prozess verarbeitet. Es zeigt sich, dass eine Abbildung des Studienplanungsproblems auf ein Scheduling-Problem möglich ist. Auf PIM-Ebene wird deshalb ein generisches Scheduling-Metamodell bereitgestellt, welches die fachlichen Anforderungen des CIM fasst und um Scheduling-Details erweitert. Das Studienplanungsproblem wird als partielle Instanz des Scheduling-Metamodells beschrieben. Auf plattformspezifischer Ebene wird die Planning Domain Definition Language (PDDL) (siehe [37, 18]) als Repräsentationsform verwendet - somit ist das Modell deklarativ und maschinell lösbar.

### 4.3. Metamodellerzeugung

Bei der Modellierung von wissensintensiven Prozessen auf MDA-Basis müssen für die verschiedenen Abstraktionsebenen (CIM, PIM, PSM) geeignete Metamodelle bereitge-

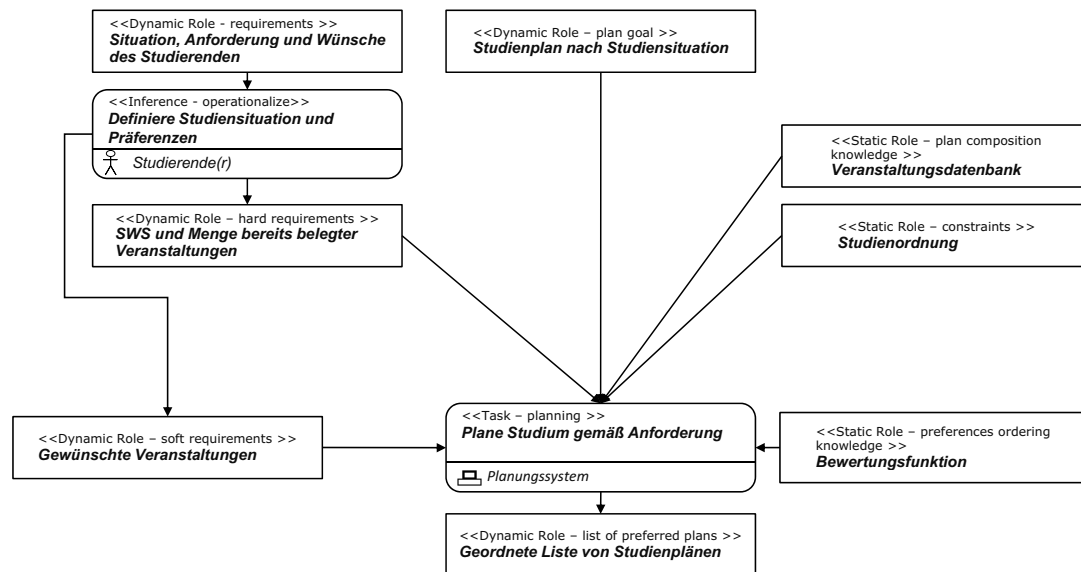


Abbildung 4.3.: Instanziierung des abstrahierten TKM Planning in MINT-XL [34]

stellt werden, welche die jeweiligen Anforderungen repräsentieren können. Da aber zu Beginn des Softwareentwicklungsprozesses gegebenenfalls noch nicht vollständig spezifiziert ist, wie diese Anforderungen aussehen, wird ein iterativer, zweistufiger Prozess vorgeschlagen, der die domänenspezifischen Anforderungen zielgerichtet analysiert und diese entsprechend an die weiteren MDA-Ebenen überträgt. Auf der anderen Seite werden Softwareprototypen entwickelt (PSM), die die Teile der Anforderungen implementieren und damit wiederum aus anderer Perspektive Anforderungen identifizieren, die Richtung PIM und CIM übertragen werden. Dieser Prozess wird solange wiederholt, bis die für das Softwaresystem gewünschten Domänenanforderungen entsprechend repräsentiert werden können. Abbildung 4.4 zeigt eine Visualisierung des vorgeschlagenen Prozesses. Im Folgenden werden die im MINT-Projekt erzeugten Modelle vorgestellt.

#### 4.4. CIM

Das adaptierte instanziierte TKM Planning (Abbildung 4.3) verfeinert den Geschäftsprozess zur Studienplanung (Abbildung 4.2) um planungsspezifische Aspekte. CommonKADS definiert für alle Elemente des Musters Rollen, an denen erkennbar ist, welche Funktion die Daten bzw. Aktivitäten im Prozess übernehmen. Bei Daten wird zwischen dynamischen und statischen Rollen unterschieden: erstere bezeichnen Daten mit geringer und letztere mit längerer Lebensdauer. In der Anwendungsdomäne der Studienplanung hat die Studienordnung mit einer Lebensdauer von mehreren Jahren die längste Gültigkeit. Für die Definition von Planungsproblemen wird im MINT-Vorgehensmodell die Domänenmodellierung über die Strukturierung der Domänenobjekte hinaus auf die Modellierung der statischen Struktur des Planungsproblems erweitert.

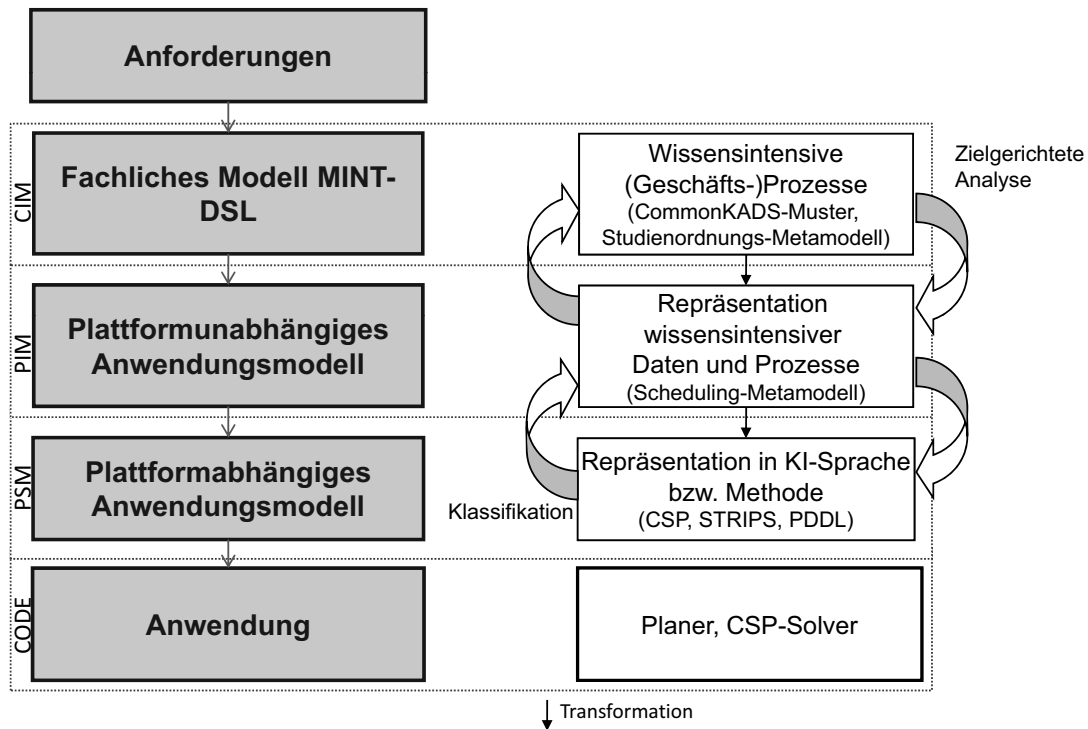


Abbildung 4.4.: Metamodellerzeugung

Für die Domäne Studienplanung wird dafür das Studienordnungsmetamodell zur Verfügung gestellt.

Das Studienordnungsmetamodell erlaubt die domänenspezifische Modellierung von studienordnungsspezifischen Eigenschaften. Im Folgenden werden die Elemente des Metamodells beschrieben.

- **Studienordnung:**
  - **erforderlicheECTS:** Anzahl der erforderlichen ECTS-Punkte
  - **name:** Bezeichner der Studienordnung
  - **modulSlots:** siehe unten
  - **vorbedingungen:** siehe unten
- **ModulSlot** (Modulslots, abstrahieren von einer konkreten Veranstaltung. Für die Planung ist später ein Mapping zwischen Modulslot und tatsächlichem Modul notwendig):
  - **ECTS:** Anzahl der ECTS-Punkte, die für einen Modulslot notwendig sind
  - **laufzeit:** Definiert die Laufzeit eines Modulslots
  - **sws:** Anzahl der Semesterwochenstunden für einen Modulslot

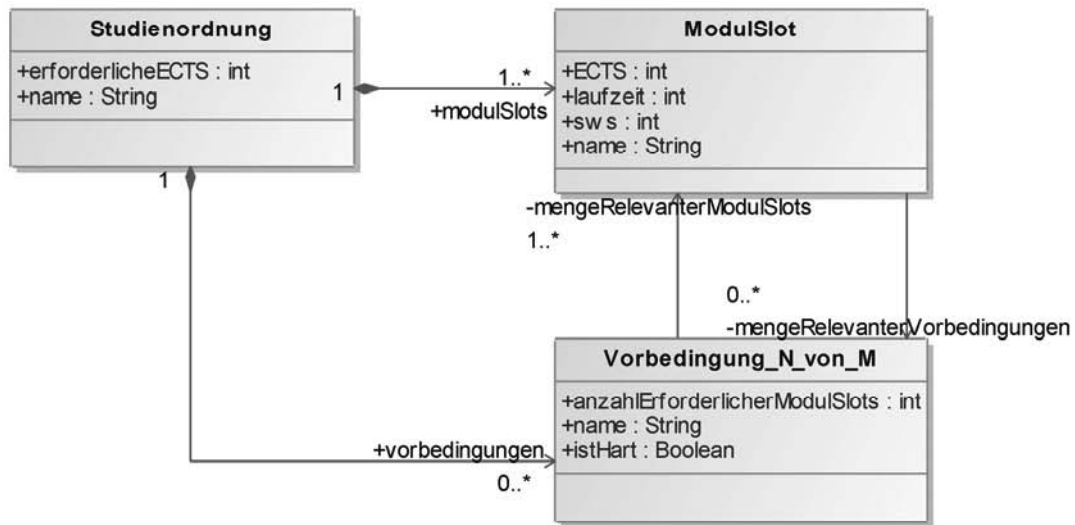


Abbildung 4.5.: Metamodell zur Modellierung der statischen Aspekte der Studienordnung

- **name**: Bezeichner eines Modulslots
- **mengeRelevanterVorbedingungen**: Menge der Vorbedingungen, die erfüllt sein müssen/sollen, damit dieser Modulslot eingeplant werden kann.
- **Vorbedingung\_N\_von\_M** (Definiert Vorbedingungen, die für einzelne Modulslots gelten):
  - **anzahlErforderlicherModulSlots**: Mindestanzahl der bestandenen Modulslots, damit diese Vorbedingung erfüllt ist
  - **mengeRelevanterModulSlots**: Menge/Teilmenge der Modulslots, die für diese Vorbedingung bestanden sein müssen
  - **name**: Name der Vorbedingung
  - **istHart**: Wahr, falls es sich um eine harte Vorbedingung handelt, nein bei Empfehlungen

## 4.5. PIM

Das Problem der Studienplanung lässt sich auf ein nicht-präemptives Scheduling-Problem mit harten und weichen Beschränkungen abbilden. Für die plattformunabhängige Beschreibung von Scheduling-Problemen wurde ein generisches Metamodell entwickelt (siehe Abbildung 4.6), das sich an einer Scheduling-Ontologie orientiert [50]. Dieses erlaubt die Abbildung der Domänenobjekte auf Jobs, Aktivitäten, Vorbedingungen und anderen Scheduling-spezifischen Eigenschaften, wie Fälligkeiten, Kapazitäten usw. Die Überführung der Elemente des Studienordnungsmodells (CIM) erfolgt



über eine Annotation der vorliegenden Elemente mittels eines Scheduling-Profiles. Die automatisierte Transformation erfolgt über ein erstelltes Template.

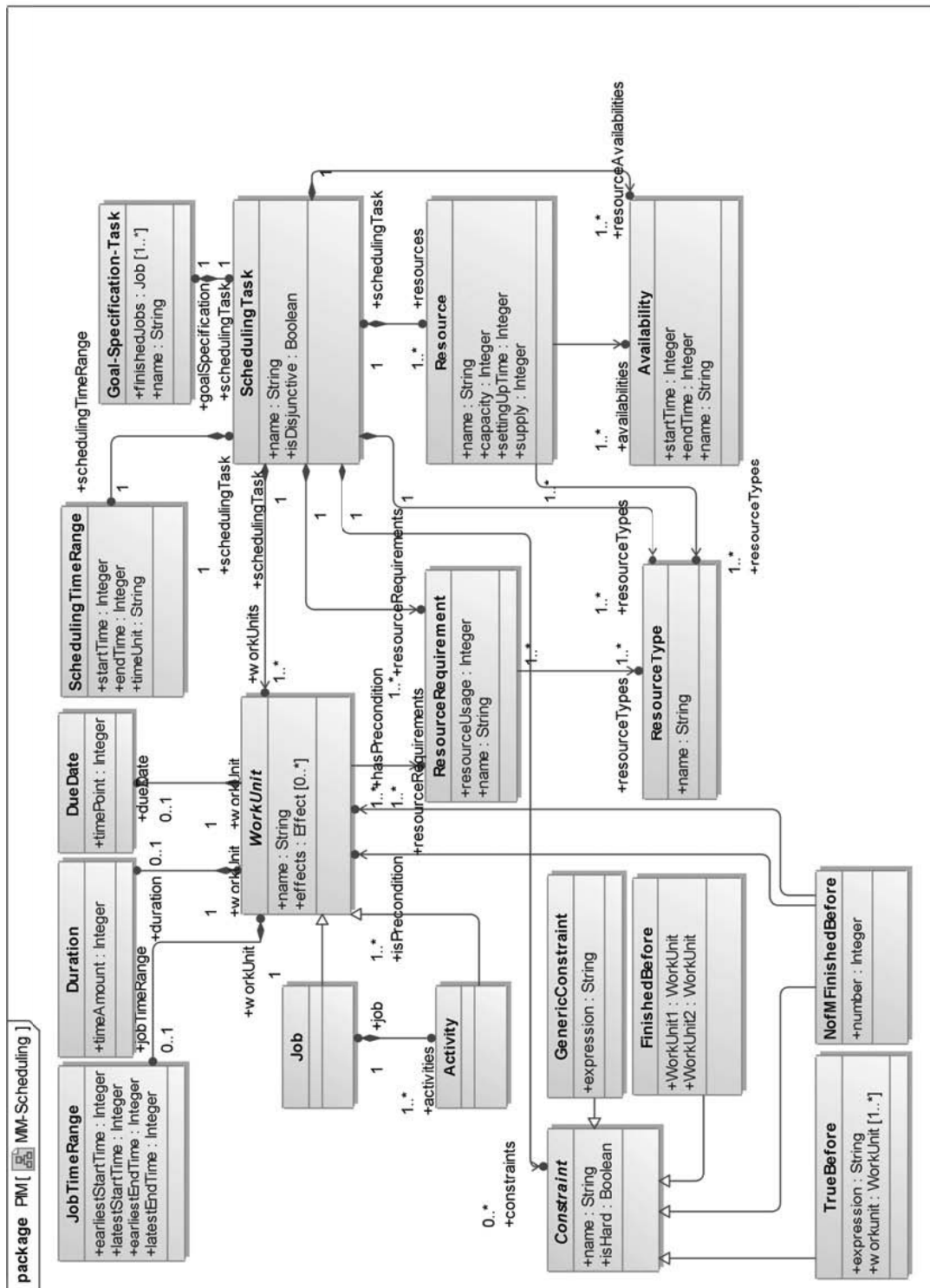


Abbildung 4.6.: Scheduling Metamodell

Die in der Studienordnung definierten Modulslots entsprechen Aktivitäten - mit Eigenschaften wie Dauer und Ressourcenverbrauch. Alle Modellinformationen, die zur Entwicklungszeit für eine Studienordnung definiert wurden, werden zur Erzeugung des Scheduling-Modells verwendet. Daten mit kürzerer Lebensdauer beziehungsweise Daten, die zur Entwicklungszeit des Systems noch nicht vorliegen, aber für die Planung relevant sind, wie zum Beispiel das aktuelle Vorlesungsverzeichnis und eine konkrete Anfrage eines Studierenden, werden erst zur Laufzeit im Scheduling-Modell instantiiert. Vorlesungen, Zeitbeschränkungen etc. werden beispielsweise als Ressourcen mit dazugehörigen Verfügbarkeiten repräsentiert.

Die partielle Instanz des Scheduling-Metamodells beschreibt somit die berechnungsabhängigen, planungsrelevanten Aspekte der Domäne. Zur Laufzeit wird das Scheduling-Modell mit den Daten des Studierenden, den Vorlesungsdaten und dem aktuellen Mapping von Modulen auf Modulslots vervollständigt. Zur Lösung des Scheduling-Problems müssen dem Planer alle Daten zu Verfügung gestellt werden. Dies erfolgt über den Einsatz eines Generators zur Laufzeit. Das Scheduling-Modell fügt sich zu anderen Modellen, die nach gewöhnlichen MDA-Methoden entwickelt werden und zusammen alle Komponenten des Softwaresystems abbilden.

## 4.6. PSM

Für die automatisierte Lösung des Scheduling-Problems wird der PDDL (Planning Domain Definition Language)-Planer `sgplan`<sup>1</sup> eingesetzt. Als Eingabe benötigt der Planer die Daten der Studienordnung, eine Liste der planbaren Veranstaltungen, das Mapping zwischen Veranstaltungen und Modulslots sowie die Anforderungen des Studierenden (Studiensituation, gewünschte Veranstaltungen, usw.). Diese müssen als PDDL-Domänenbeschreibung und als PDDL-Problembeschreibung zur Laufzeit erzeugt werden.

PDDL ist eine generische Planersprache, die zur Definition von Planungsproblemen verwendet wird. Sie ist von verschiedenen generischen Off-the-shelf-Planern lesbar und somit sind die in ihr formulierten Probleme auch maschinell lösbar. Die Domänenbeschreibung definiert unter anderem die Struktur der Planungsobjekte, sowie die für die Erstellung eines Plans möglichen Aktionen. In der Problembeschreibung werden die Ziele, die konkreten Objekte sowie ihre Relationen definiert. Zur Laufzeit wird dem Planer die Domänen- und die Problembeschreibung übergeben. Dieser berechnet einen möglichen Plan, der dann wiederum von einem Parser gelesen und schließlich an das Softwaresystem zurückgegeben wird.

Es sind durchaus alternative KI-Techniken, wie zum Beispiel Problemlösung über *Constraint Satisfaction Problems* (CSPs), zur automatisierten Planung denkbar. Die Art des Planungsproblems beeinflusst dabei die Wahl der entsprechenden Zielplattform.

---

<sup>1</sup><http://manip.crhc.uiuc.edu/programs/SGPlan/index.html>

## 4.7. Zielsystem

Abbildung 4.7 zeigt eine Visualisierung des Zielsystems, welches im Rahmen von MINT zur Lösung von Planungsproblemen im Bereich der Studienberatung entstanden ist. Die Akquisition der dynamischen Daten des Planungsproblems (Studiensituation, Menge der gewünschten Veranstaltungen, etc.) wird mittels einer manuell erzeugten Web-Anwendung durchgeführt. In einem Dialog wird dem Studierenden die Möglichkeit gegeben, seine Studiensituation und seine Wünsche zu spezifizieren. Dies umfasst beispielsweise die Auswahl von gewünschten Veranstaltungen. Hierfür wird die Liste der aktuellen Veranstaltungen aus einer Datenbank gelesen und mittels der Mapping-Informationen auf Relevanz für die entsprechende Studienordnung geprüft. Nach der Akquisition der dynamischen Daten, wird das Scheduling-Modell entsprechend vervollständigt und mittels eines Generators in eine PDDL-Beschreibung transformiert und dem Planer übergeben. Der durch den Planer erstellte Plan wird von einem Parser gelesen und der Web-Applikation übergeben, die dem Studierenden diesen entsprechend anzeigt (siehe Abbildung 4.8).

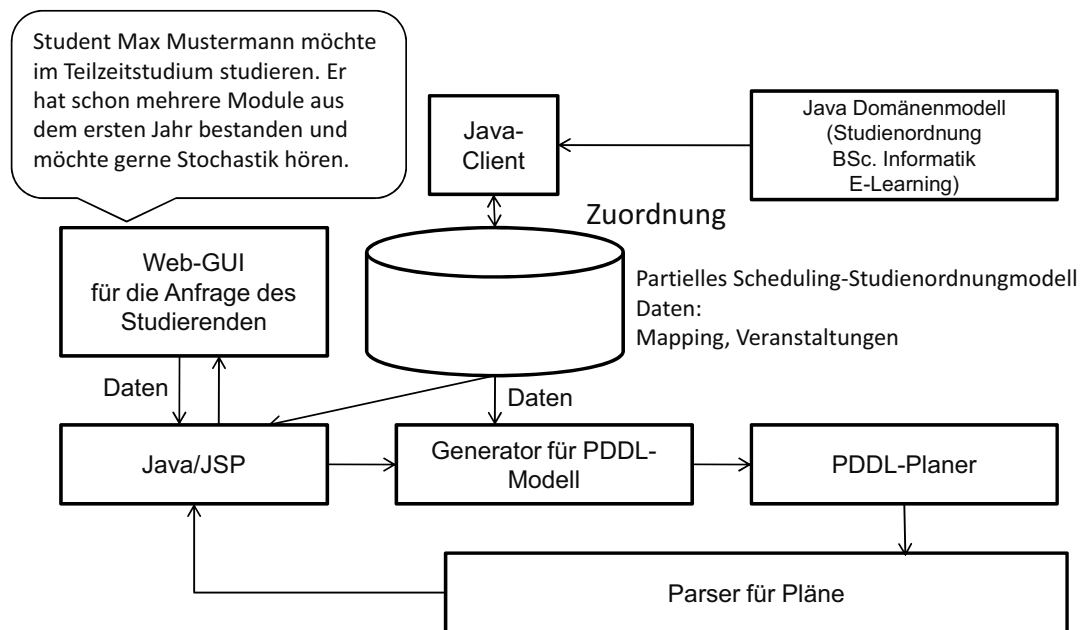


Abbildung 4.7.: Zielsystem

## 4.8. Zusammenfassung

In diesem Kapitel wurde ein Vorgehensmodell zur Modellierung von wissensintensiven Prozessen im Bereich der MDA vorgestellt. Dafür sind ein Studienordnungsmetamodell und ein Scheduling-Metamodell mit entsprechenden Transformationen entwickelt



Abbildung 4.8.: Web-Client Studienplanung

worden. Eine Besonderheit für die Lösung des Planungsproblems und somit eine Anforderung für das Zielsystem ist, dass sich das Scheduling-Modell erst zur Laufzeit vervollständigen lässt und aufgrund dieser Tatsache dann zu einer PDDL-Spezifikation transformiert wird.

Im Bereich der Studienplanung wurde exemplarisch gezeigt, wie man vom Computation Independent Model bis zum ablauffähigen System gelangt. Für die Studienplanungsdomäne wurde als Platform Independent Model ein generisches Scheduling-Metamodell bereitgestellt, in welchem sich Anforderungen der Domäne repräsentieren lassen. Durch eine Transformation eines Scheduling-Modells in eine geeignete Methode bzw. Sprache der Künstlichen Intelligenz kann ein solches Problem automatisiert gelöst werden.



# 5. Modellgetriebene Prozessintegration

*Xinghai Chi, Heiner Feislachen, Michael Gründler, Niels Streekmann*

Dieses Kapitel beschreibt das in MINT entwickelte Vorgehensmodell zur modellgetriebenen Prozessintegration. Das Modell beinhaltet die Stufen der Entwicklung eines integrierten Systems zur Erfüllung eines bestimmten Geschäftsprozesses. Das Vorgehensmodell basiert dabei auf den MDA-Prinzipien (siehe Abschnitt 2.1) und erweitert diese um die Berücksichtigung der Integration bestehender Softwaresysteme. Abbildung 5.1 zeigt eine Übersicht zum MINT-Vorgehensmodell.

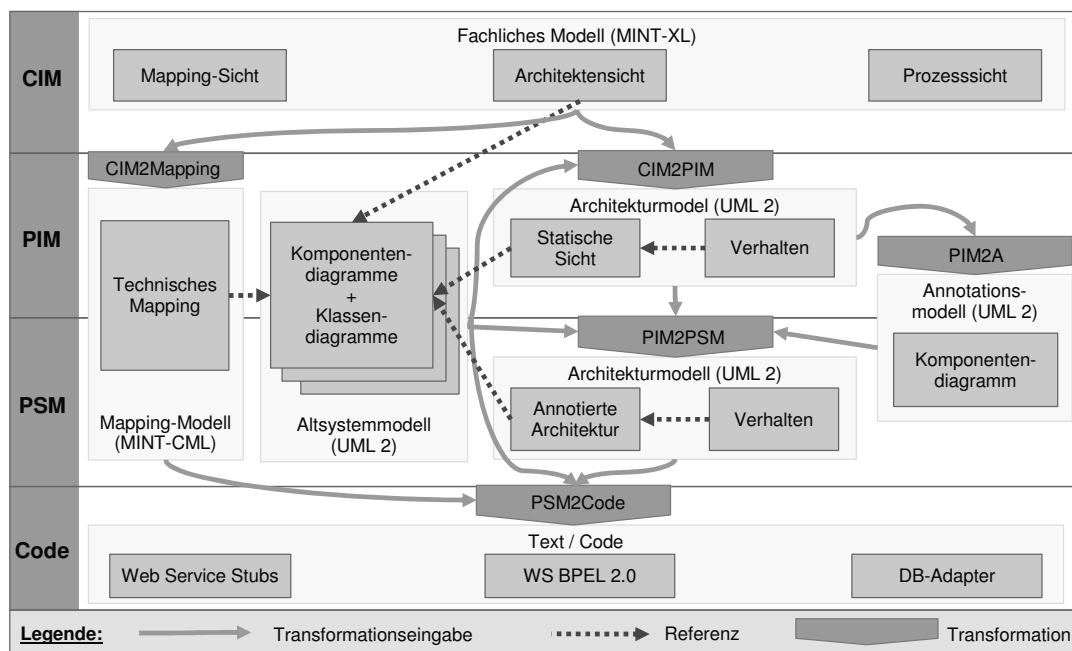


Abbildung 5.1.: Übersicht zum MINT-Vorgehensmodell

In der Abbildung findet sich das MDA-Vorgehensmodell aus Abbildung 2.1 auf Seite 12 wieder. Darin lassen sich auch die beiden Hauptziele des MINT-Projektes wiedererkennen: Die Integration bestehender Systeme in einem modellgetriebenen Entwicklungsprozess und die Verbesserung der fachlichen Modellierung. Für letztere wurde die in Kapitel 3 beschriebene MINT-DSL in das Vorgehensmodell eingeführt. Für die Modellierung der Architektur des integrierten Systems wird die UML verwendet. Auf

der Code-Ebene finden sich sowohl Artefakte für die Integration auf Prozessebene, wie z. B. Webservice-Stubs und Prozessbeschreibungen in BPEL, als auch Artefakte für die in Kapitel 6 beschriebene Datenintegration wie z.B. Datenbankadapter.

Die Integration von bestehenden Systemen zeigt sich auf der linken Seite von Abbildung 5.1. Aus den bestehenden individuellen Altsystemen bzw. bestehender Standardsoftware wird ein sogenanntes Altsystemmodell extrahiert, das die Schnittstellen der Systeme beschreibt. Hier zeigt sich auch die Parallele zum in Kapitel 2.2.2 beschriebenen BALES-Ansatz. Das Altsystemmodell entspricht dem dort durch Reverse-Engineering erstellten PSM-Modell. Wie auch beim BALES-Ansatz werden die Informationen aus dem Forward-Engineering (CIM) und dem Reverse-Engineering (Altsystemmodell) im PIM zusammengeführt. Da im MINT-Vorgehensmodell Informationen aus dem Altsystemmodell auf alle Modellebenen des Forward-Engineering des integrierten Systems (CIM, PIM und PSM) einfließen, bestehen auch in der Abbildung von diesen Ebenen Verbindungen zum Altsystemmodell. Eine detaillierte Beschreibung des Altsystemmodells erfolgt in Abschnitt 5.1.5. Des Weiteren wird im technischen Mapping-Modell die Integration der Daten, die zwischen den verschiedenen Schritten eines Prozesses ausgetauscht werden, beschrieben. Aus diesen Informationen lassen sich z. B. Service-Schnittstellen und Adapter für bestehende Systeme generieren. Solche Adapter werden auch im in Abschnitt 2.2.1 beschriebenen Dublo-Muster verwendet. Weitere Details zu diesem Modell werden in Abschnitt 5.1.6 erläutert.

## 5.1. Modelle

Dieser Abschnitt beschreibt die im MINT-Vorgehensmodell verwendeten Modelle und ihr Zusammenspiel im Rahmen der modellgetriebenen Prozessintegration.

### 5.1.1. Fachliches Modell

Für die Modellierung des fachlichen Modells wird die in Kapitel 3 beschriebene MINT-XL verwendet. Das fachliche Modell gliedert sich, wie in Abbildung 5.2 zu sehen ist, in drei Sichten: die Prozesssicht, die Mapping-Sicht und die Architektensicht.



Abbildung 5.2.: Fachliches Modell im MINT-Vorgehensmodell

Die Prozesssicht beschreibt aus fachlicher Sicht den Integrationsprozess. Hier werden die durchzuführenden Aktionen und ihr Ablauf definiert. Des Weiteren wird festgelegt, welche Systeme bzw. menschlichen Akteure die Aktionen durchführen. Außerdem werden die Daten, die ausgetauscht werden, in Form von Nachrichten beschrieben. Die genauen Schnittstellen der Systeme sind auf dieser Ebene allerdings nicht von Interesse.



Die Modellierung der Prozesssicht ist die Aufgabe von Fachexperten und Softwarearchitekten. Ein Beispiel für ein solches Modell zeigt Abbildung 3.8 auf Seite 64.

In der Mapping-Sicht werden die beim Austausch von Nachrichten verwendeten Daten konzeptuell, d. h. ohne die Angabe von technischen Details wie z. B. Datentypen, modelliert und Abbildungen zwischen Datenrepräsentationen in verschiedenen Systemen definiert. Dabei wird auf dieser Ebene nur festgelegt, welche Daten eines Systems auf die Daten eines anderen Systems abgebildet werden sollen. Technische Details, wie z.B. dabei verwendete Funktionen, werden nicht berücksichtigt. Diese werden im in 5.1.6 beschriebenen technischen Mapping ergänzt. Das fachliche Mapping wird ebenfalls von Fachexperten und Softwarearchitekten gemeinsam modelliert. Die Elemente der Mapping-Sicht sind im Integration-Language-Features-Paket der MINT-XL (siehe Abschnitt 3.3.1) definiert. Die Mapping-Sicht lässt sich auch unabhängig von der Prozessdefinition als fachliche Sicht für die in Abschnitt 6.5.1 beschriebenen Mappings zur Datenintegration verwenden. Abbildung 5.3 zeigt ein einfaches Beispiel für ein fachliches Mapping.

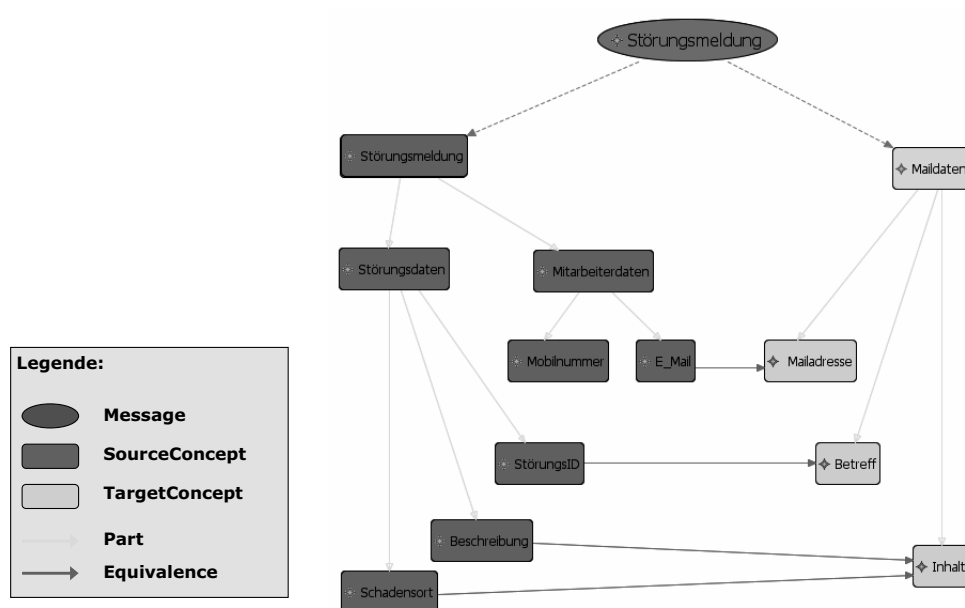


Abbildung 5.3.: Beispiel für ein fachliches Mapping

Die Architektensicht ist streng genommen eine Annotationssicht, in der Softwarearchitekten Informationen ergänzen können, die von der in 5.2.1 beschriebenen Transformation in das plattformunabhängige Architekturmodell benötigt werden. Die Architektursicht erweitert dabei die beiden anderen Sichten durch die Elemente des Technical-Annotations-Pakets der MINT-XL (siehe Abschnitt 3.2).

### 5.1.2. Plattformunabhängiges Architekturmodell

Das plattformunabhängige Architekturmodell (PIM) ist ein UML-Modell, auf das verschiedene Sichten in Form von UML-Diagrammen definiert sind. UML wurde als Modellierungssprache aus folgenden Gründen gewählt:

- Softwarearchitekten beherrschen in der Regel die Verwendung von UML als Modellierungssprache.
- Für die Modellierung von UML-Modellen existiert eine große Bandbreite kommerzieller und nichtkommerzieller Werkzeuge.
- UML ist zur Dokumentation von Softwarearchitekturen in Unternehmen etabliert.

Die Sichten werden unterteilt in die statische und die dynamische Sicht. Dieses Vorgehen ist dem in der Beschreibung von Architekturen in [54, Kapitel 3] sehr ähnlich, jedoch ohne den dort beschriebenen Verteilungsstandpunkt.

#### Statische Sicht

Die statische Sicht beschreibt die strukturelle Architektur des Systems, also die Aufteilung des Systems in Komponenten und Klassen. Die Elemente, die in den verschiedenen Diagrammen beschrieben werden, leiten sich größtenteils aus dem Integrationspaket und den technischen Annotationen der in Kapitel 3 beschriebenen MINT-XL ab. Die Annotationen geben dabei den Komponenten-, Interface- und Operationsnamen, mit dem einzelne Aktionen im fachlichen Modell umgesetzt werden, an.

**Komponentendiagramme** Für die Beschreibung der Struktur wird aus dem fachlichen Modell und dem Altsystemmodell ein UML-Komponentendiagramm erzeugt. Das Komponentendiagramm beinhaltet alle aus dem fachlichen Modell referenzierten Komponenten aus dem Altsystemmodell und alle im fachlichen Modell verwendeten, aber nicht im Altsystemmodell vorhandenen Komponenten. Ein UML-Komponentendiagramm besteht aus den im Folgenden beschriebenen Bestandteilen:

- Eine **Komponente** ist ein modularer Systemteil mit einer transparenten Kapselung seines Inhaltes und besteht aus Elementen mit einer definierten Funktionalität. Eine Komponente wird nur über Interfaces (Schnittstellen) beschrieben.
- Komponenten können **Interfaces** mit Operationen enthalten, mit denen Verbindungen zwischen Komponenten möglich werden.
- Die Spezifikation einer **Operation** besteht grundsätzlich aus dem Namen der Operation und aus einer Liste von Parametern. Wie jedes Merkmal kann eine Operation zusätzlich auch Details zur Sichtbarkeit, zur Multiplizität und zum Typ der Operation spezifizieren.

**Klassendiagramme** Für die Beschreibung der in den Komponenten verwendeten Klassen und der in den Signaturen der Operationen der Interfaces verwendeten Klassen wird ein Klassendiagramm erzeugt.

### **Dynamische Sicht**

Die dynamische Sicht beschreibt das Verhalten des Systems zur Laufzeit. Hier wird der Kontrollfluss und der Datenfluss des Systems modelliert. Die dynamische Sicht wird mit Hilfe von UML-Aktivitätsdiagrammen beschrieben. In diesen finden sich die Informationen der Geschäftsprozessbeschreibungen auf fachlicher Modell-Ebene sowie weitere technische Details wieder.

Die Verbindungen zwischen den Elementen der dynamischen und der statischen Sicht werden mittels UML-Dependencies hergestellt. Mit Dependencies werden beispielsweise die Verweise einer CallBehaviorAction in der dynamischen Sicht zu einer Operation der statischen Sicht realisiert.

### **Erweiterungen durch UML-Profile**

Die Elemente der MINT-XL lassen sich vollständig auf UML-Elemente abbilden, nicht jedoch alle benötigten Attribute. Aus diesem Grund werden Attribute von Elementen der MINT-XL, die im fachlichen Modell beschrieben werden und die keine Entsprechung in UML besitzen, über Stereotypen und tagged Values im plattformunabhängigen Architekturmodell abgebildet. Die verwendeten Stereotypen und tagged Values werden zusammen in einem Profil vorgehalten. Jedes Element im fachlichen Modell verfügt beispielsweise über eine ID, über die es eindeutig im Modell identifiziert werden kann. Diese ID wird im plattformunabhängigen Architekturmodell mit dem Stereotyp IDForAll und einer tagged Value ID abgebildet.

Des Weiteren werden Informationen für weitere Transformationen über Stereotypen abgebildet. Alle Komponenten, Interfaces, Operationen, Klassen und Parameter werden mit einem Stereotypen versehen aus dem ersichtlich ist, ob eine Entsprechung im Altsystemmodell vorhanden ist.

### **5.1.3. Annotationsmodell**

Die zu integrierenden Dienste aus dem Altsystemmodell stammen aus der Produktivumgebung des Unternehmens und sind daher vollständig beschrieben. Für neu zu erstellende Dienste, die im fachlichen Modell referenziert werden und nicht im Altsystemmodell vorhanden sind, werden zusätzliche, plattformabhängige technische Informationen für die Transformation in das plattformabhängige Architekturmodell benötigt. Für die Annotation des plattformunabhängigen Modells mit plattformspezifischen Informationen ist ein separates Modell notwendig. Das Annotieren des plattformunabhängigen Modells ist nicht möglich, da nach jedem erneuten Durchlauf der MDA-Ebenen das plattformunabhängige Architekturmodell und damit auch die Annotationen überschrieben werden würden. Aus diesem Grund werden Annotationen in

einem separaten Modell getätigt; im weiteren Verlauf Annotationsmodell genannt. In diesem Annotationsmodell werden die folgenden Sichten beschrieben.

### **Statische Sicht**

Die statische Sicht des Annotationsmodells entspricht dem Komponentendiagramm des plattformunabhängigen Modells aus Abschnitt 5.1.2. Der Unterschied besteht darin, dass die Elemente des Komponentendiagramms im Annotationsmodell (Komponenten, Interfaces oder Operationen) womöglich keine Entsprechung im Altsystemmodell besitzen. Aus diesem Grund können technische plattformabhängige Informationen annotiert werden, die für die im weiteren Verlauf notwendige Erzeugung neuer Komponenten, Interfaces oder Operationen notwendig sind. Die Informationen werden mit dem gleichen Erweiterungsmechanismus wie im Altsystemmodell über Stereotypen und tagged Values annotiert. Für diesen Zweck wird das gleiche Profil wie im Altsystemmodell verwendet, da für die Umsetzung der neu zu erstellenden Dienste die gleichen technischen Informationen benötigt werden, wie für die existierenden Dienste im Altsystemmodell.

### **Dynamische Sicht**

Die dynamische Sicht des Annotationsmodells entspricht dem Aktivitätsdiagramm des plattformunabhängigen Modells aus Abschnitt 5.1.2. In dieser Sicht lassen sich Annotationen bezüglich plattformabhängiger Informationen, z. B. Details für die Code-Erzeugung anfügen. Damit werden die unterschiedlichen Informationserfordernisse, z. B. für die Erzeugung von BPEL oder von Java-Code, berücksichtigt.

## **5.1.4. Plattformabhängiges Architekturmodell**

Das plattformabhängige Architekturmodell ist ein UML-Modell und besteht aus dem plattformunabhängigen Modell, ergänzt durch die plattformspezifischen Informationen aus dem Altsystemmodell und den technischen Annotationen des Annotationsmodells.

### **Statische Sicht**

Die Struktur der statischen Sicht des plattformabhängigen Architekturmodells entspricht der Struktur der statischen Sicht des plattformunabhängigen Architekturmodells aus Abschnitt 5.1.2. Zusätzlich sind technische plattformabhängige Informationen aus dem Annotationsmodell (siehe Abschnitt 5.1.3) in den zu den UML-Elementen zugewiesenen Stereotypen und tagged Values enthalten.

### **Dynamische Sicht**

Die dynamische Sicht des plattformabhängigen Architekturmodells entspricht der dynamischen Sicht des plattformunabhängigen Architekturmodells aus Abschnitt 5.1.2.

Abbildung 5.4 zeigt einen Auszug mit der aus dem Beispielprozess generierten dynamischen Sicht des plattformabhängigen Architekturmodells. Auf dem UML-Element

CallBehaviorAction mit dem Namen „Stoerung anlegen“ sind die zugewiesenen Stereotypen und die enthaltenen tagged Values zu sehen. Die Werte der tagged Values wurden in der Modell-zu-Modell-Transformation dem Altsystemmodell entnommen.

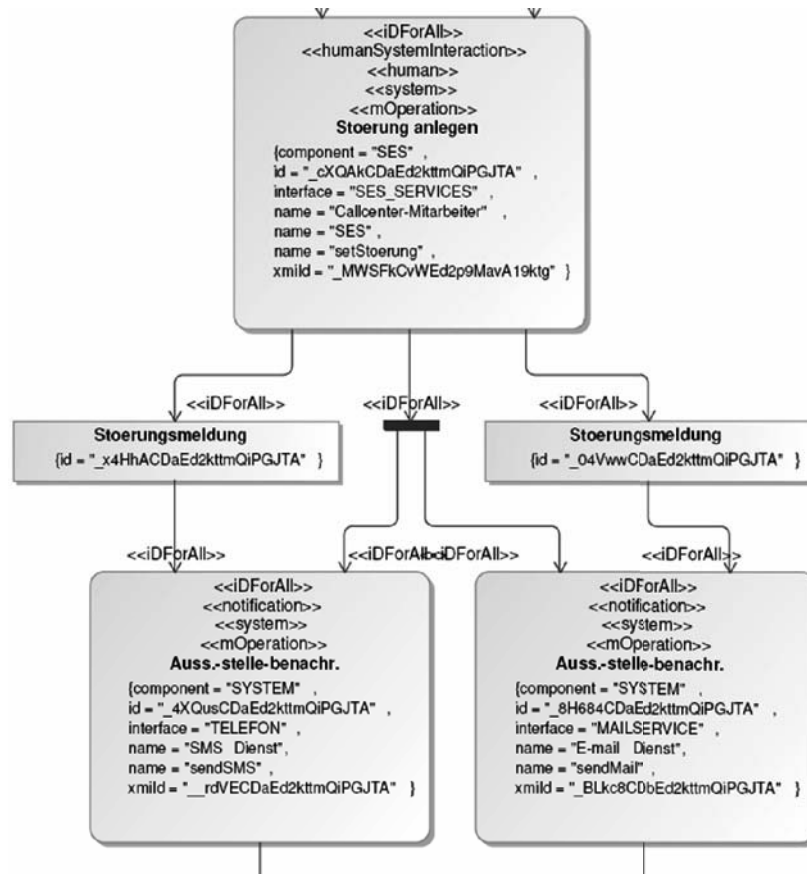


Abbildung 5.4.: Auszug aus dem plattformabhängigen Architekturmodell

### 5.1.5. Altsystemmodell

Das Wachstum von Unternehmen spiegelt sich oftmals in ihrer gewachsenen und dadurch heterogenen IT-Landschaft wieder. Der Ansatz des MINT-Projekts nimmt diese Altsysteme auf und integriert sie in die neu zu erstellenden Prozesse. Im weiteren Verlauf wird aus einem Altsystem ein Altsystemmodell erstellt. Die Altsysteme sind gekennzeichnet durch Unternehmensanwendungen (wie z. B. Individualssoftware oder SAP-Anwendungen) und Dienste die durch diese Anwendungen zur Verfügung gestellt werden. Aus technischer Sicht gibt es in Unternehmen eine Anzahl von Anwendungen die unterschiedliche Schnittstellen für einen Datenaustausch anbieten. Die Schnittstellen können sich unterschiedlichster Technologien bedienen, wie z. B. Webservices, FTP und Dateisystemschnittstellen.

## Aufbau des Altsystemmodells

Die Modellierung des Altsystemmodells erfolgt als UML-Komponentendiagramm. Folgende Gründe sprechen für eine Wahl von UML als Modellierungssprache:

- Sofern das Altsystemmodell manuell erstellt wird, ist davon auszugehen, dass Softwarearchitekten die Anwendung von UML beherrschen.
- Für die Modellierung von UML-Modellen existiert eine große Bandbreite kommerzieller und nichtkommerzieller Werkzeuge.
- Die im weiteren Verlauf verwendeten Transformationen müssen das Altsystemmodell einlesen können. UML-Modelle lassen sich auch in einem XMI-Format abspeichern.

Für die Modellierung des Altsystems als UML-Modell stehen unterschiedliche Werkzeuge zur Auswahl. Im MINT-Kontext findet Magic Draw<sup>1</sup> als UML-Modellierungswerkzeug Verwendung, da es aus technischer Sicht den Vorteil bietet, erstellte UML-Modelle in einem, von dem openArchitectureWare-Framework<sup>2</sup> lesbaren, XMI-Format exportieren zu können.

Eine Möglichkeit zur Erzeugung eines Altsystemmodells ist die manuelle Erstellung, z. B. in MagicDraw. Des Weiteren wurde im Projektverlauf eine rudimentäre WSDL-zu-UML-Transformation entwickelt. Aus Altsystemen, die Komponenten mit Webservices zur Verfügung stellen, kann mittels dieser Transformation automatisch ein Altsystemmodell in UML erzeugt werden. Für Komponenten die Dienste zur Verfügung stellen die auf anderen Protokollen basieren, müssen weiterhin manuell Altsystemmodelle erstellt werden.

Das Altsystemmodell beinhaltet alle bestehenden Dienste (Schnittstellen) bzw. Webservices der Altsysteme eines Unternehmens, aus denen ausgewählte Schnittstellen zu integrieren sind. Auf das Altsystemmodell wird aus unterschiedlichen Ebenen des MINT-Prozesses verwiesen. Jede Ebene benötigt unterschiedliche Informationen, die das Altsystemmodell zur Verfügung stellen muss.

- Im fachlichen Modell werden Verweise auf Operationen im Altsystemmodell modelliert.
- Im technischen Mapping werden Verweise auf Parameter von Operationen modelliert.
- Für die Transformation vom plattformabhängigen Architekturmodell zu Code werden technische Informationen der Operationen im Altsystemmodell benötigt.

Aufgrund dieser Anforderungen werden die Anwendungen als UML-Komponenten modelliert. Für den MINT-Kontext ist eine Komponente eine Sammlung von Diensten, die von außerhalb der Komponente benutzt werden können. Informationen über

---

<sup>1</sup><http://www.magicdraw.com>, letzter Zugriff: 15.12.2008

<sup>2</sup><http://openarchitectureware.org/>, letzter Zugriff: 15.12.2008

den inneren Aufbau der Komponenten sind für die Modellierung des Altsystemmodells nicht notwendig. Die Kommunikation mit den Komponenten erfolgt ausschließlich über Interfaces. Diese stellen eine oder mehrere Operationen für den Datenaustausch zur Verfügung. Interfaces sind z. B. Webservices, die eine Anwendung als Dienste anbietet. Des Weiteren wird in den Operationen der Interfaces festgelegt, mit welchen Parametern der Datenaustausch durchgeführt wird. Die Parameter werden, sofern sie über einen komplexeren Aufbau verfügen, als Klassen mit Attributen modelliert.

Für die unterschiedlichen Ausprägungen der technischen plattformabhängigen Informationen werden die UML-Elemente im Altsystemmodell mit Stereotypen und tagged Values erweitert. In diese werden dann technische Informationen, wie z. B. IP-Adressen des Servers, der einen Dienst zur Verfügung stellt, geschrieben. Die verwendeten Stereotypen und tagged Values werden in einem gemeinsamen Profil vorgehalten. Sollte das zur Verfügung stehende Profil nicht ausreichen, kann dieses mit weiteren tagged Values und Stereotypen erweitert werden.

### **Beispielhaftes Altsystemmodell**

Im Beispielprozess aus Kapitel 2.7.1 wird auf verschiedene Operationen aus unterschiedlichen Anwendungen in einem Unternehmen verwiesen. Diese Dienste wurden in einer Testlandschaft nachgebildet und als Webservices zur Verfügung gestellt. In der Testlandschaft wurde die Operation für die Generierung der Karte mit in die Komponente SES integriert. Über die von den Webservices gelieferten WSDL-Dateien konnte das Beispiel Altsystemmodell mittels einer im Projekt entwickelten WSDL-zu-UML-Transformation erzeugt werden.

Das Altsystemmodell für den Beispielprozess setzt sich folgendermaßen zusammen:

- Komponente SES (**StörungsErfassungsSystem**) mit einem Interface SES mit
  - Operation: getCIMMessage
  - Operation: getStoerung
  - Operation: setStoerung
  - Operation: isVersorgungsgebiet
  - Operation: getKarte
  - Klasse: StoerungsMeldung
  - Klasse: Karte
  - Klasse: CIMStoerung

Die Komponente SES stellt ein Interface mit verschiedenen Operationen für die Störungserfassung zur Verfügung.

- Komponente System mit dem Interface System mit
  - Operation: sendSMS
  - Operation: sendMail

- Klasse: SMSDaten
- Klasse: MailDaten

Die Komponente System stellt ein Interface mit verschiedenen Operationen für den Versand von SMS (Short Message Service) und Email zur Verfügung. Da die Namen aus sprechenden Bezeichnern gebildet wurden, wird an dieser Stelle nicht auf weitere Details eingegangen.

Abbildung 5.5 zeigt beispielhaft die Komponente System mit dem angebotenen Interface System. Das Interface enthält zwei Operationen und verwendet zwei Klassen als Parametertypen.

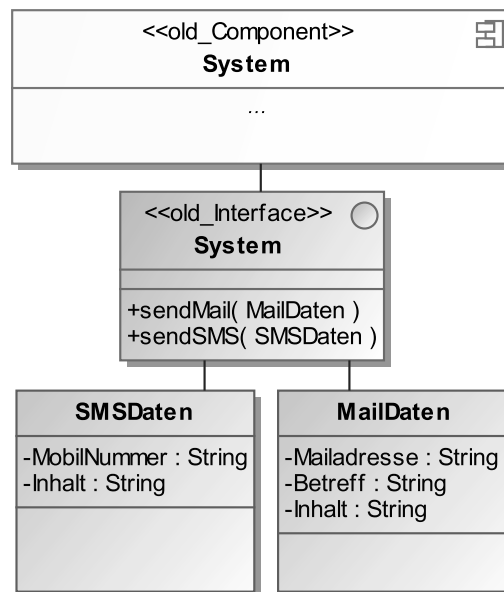


Abbildung 5.5.: Auszug aus dem Altsystemmodell

### 5.1.6. Technisches Mappingmodell

Das technische Mappingmodell stellt eine technischere Sicht auf die auf der fachlichen Ebene definierten Mappings dar. Während auf fachlicher Ebene die Mappings rein konzeptuell beschrieben wurden, kommen hier zusätzliche Informationen für die Umsetzung hinzu. In diesem Modell gibt es zudem Verweise zum Altsystemmodell um einen Bezug zwischen den Konzepten der fachlichen Ebene und den konkreten Schnittstellen der bestehenden Systeme herzustellen. Abbildung 5.6 zeigt den in diesem Abschnitt beschriebenen Ausschnitt aus dem MINT-Vorgehensmodell.

Das Metamodell baut auf dem in Kapitel 3 beschriebenen Metamodell der MINT-XL auf, ist aber unabhängig davon. Das Metamodell besteht aus zwei Paketen: dem struc-Paket, das die grundlegenden Modellkonzepte enthält, und dem ilf-Paket, das



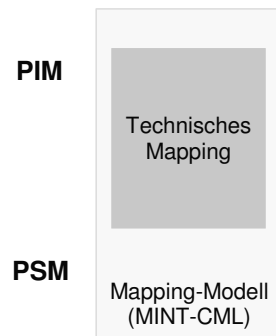


Abbildung 5.6.: Das technische Mapping im MINT-Vorgehensmodell

Teile des ilf-Pakets der MINT-XL übernimmt und erweitert. Abbildung 5.7 zeigt das ilf-Paket des Metamodells.

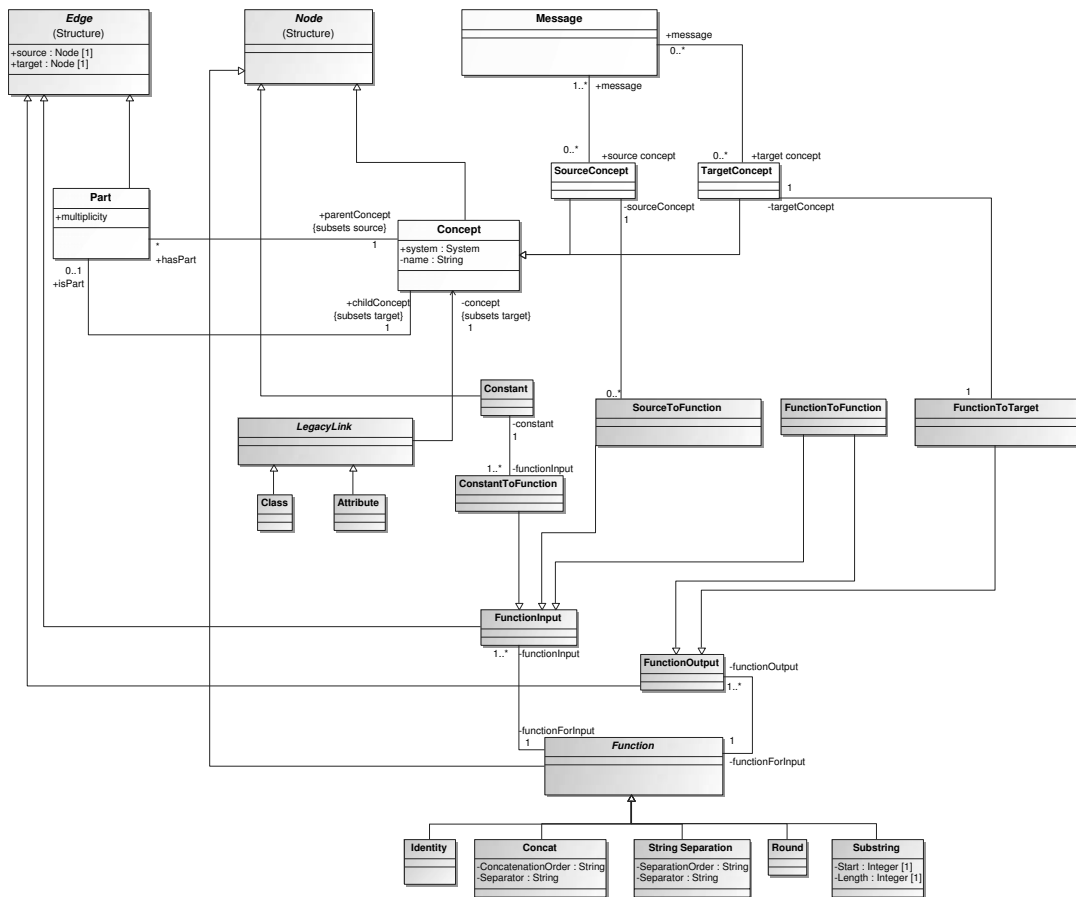


Abbildung 5.7.: ILF-Paket des Metamodells für die technischen Mappings

Die hauptsächliche Erweiterung zu den Mappings auf fachlicher Ebene ist die Ersetzung der Equivalence-Beziehung durch konkrete Funktionen des Zielsystems. Abbildung 5.7 zeigt die abstrakte Klasse *Function*, von der diese konkreten Funktionen erben. Die konkreten Funktionen können eigene Attribute beschreiben, wie z. B. *ConcatenationOrder* und *Separator* für die *Concat*-Funktion. Alle Funktionen können mehrere Ein- und Ausgangsparameter besitzen. Diese lassen sich über die Beziehungen *SourceToFunction* und *FunctionToTarget* mit den entsprechenden Konzepten des Quell- bzw. Zielsystems instanziiieren. Des Weiteren lassen sich Funktionen untereinander derart verbinden, dass ein Ausgabeparameter einer Funktion einem Eingabeparameter einer anderen Funktion entspricht. Zudem lassen sich Konstanten als Eingaben für Funktionen definieren. Das technische Mapping lässt sich auch unabhängig von den Prozess- und Architekturmodellen zur Beschreibung der Mappings zur Datenintegration (siehe Abschnitt 6.5.1) verwenden. Dabei können die Funktionen z. B. für entsprechende Mapping-Patterns stehen. Abbildung 5.8 zeigt das technische Mapping zu dem Beispiel aus Abbildung 5.3.

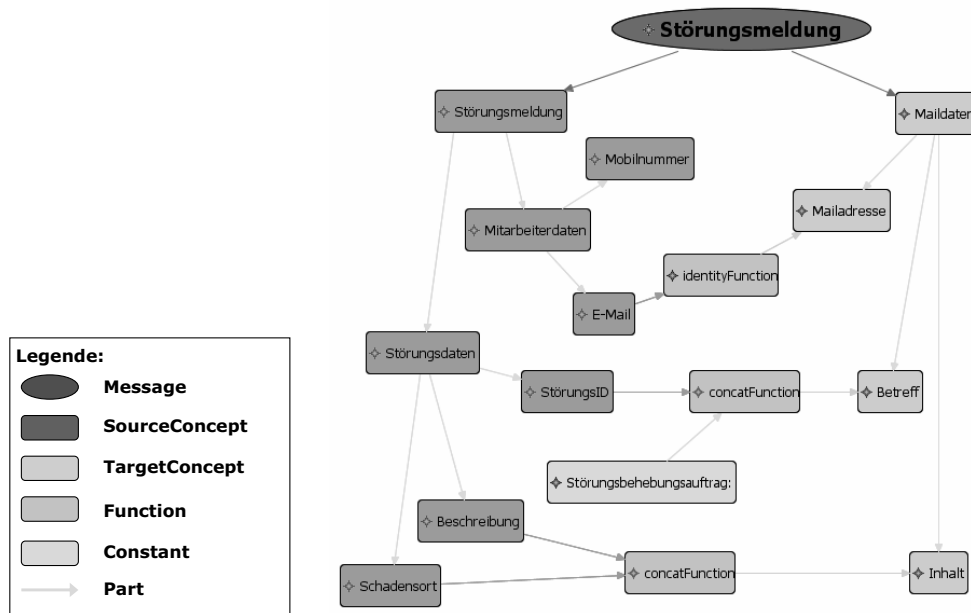


Abbildung 5.8.: Beispiel für ein technisches Mapping

Neben der Modellierung konkreter Funktionen für die Umsetzung der fachlichen Mappings lassen sich im technischen Mapping Referenzen auf das Altsystemmodell definieren. Dies geschieht mit Hilfe der Klasse *LegacyLink*, die z. B. auf Klassen oder Attribute von Klassen im Altsystemmodell verweisen kann. Über diese Referenz können die fachlichen Konzepte auf technische Objekte mit plattformspezifischen Datentypen abgebildet werden.

Um die Umsetzung der Transformationen *CIM2Mapping* und *PSM2Code* zu gewährleisten, können zudem alle Modellelemente des technischen Mappings mit einer

*CIM\_ID* versehen werden. Diese stellt eine Referenz auf das dem Modellelement im technischen Mapping entsprechende Modellelement im MINT-XL-Modell her. Die genaue Verwendung dieser ID folgt in den entsprechenden Transformationsbeschreibung in Abschnitt 5.2.

Das Vorgehensmodell ist so konzipiert, dass die technischen Informationen nicht wie im Fall der PIM2PSM-Transformation über ein Annotationsmodell hinzugefügt werden, sondern direkt im Modell änderbar sind. Dafür wurden Regeln eingeführt, die beschreiben welche Teile des Modells änderbar sind. Grundsätzlich ist es so, dass die Nachrichten und Konzeptbäume nicht verändert werden dürfen, sondern nur an den in Abbildung 5.7 dunkler hervorgehobenen Elementen Änderungen in einem bestimmten Umfang möglich sind. So lassen sich z. B. Funktionen ersetzen, die spezielle Attribute der konkreten Funktionen schreiben und sich neue Beziehungen zwischen Funktionen (*FunctionToFunction*) und Konstanten einfügen. Die Beziehungen *SourceToFunction* und *FunctionToTarget* dürfen hingegen nicht neu erstellt werden und auch die *sourceConcept*- bzw. *targetConcept*-Enden dürfen nicht verändert werden. Es ist nur erlaubt die Beziehungen zu jeweils neu eingefügten Funktionen herzustellen. Dabei muss allerdings auch beachtet werden, dass dabei die auf fachlicher Ebene durch die Equivalence-Beziehung beschriebenen Verbindungen zwischen Quell- und Zielkonzepten erhalten bleiben müssen. Es darf daher keine Änderungen geben, die Auswirkungen auf das CIM haben.

### 5.1.7. Code

Auf der Code-Ebene wird aus dem zuvor im MINT-Prozess modellierten, plattformabhängigen Architekturmodell Code für eine Zielplattform generiert. Die *Business Process Execution Language* (BPEL) [46] bietet als XML-basierte Sprache eine gute Möglichkeit, die Geschäftsprozesse eines Unternehmens durch die Orchestrierung von Webservices abzubilden. Für den Fall, dass die benötigten Dienste noch nicht im Unternehmen existieren oder ihre Dienste nicht über Webservices anbieten, können Webservice-Stubs generiert werden.

#### BPEL

Im MINT-Kontext wird eine Prozessintegration modelliert in der die zu integrierenden Dienste über Webservices angesprochen werden. Für die Orchestrierung der Webservices wird im MINT-Kontext auf der Code-Ebene BPEL aus dem plattformabhängigen Architekturmodell generiert. Für die Code-Erzeugung wird eine Abbildung sowohl von den UML-Elementen des Komponentendiagramms als auch von den UML-Elementen des Aktivitätsdiagramms des plattformabhängigen Architekturmodells auf Elemente von BPEL durchgeführt. Nach der Generierung des BPEL-Skripts kann der Code in einer BPEL-Engine ausgeführt werden. Im Folgenden werden die wichtigsten und für den MINT-Integrationsprozess relevanten Bestandteile von BPEL kurz vorgestellt.

- Im Header wird der Standard Namespace der BPEL definiert

- Partnerlinks verweisen auf alle zugehörigen Kommunikationspartner (Webservices der bestehenden Systeme). Mit dem vom Webservice definierten PartnerlinkType vom Partnerlink wird definiert, welche Operation des Webservices aufgerufen wird. Über die Rolle des in der WSDL-Datei eines Webservice definierten Partnerlinks wird beschrieben, ob der Webservice von anderen Webservices konsumiert wird oder dieser selbst andere Webservices konsumiert.
- Mit den wichtigsten Elementen von BPEL, den Aktivitäten, wird der Ablauf des Prozesses definiert. Es wird festgelegt wie ein Prozess abläuft, wie Informationen zwischen den Kommunikationspartnern über Webservices ausgetauscht werden, wie die Webservices aufgerufen werden und wie der Prozess strukturiert wird.

Das folgende Listing zeigt einen Auszug aus dem mit einem Template generierten BPEL-Skript des in den Abbildungen 3.8 bis 3.10 beschriebenen Beispielprozesses.

In dem flow-Block werden die Webservices sendSMS und sendEmail parallel aufgerufen. In dem ersten sequence-Block werden in dem enthaltenen assign-Block (Zeile 2) zwei Variablen neue Werte zugewiesen. Die Variable *Inhalt* wird aus den Werten der Variablen StörungsID, Schadensort und Beschreibung konkateniert (Zeile 6). Die Funktion concat ist eine XPath-Funktion die von BPEL unterstützt wird. Des Weiteren wird der Variable Mailadresse der Wert aus der Variable E-Mail zugewiesen. Anschließend wird im invoke-Block der Webservice sendMail mit den zuvor aktualisierten Variablen aufgerufen. Das Ergebnis des Webservice-Aufrufs wird in der outputVariable SendMailOut gespeichert. Parallel zu dem ersten sequence-Block werden im zweiten sequence-Block ebenfalls in einem assign-Block zwei Variablen neue Werte zugewiesen. Die Variable Inhalt setzt sich aus der Konkatenation (Zeile 26) von StörungsID, Schadensort und Beschreibung zusammen. Die Variable Mobilnummer erhält den Wert aus der Variable Mobilnummer. Anschließend wird in dem folgenden invoke-Block der Webservice sendSMS mit den zuvor aktualisierten Variablen aufgerufen. Das Ergebnis des Webservice -Aufrufes wird in der outputVariable SendSMSOut gespeichert. In den Aufrufen der Webservices (Zeile 18, 38) sind die Informationen (name, partnerLink, operation, portType) aus dem Altsystemmodell zu sehen.

```
1 <flow>
2   <sequence>
3     <assign>
4       <copy>
5         <from>
6           concat (
7             $GetStoerungOut.parameters/ns0:getStoerungResult/ns0:Stoerungsdaten/
              ns0:StoerungsID,
8             $GetStoerungOut.parameters/ns0:getStoerungResult/ns0:Stoerungsdaten/
              ns0:Schadensort,
9             $GetStoerungOut.parameters/ns0:getStoerungResult/ns0:Stoerungsdaten/
              ns0:Beschreibung)
10          </from>
11          <to>$SendMailIn.parameters/ns0:maildaten/ns0:Inhalt</to>
12        </copy>
13        <copy>
14          <from>$GetStoerungOut.parameters/ns0:getStoerungResult/ns0:Mitarbeiterdaten/
              ns0:E-Mail</from>
```

```

15     <to>${SendMailIn.parameters/ns0:maildaten/ns0:Mailadresse}</to>
16     </copy>
17     </assign>
18     <invoke name="Auss-stelle-benachr_8H684CDaEd2kttmQiPGJTA"
19         partnerLink="SYSTEM"
20         operation="sendMail" portType="ns0:SYSTEMSoap" inputVariable="SendMailIn"
21         outputVariable="SendMailOut"
22     />
23 </sequence>
24 <sequence>
25     <assign>
26         <copy>
27             <from>concat(
28                 ${GetStoerungOut.parameters/ns0:getStoerungResult/ns0:Stoerungsdaten/
29                 ns0:StoerungsID,
30                 ${GetStoerungOut.parameters/ns0:getStoerungResult/ns0:Stoerungsdaten/
31                 ns0:Schadensort,
32                 ${GetStoerungOut.parameters/ns0:getStoerungResult/ns0:Stoerungsdaten/
33                 ns0:Beschreibung )
34             </from>
35             <to>${SendSMSIn.parameters/ns0:smsdaten/ns0:Inhalt}</to>
36         </copy>
37     </assign>
38     <invoke name="Auss-stelle-benachr_4XQusCDaEd2kttmQiPGJTA"
39         partnerLink="SYSTEM"
40         portType="ns0:SYSTEMSoap" operation="sendSMS" inputVariable="SendSMSIn"
41         outputVariable="SendSMSOut"
42     />
43 </sequence>
44 </flow>

```

### BPEL-Skript Auszug aus dem Beispiel-Prozess

Das oben gezeigte BPEL-Skript ist in einer BPEL-Engine ausführbar. Bei der prototypischen Implementierung in MINT wurde die Glassfish-Engine<sup>3</sup> verwendet.

### Webservice-Stubs

Im MINT-Integrationsprozess können nicht nur Dienste aus dem Altsystemmodell integriert werden, sondern auch Dienste, die neu erstellt werden müssen. Um im weiteren Verlauf auf diese Dienste über das erzeugte BPEL-Skript zugreifen zu können, werden Webservice-Stubs generiert. Die Webservice-Stubs werden im MINT-Kontext in C# implementiert. Es sind aber auch andere Programmiersprachen wie z. B. Java, C oder C++ generierbar. Für die Erzeugung der Webservice-Stubs gibt es eine Abbildung der UML-Elemente der statischen Sicht des plattformabhängigen Architekturmodells auf Elemente der Zielsprache, z. B. C#.

Aus den Elementen des technischen Mappingmodells, die in der Reihenfolge im MINT-Integrationsprozess vor und hinter dem neu zu erstellenden Dienst liegen, wer-

<sup>3</sup><https://glassfish.dev.java.net/>, letzter Zugriff: 15.12.2008

UML-Element	C#-Element
Component	C#-Namespace
Interface	C#-Klasse
Operation	C#-Methode

Tabelle 5.1.: Abbildung der UML-Elemente des Komponentendiagramms auf C#-Elemente

den die Ein- und Ausgabeparameter der Webservice-Stubs-Methoden generiert.

Die aus dem generierten C#-Quelltext erzeugten Webservices werden im Anschluss auf einem Webserver deployed. Über die plattformabhängigen Informationen des Komponentendiagramms aus dem plattformabhängigen Architekturmodell können sie dann über das generierte BPEL-Skript aus der BPEL-Engine heraus aufgerufen werden.

## 5.2. Transformationen

In diesem Abschnitt werden die notwendigen Modell-zu-Modell- und Modell-zu-Text-Transformationen zwischen den verschiedenen Ebenen des MINT-Prozesses erläutert.

### 5.2.1. Vom fachlichen Modell zum plattformunabhängigen Architekturmodell

Abbildung 5.9 zeigt, an welcher Position im MINT-Vorgehensmodell die Transformation vom fachlichen Modell zum plattformunabhängigen Architekturmodell stattfindet.

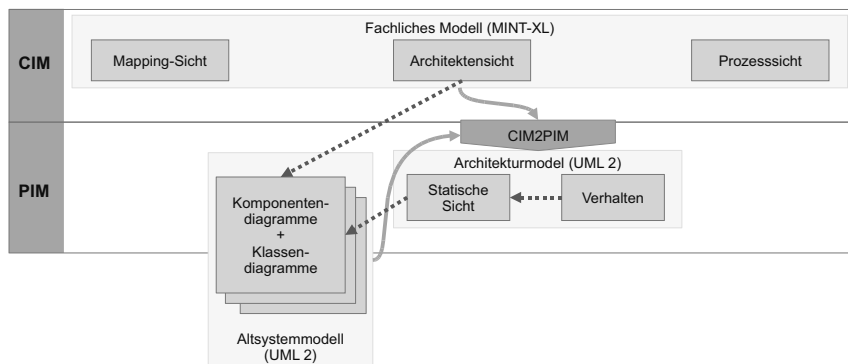


Abbildung 5.9.: Übersicht der CIM-zu-PIM-Transformation

Das fachliche Modell wird mittels einer Modell-zu-Modell-Transformation in ein plattformunabhängiges Architekturmodell transformiert. Für die Durchführung der Modell-zu-Modell-Transformation werden folgende Artefakte benötigt:

- Ein in der MINT-XL modelliertes fachliches Modell

- Das MINT-XL-Metamodell
- Ein Altsystemmodell (siehe Abschnitt 5.1.5)
- Die Modell-zu-Modell-Transformation

Die Grundelemente der MINT-XL können 1-zu-1 in ein UML-Aktivitätsdiagramm abgebildet werden. Tabelle 5.2 zeigt diese Abbildung im Detail.

MINT-XL-Element	UML
Action	CallBehaviorAction
Concept	Class
Decision	DecisionNode
Start	InitialNode
End	ActivityFinalNode
Msg2ActDF	ObjectFlow
Act2MsgDF	ObjectFlow
ProcessFlow	ControlFlow
ParallelJoin	JoinNode
ParallelSplit	ForkNode
Loop	LoopNode
Merge	MergeNode
Message	CentralBufferNode

Tabelle 5.2.: Mapping der MINT-XL-Elemente auf UML für Aktivitätsdiagramme

Während der Modell-zu-Modell-Transformation wird überprüft, ob die im fachlichen Modell referenzierten Dienste im Altsystemmodell vorhanden sind. Der Status des Vorhandenseins wird mit den Präfixen „old“ und „new“ in den Namen der Stereotypen des Komponentendiagramms ausgedrückt. Weiterhin werden alle Stereotypen und tagged Values der referenzierten Dienste im plattformunabhängigen Modell angelegt, jedoch noch ohne die vorhandenen technischen plattformabhängigen Informationen. Aus dem fachlichen Modell, das sich aus Elementen der MINT-XL zusammensetzt, wird durch die Modell-zu-Modell-Transformation ein plattformunabhängiges Architekturmodell in UML erzeugt. Dabei wird die Abbildung der Elemente der MINT-XL auf UML aus Tabelle 5.2 umgesetzt.

Bei der Transformation des fachlichen Modells in die verschiedenen UML-Diagramme des plattformunabhängigen Architekturmodells werden folgende Transformationsregeln angewendet (Abbildung 5.10 zeigt das Vorgehen im Einzelnen):

1. Jedes Element mit Ausnahme der Erweiterungen aus dem fachlichen Modell hat eine Entsprechung in UML (siehe Tabelle 5.2)
2. Die Erweiterungen werden auf die Entsprechungen der erweiterten Elemente im fachlichen Modell abgebildet. Die durch die Erweiterungen zusätzlich angelegten Properties werden als tagged Value weitergegeben. Die dafür notwendigen

- Stereotypen und tagged Values werden in zusätzlichen Profilen definiert und geladen.
3. Die impliziten bzw. optionalen Elemente des fachlichen Modells ParallelJoin und ParallelSplit werden explizit in dem plattformunabhängigen Architekturmodell in UML als ForkNode und JoinNode angelegt.
  4. Die Erweiterungs-Elemente des fachlichen Modells wie z. B. System beinhaltet das Linking auf eine Operation im Altsystemmodell. Dieses Linking wird als Dependency in das UML-Element CallBehaviorAction eintragen.
  5. Die Properties wie z. B. die eindeutigen IDs aus dem fachlichen Modell werden als tagged Value weitergegeben.
  6. Komponenten, Interfaces und Operationen die aus dem fachlichen Modell referenziert werden bekommen den Stereotype „Old“ zugewiesen, wenn sie im Altsystemmodell vorhanden sind.
  7. Komponenten, Interfaces und Operationen, die aus dem fachlichen Modell referenziert werden bekommen den Stereotype „New“ zugewiesen, wenn sie im Altsystemmodell nicht vorhanden sind.

Das durch die Modell-zu-Modell-Transformation erstellte plattformunabhängige Architekturmodell in UML darf nicht manuell verändert werden, da diese Änderungen bei einer erneuten Modell-zu-Modell-Transformation überschrieben werden würden. Um dieses zu verhindern könnten die Modell-zu-Modell-Transformationen so angepasst werden, dass sie Änderungen an den Modellen erkennen und entsprechend berücksichtigen. Das ist jedoch ein sehr aufwändiges Verfahren. Daher werden Änderungen, die die plattformunabhängige Architektur betreffen, entweder als Annotation in der technischen Sicht auf das fachliche Modell oder in entsprechenden Annotationsmodellen vorgenommen.

Durch den iterativen Entwicklungsprozess besteht die Möglichkeit, dass es auch unvollständige plattformunabhängige Architekturmodelle geben kann. Aus diesen Modellen kann im weiteren Verlauf zwar kein Code generiert werden, aber sie können als Grundlage für Analysen der Architektur bzw. zur Aufdeckung nicht oder ungenau spezifizierter Anforderungen verwendet werden. Es werden so viele Iterationen durchgeführt, bis alle Anforderungen im fachlichen Modell vollständig spezifiziert sind.

### 5.2.2. Vom fachlichen Modell zum technischen Mapping

Dieser Abschnitt beschreibt die Transformation vom fachlichen Mapping in der MINT-XL in das technische Mapping. Abbildung 5.11 zeigt den entsprechenden Ausschnitt aus dem MINT-Vorgehensmodell.

Wie schon in 5.1.6 beschrieben, baut das Metamodell des technischen Mappings sehr stark auf dem Metamodell der MINT-XL auf. Das führt dazu, dass große Teile der Transformation nur eine direkte Übersetzung der jeweiligen Elemente darstellen.



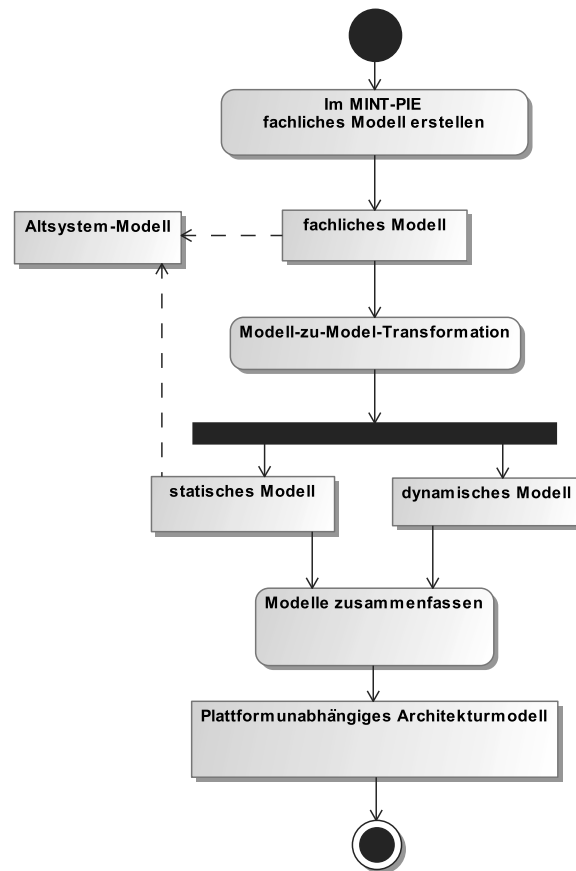


Abbildung 5.10.: Der Workflow vom fachlichen Modell zum plattformunabhängigen Architekturmodell

Die einzige Ausnahme stellt dabei die Equivalence-Beziehung der MINT-XL dar. Diese abstrakte Darstellung einer Abbildung zwischen den Elementen in zwei verschiedenen Systemen, wird im technischen Mapping durch detaillierte Funktionen ersetzt, die für die Umsetzung in Code notwendig sind.

Die Transformation legt dabei zunächst Default-Funktionen an, die von Entwicklern mit den benötigten Funktionen ersetzt werden müssen. Dabei fasst die Transformation auch mehrere Equivalence-Beziehungen zusammen, wenn diese das gleiche Zielkonzept haben. Der Grund dafür ist, dass jedes Zielkonzept nur durch den Ausgang einer Funktion geschrieben werden kann. Würde das Metamodell es zulassen, dass ein Zielkonzept durch mehrere Funktionen geschrieben werden kann, würde an dieser Stelle ein nicht gewünschter Nichtdeterminismus auftreten.

Eine weitere Besonderheit der Transformation ist, dass wie in 5.1.6 erläutert, eine manuelle Modellierung im generierten Modell ermöglicht wird. Dieses wird dadurch gewährleistet, dass während der Transformation geprüft wird, ob für die zu transformierenden Equivalence-Beziehungen bereits Funktionen im bestehenden technischen Map-

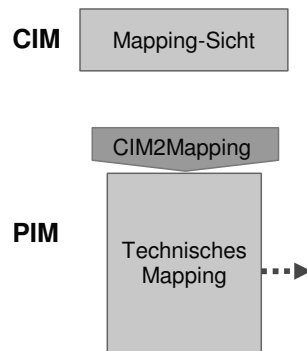


Abbildung 5.11.: Die Transformation vom CIM zum PIM im MINT-Vorgehensmodell

pingmodell existieren. Ist das der Fall, werden diese übernommen. Auf diese Weise werden nur für im fachlichen Modell neu hinzugekommene Equivalence-Beziehungen auch neue Default-Funktionen angelegt. Auch das Löschen von Equivalence-Beziehungen im fachlichen Modell führt zu keinen Problemen bei der Transformation, da die entsprechenden Funktionen nicht übernommen werden und somit in der aktualisierten Version des technischen Mappings nicht mehr vorhanden sind.

Ein Roundtrip-Engineering, also das Einfügen von Equivalence-Beziehungen in das fachliche Mapping für bestehende Funktionen im technischen Mapping, ist nicht vorgesehen. Da das bestehende technische Mappingmodell von der Transformation überschrieben wird, heißt das auch, dass das Löschen von Equivalence-Beziehungen im fachlichen Modell zum Verlust der entsprechenden Informationen im technischen Mapping führen, wenn nicht entsprechende Versionierungsmechanismen eingeführt werden.

### 5.2.3. Vom plattformunabhängigen Architekturmodell zum Annotationsmodell

In Abschnitt 5.1.3 wurde die Notwendigkeit eines separaten Annotationsmodells beschrieben. Die für das Annotieren notwendigen Transformationen werden im weiteren Verlauf detailliert erläutert.

Abbildung 5.12 zeigt, an welcher Position im MINT-Vorgehensmodell die Transformation vom plattformunabhängigen Architekturmodell zum Annotationsmodell stattfindet.

Für die Erstellung eines initialen Annotationsmodells wird eine Kopie des plattformunabhängigen Architekturmodells angelegt. Gegenstand der zu tätigen Annotationen sind diejenigen Elemente in der statischen Sicht, die das Präfix „new“ in ihrem Stereotype tragen. Das sind genau die Dienste, die nicht im Altsystemmodell vorhanden und somit neu zu erstellen sind. In die über Profile hinzugefügten Stereotypen und tagged Values können nun technische Informationen annotiert werden.

Wird nun die im Projektverlauf entwickelte Modell-zu-Modell-Transformation angestoßen, werden die Inhalte der tagged Values in der statischen Sicht des plattformun-

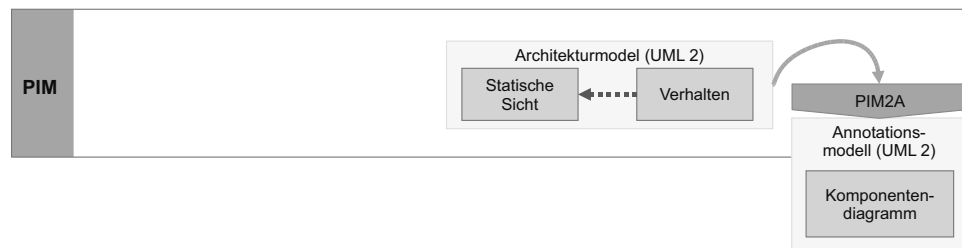


Abbildung 5.12.: Übersicht der PIM-zu-Annotationsmodell-Transformation

abhängigen Architekturmodells mit den Werten der tagged Values im Annotationsmodell verglichen. Ist ein tagged Value im initialen Annotationsmodell mit einem Wert belegt, wird dieser in das Ergebnis-Annotationsmodell dieser Modell-zu-Modell-Transformation geschrieben. Wenn kein Wert im betrachteten tagged Value des Annotationsmodell steht, wird der Wert dieses tagged Values aus dem plattformunabhängigen Architekturmodell gelesen und in das Ergebnis-Annotationsmodell geschrieben. Das Ergebnis-Annotationsmodell besteht am Ende der Iteration aus allen neu zu erstellenden Diensten der statischen Sicht des plattformunabhängigen Architekturmodells, annotiert mit technischen Informationen.

Durch dieses Vorgehen ist Folgendes sichergestellt:

- Das plattformunabhängige Architekturmodell wird nur gelesen und nicht beschrieben.
- Bei weiteren Iterationen des MINT-Prozesses und damit verbundenen Modell-zu-Modell-Transformationen hin zu einem plattformunabhängigen Architekturmodell werden keine Annotationen überschrieben.
- Alle technischen Annotationen befinden sich in einem separaten Modell.

#### 5.2.4. Vom plattformunabhängigen zum plattformabhängigen Architekturmodell

Im Folgenden wird die Modell-zu-Modell-Transformation vom plattformunabhängigen zum plattformabhängigen Architekturmodell in UML beschrieben. Grundlage der Transformation sind:

- Das plattformunabhängige Architekturmodell,
- das Altsystemmodell und
- das Annotationsmodell.

Abbildung 5.13 zeigt, an welcher Position im MINT-Vorgehensmodell die Transformation vom plattformunabhängigen zum plattformabhängigen Architekturmodell stattfindet.

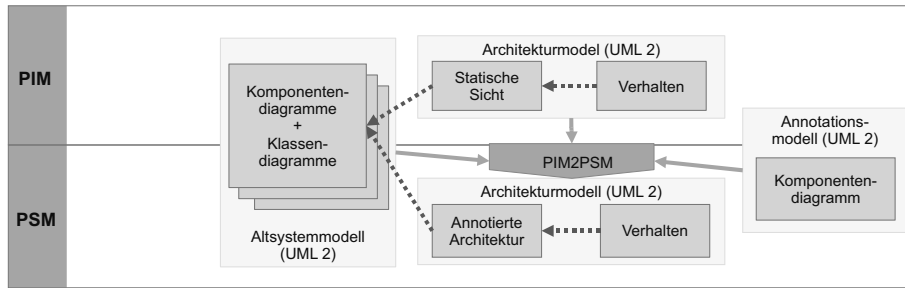


Abbildung 5.13.: Übersicht der PIM zu PSM Transformation

Das Komponentendiagramm des plattformabhängigen Architekturmodells besteht aus den im plattformunabhängigen Architekturmodell definierten Elementen. Die für die Elemente des Komponentendiagramms notwendigen plattformabhängigen Informationen bereits bestehender Dienste im Unternehmen werden dem Altsystemmodell entnommen und in entsprechende Stereotypen bzw. tagged Values geschrieben. Aus dem Annotationsmodell werden die plattformabhängigen Informationen von nicht im Altsystemmodell bzw. im Unternehmen existierenden und somit neu zu erstellenden Diensten entnommen und in entsprechende Stereotypen bzw. tagged Values geschrieben. Das Aktivitätsdiagramm des plattformabhängigen Architekturmodells entspricht dem des plattformunabhängigen Architekturmodells. Mit diesen Informationen ist das plattformabhängige Architekturmodell vollständig beschrieben, sodass im letzten Schritt des MINT-Prozesses der Code der Zielplattform generiert werden kann. Abbildung 5.14 zeigt das Vorgehen im Einzelnen.

### 5.2.5. Vom plattformabhängigen Architekturmodell zu Code

In diesem Abschnitt wird das Vorgehen vorgestellt, wie mittels einer Modell-zu-Text-Transformation die plattformabhängigen Architekturmodelle in Code einer Zielplattform transformiert werden. Eingaben sind die vollständig annotierten plattformabhängigen Architekturmodelle, deren Erstellung im vorherigen Abschnitt beschrieben worden ist. Für jede Zielplattform wird ein plattformabhängiges Architekturmodell und ein Template für die jeweilige Zielsprache benötigt. Des Weiteren wird ein technisches Mappingmodell für die detaillierten Aufrufe der Webservices benötigt. Mittels der Templates wird die Abbildung der Elemente des plattformabhängigen Architekturmodells auf Elemente der Zielsprache gesteuert (siehe Tabelle 5.3).

Abbildung 5.15 zeigt, an welcher Position im MINT-Vorgehensmodell die Transformation vom plattformabhängigen Architekturmodell zu Code stattfindet.

Im MINT-Projekt sind auf der Code-Ebene die Erzeugung von Webservice-Stubs und BPEL umgesetzt worden. Im weiteren Verlauf wird die Modell-zu-Text-Transformation anhand der Generierung von BPEL beschrieben. Für die Modell-zu-Text-Transformation wurde ein Template in XPand entwickelt, das die Elemente des plattformabhängigen Architekturmodells auf BPEL-Artefakte abbildet. Durch das techni-

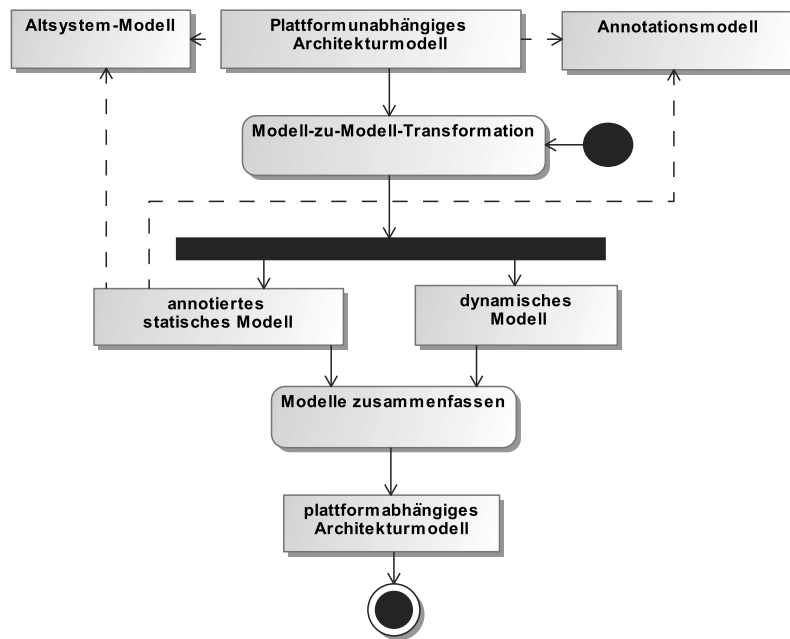


Abbildung 5.14.: Der Workflow vom plattformunabhängigen zum plattformabhängigen Architekturmodell

sche Mappingmodell und die dynamische und statische Sicht des plattformabhängigen Architekturmodells wird der Integrationsprozess vollständig beschrieben. BPEL wird aus diesen Modellen mittels einer Modell-zu-Text-Transformation erzeugt. Tabelle 5.3 zeigt die Beziehung zwischen den UML-Elementen des Aktivitätsdiagramm und dem BPEL-Skript.

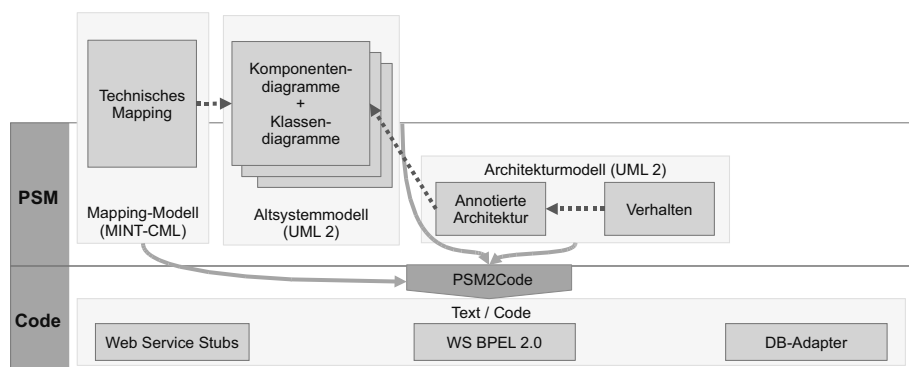


Abbildung 5.15.: Die Transformation vom plattformabhängigen Modell zu Code

UML-Element des Aktivitätsdiagramms	BPEL	Beschreibung
InitialNode, FinalNode	<process>... </process>	Start- und Endpunkt eines Prozesses
CallBehaviorAction	<invoke/>	Aufruf eines Webservices
CentralBufferNode	<assign/>	Aktualisierung einer Variablen mit neuen Daten
ForkNode, JoinNode	<flow>... </flow>	Parallele Verarbeitung von mehreren Aktivitäten
DecisionNode, MergeNode	<if>... </if>	Kontrollstruktur für Prozesse
StrutredActivityNode	<scope />	Unterprozess eines Prozesses

Tabelle 5.3.: Abbildung von UML-Elementen der dynamischen Sicht des plattformabhängigen Architekturmodells auf BPEL

### 5.3. Vorgehen

Nachdem die vorherigen Abschnitte die im MINT-Vorgehensmodell verwendeten Modelle und Transformationen beschrieben haben, wird in diesem Abschnitt der durch das Vorgehensmodell definierte Ablauf für die modellgetriebene Prozessintegration und die dabei auftretenden Rollen beschrieben.

Das in diesem Kapitel vorgestellte Vorgehensmodell beschreibt, wie auch das MDA-Vorgehen, einen iterativen Prozess. Die einzelnen Phasen dieses Prozesses hängen dabei vom individuellen Entwicklungsverlauf ab. Ein möglicher Ablauf sieht dabei so aus, dass zunächst eine erste Version des fachlichen Modells von Fachexperten allein oder in Zusammenarbeit mit Softwarearchitekten erarbeitet wird. Aus dieser ersten Versionen lassen sich bereits technische Modelle ableiten. Allerdings werden die Informationen im Normalfall nicht ausreichen um ablauffähigen Code zu generieren. Hierzu fehlen noch Informationen, welche Schnittstellen der bestehenden Systeme angesprochen werden sollen und wie diese ggf. aufeinander abgebildet werden müssen.

Die Zuordnung von konkreten technischen Schnittstellen der bestehenden Systeme zu Aktionen im beschriebenen Prozess lassen sich durch den Softwarearchitekten in der Architektensicht der MINT-XL mit Hilfe von Referenzen auf das Altsystemmodell durchführen. Nach dieser Zuordnung lässt sich auf der technischen Ebene durch einen Vergleich der Schnittstellen feststellen welche Abbildung zwischen diesen durchgeführt werden müssen. Handelt es sich bei diesen Abbildungen um rein technische Aspekte, können sie von Softwarearchitekten oder Entwicklern vorgenommen werden. Ist es allerdings notwendig, dass spezielles fachliches Wissen in den Abbildung verlangt

wird, können diese Abbildungen in der Mapping-Sicht der MINT-XL von Fachexperten durchgeführt werden. Die Abbildungen aus der fachlichen Sicht müssen auf der technischen Ebene durch Softwarearchitekten oder Entwickler im technischen Mappingmodell durch weitere Informationen wie Datentypen und konkret auszuführende Funktionen präzisiert werden.

Der beschriebene Prozess für die Modellierung von Integrationsprozessen und Abbildungen lässt sich iterativ wiederholen. Dabei können in jeder Iteration weitere Details zu den entsprechenden Modellen hinzugefügt und Änderungen an den Modellen vorgenommen werden. Stehen ausreichende Informationen zur Verfügung, lassen sich auch die entsprechenden Code-Artefakte generieren. Dieser Code muss eventuell durch manuelle Ergänzungen von Entwicklern erweitert werden.

Generell lässt sich sagen, dass Änderungen an den oben angegebenen Modellen immer auf der adäquaten und höchstmöglichen Ebene vorgenommen werden sollten. D. h., wenn Änderungen aufgrund von fachlichen Entscheidungen gemacht werden, sollten diese auch im CIM festgehalten werden. Technische Änderungen sollten auf den entsprechend tiefer liegenden Ebene vorgenommen werden. Architekturentscheidungen werden z. B. auf PIM-Ebene und Plattformscheidungen auf PSM-Ebene getroffen. Da die generierten Modelle in der Umsetzung nicht verändert werden sollten, kann es z. B. bei Informationen, die die PIM-Ebene betreffen, vorkommen, dass aus pragmatischen Gründen kein zusätzliches Annotationsmodell eingeführt wird, in dem die Änderungen mit Referenzen zu bestehenden Entitäten des CIM eingefügt werden, sondern dass diese Informationen direkt als technische Annotationen in das CIM eingefügt werden. Dafür sollte aber immer eine zusätzliche Sicht auf das CIM, z. B. in Form eines zusätzlichen Editors, eingefügt werden, so dass diese technischen Informationen den Fachexperten verborgen bleiben.





# 6. Modellgetriebene Datenintegration

*Cord Giese*

Im Rahmen von MINT war die Untersuchung der Datenintegration eine zentrale Aufgabe der Delta Software Technology GmbH (DSTG). Auch in Bezug auf die Datenintegration wurde in MINT ein modellgetriebener Ansatz verfolgt, welcher sich an der MDA orientiert (vgl. 5). Konkret stand in MINT die modellbasierte Kopplung moderner, objektorientiert modellierter Geschäftslogik mit bestehenden relationalen Datenbanksystemen im Mittelpunkt der Aktivitäten.

In den folgenden Abschnitten wird zuerst das Thema „Persistenzadapter“ allgemein erläutern, und dann auf die Generierung der Persistenzadapter mit SCORE eingegangen. Das Werkzeug SCORE wird im Grundlagen-Kapitel 2.5.1 vorgestellt. Hier geht es um konkrete Erweiterungen, Beispiele und Erfahrungen im Zusammenhang des Einsatzes von SCORE in MINT. Die weitere Struktur dieses Kapitels orientiert sich an SCORE-Arbeitsschritten bzw. -Schichten. Hier sind insbesondere die Datenzugriffsschicht und die Service-Schicht gemeint, welche sich aus der SCORE-Schichtenarchitektur ergeben (s. Abb. 2.8). Den Abschluss bildet eine Zusammenfassung.

## 6.1. Persistenzadapter

Die Abbildung 2.22 zeigt die Schichten der im Rahmen von MINT verwendeten Evaluierungsumgebung (vgl. Kap. 2.8). Verschiedene Client-Applikationen verwenden ein gemeinsames Objektmodell und, basierend auf diesem, eine gemeinsame Geschäftslogik. Zudem arbeiten diese Applikationen auf einer gemeinsamen Oracle-Datenbank. Die technische und semantische Abbildung des Objektmodells auf Oracle-DB-Operationen erfolgt durch Persistenzadapter, die – je nach gewählter Technik – auf einem zusätzlichen Persistenz-(Laufzeit-)System basieren. Teil dieser Abbildung ist insbesondere das erforderliche Objekt-relationale Mapping.

Es wurden sehr unterschiedliche Adapter-Konzepte untersucht: Generische Adapter unterstützen alle Testapplikationen, sind jedoch spezifisch für das eingesetzte DBMS sowie das DB-Schema. Sie sind datenzentriert und implementieren ein dem DB-Schema entsprechendes Objektmodell. Es gibt statische Lösungen, die für das konkrete Objektmodell entwickelt wurden, sowie dynamische Lösungen, die ein Laufzeit-Mapping durchführen und entsprechend konfiguriert werden. Im Gegensatz dazu implementieren generierte Adapter ein Mapping, das spezifisch für die jeweilige Applikation ist.

Das von DSTG entwickelte generative Werkzeug SCORE<sup>®</sup> Adaptive Bridges – Data Architecture Integration<sup>™</sup> (kurz: SCORE) ist eine der im Projekt untersuchten

Lösungsalternativen. Hierbei implementieren vollständig generierte Adapter die Abbildung der „Service Interfaces“ auf die Datenobjekte (vgl. Kap. 2.5). Die im Rahmen von MINT generierten Adapter waren Gegenstand der Evaluierung in der Testumgebung (vgl. Kap. 2.8 sowie Kap. 7). Dabei ermöglichten Testapplikationen mit unterschiedlichen Datenzugriffscharakteristiken einen aussagekräftigen Vergleich der verschiedenen Adapter-Konzepte.

## 6.2. Persistenzadapter-Generierung mit SCORE

DSTG wurden eine Oracle-Datenbank sowie Testanwendungen bereitgestellt. Die Datenbank enthält die Anwendungsdaten von *MESCOR*-Applikationen (vgl. Kap. 2.8), d. h. es sind Realdaten. Dies ist insofern von Bedeutung, da reale Datenbanken erfahrungsgemäß Unzulänglichkeiten aufweisen, mit denen die von DSTG eingesetzten Werkzeuge zurechtkommen müssen. In diesem Fall traten Tabellen ohne Primärschlüssel, formal inkonsistente Datentypen bei Fremdschlüssel-Attributen, sowie nicht explizit definierte Fremdschlüsselbeziehungen auf. Die Handhabung dieser Dinge wird weiter unten beschrieben. DSTG wurden drei Testanwendungen als Basis für die Werkzeugentwicklung geliefert:

- ChartProduction
- Accounting
- ImportTool

*ChartProduction* ist eine grafisch-interaktive Anwendung, die ausschließlich lesende Datenbankzugriffe durchführt. Diese Lesezugriffe können selektiv und als Batch verarbeitet werden. *Accounting* ist ebenfalls eine grafisch-interaktive Anwendung, die selektive lesende und schreibende Zugriffe erlaubt. Die Anwendung *ImportTool* schließlich führt ausschließlich schreibende Zugriffe im Batch-Betrieb durch. Ausführlich erläutert wird die gesamte Testumgebung in Abschnitt 2.8.

Für die drei oben genannten Testapplikationen wurden in MINT zunächst prototypische Persistenzadapter bei DSTG implementiert. Diese Prototypen dienten zum einen zur Gewinnung von Know-how, zum anderen als Vorlage für Anpassungen in den Werkzeugen, insbesondere zur Erstellung des Adapter-Generators für die neue Zielplattform C# / ADO.NET / Oracle. Anschließend wurden die Persistenzadapter vollständig mit SCORE generiert und in die Evaluierung einbezogen (s. Kap. 7).

In den folgenden Abschnitten werden die in Kap. 2.5 vorgestellten Konzepte anhand von Beispielen konkretisiert, und insbesondere die durch MINT motivierten Erweiterungen und die im Projekt gemachten Erfahrungen vorgestellt.

## 6.3. Schema-Import

Alle drei Anwendungen arbeiten auf einer gemeinsamen Datenbank. Sie verwenden dabei unterschiedliche, sich überlappende, Ausschnitte des DB-Schemas. Zuerst wur-

de bei DSTG ein Data Definition File für das Beispiel-DB-Schema erstellt. Im Rahmen von MINT wurde die automatische Ableitung des Data Definition File aus dem DB-Schema implementiert: Das Ergebnis ist der DataDef-Generator. Implementiert wurde dieser in der DSTG-eigenen Generatorsprache ANGIE [22]. Technisch wird im DataDef-Generator das DB-Schema per ODBC angesprochen, da der entsprechende SQL-Funktionsumfang von Oracle nicht ausreichend unterstützt wird. Dazu war es erforderlich, ODBC in ANGIE aufrufbar zu machen. Bewerkstelligt wurde dies durch die Entwicklung eines ANGIE-Add-Ins [12] für ODBC.

Es folgt ein Ausschnitt aus dem generierten Data Definition File (s. 2.5.1), der das „Data Object“ *WAEHRUNG* definiert:

```
begin data object: WAEHRUNG

    begin logical record: WAEHRUNG, DEFAULT NOT NULL
        , indicator set = manual
        data element: WHRNR,          numeric(6)
        data element: WHRKURZBEZ, varchar(3), with null
    end logical base record

    begin physical base record: WAEHRUNG,
        db-type = ORA,
        external = WAEHRUNG,
        DEFAULT NOT NULL
        data element: WHRNR,          numeric(6)
        data element: WHRKURZBEZ, varchar(3), with null
    end physical base record

    begin entry path: WHRNR, C
        key element: WHRNR
    end entry path

end data object
```

Das Data Object entspricht der gleichnamigen Tabelle im DB-Schema. Im „Logical Record“ wird die abstrakte, DBMS-unabhängige Schnittstelle beschrieben, während der „Physical Base Record“ die Oracle-spezifische Ausprägung definiert. Der „Entry Path“ definiert einen Primärschlüssel.

Im DB-Schema enthaltene Views werden von SCORE wie Tabellen behandelt und jeweils als Data Object beschrieben. Das Data Definition File enthält auch alle definierten Fremdschlüssel-Beziehungen.

Das Data Definition File muss für die nachfolgende Verarbeitung vollständig sein. Im konkreten Fall war dies nach der automatischen Ableitung noch nicht der Fall (s. 6.2). Ergänzungen wurden in Include-Dateien untergebracht, um den generierten Code unverändert lassen zu können. Die vervollständigte Data Definition wurde automatisch in ein Templates Repository überführt.

## 6.4. Die Datenzugriffsschicht

Während es für alle drei Testapplikationen ein gemeinsames Templates Repository gab, wurde pro Anwendung ein eigenes Composition Repository erstellt. Der Grund dafür ist, dass bei den sich überlappenden DB-Ausschnitten und auch bei sich überlappenden Bereichen des jeweiligen Service Interface Unterschiede im Detail aufgetreten waren. Als Beispiel dient hier und in den folgenden Abschnitten die Testanwendung *ChartProduction*.

Im Projekt wurde nach einer Analyse der Anwendung und des verwendeten Schema-Ausschnitts mit der Definition der Datenzugriffsschicht begonnen. Diese setzt sich aus den von der Anwendung verwendeten Tabellen und Views sowie den auf ihnen durchgeführten Selektionen zusammen. Die Testanwendung *ChartProduction* greift nur-lesend auf insgesamt acht Tabellen bzw. Views zu. Im Folgenden wird beispielhaft die Tabelle *WAEHRUNG* und die View *IMPORT\_CURRENCY\_PERFECT* herausgegriffen.

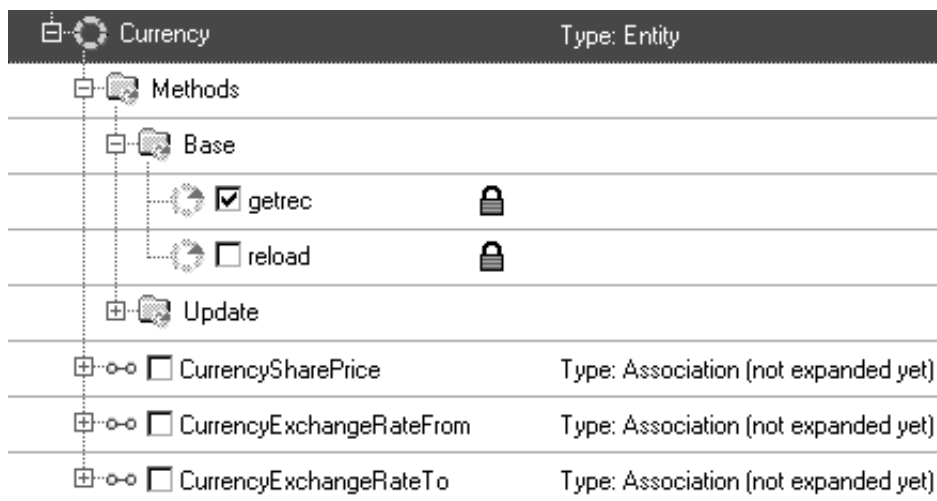


Abbildung 6.1.: Entity-Datenobjekt *Currency*

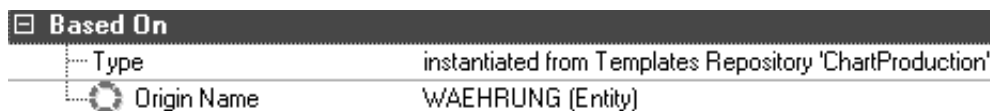


Abbildung 6.2.: Anzeige der „Cross Reference“ zum Entity-Datenobjekt *Currency*

Für jede von *ChartProduction* verwendete View oder Tabelle wurde im SCORE Composition Manager ein Datenobjekt des Typs „Entity“ auf Basis des entsprechenden Templates definiert (s. Abb. 6.1 und Abb. 6.2 sowie Abb. 6.3). Zu jedem Entity-Objekt werden mehrere Basis- und Update-Methoden angeboten, von denen die tatsächlich benötigten auszuwählen sind. In diesem Fall wurden Update-Methoden nicht benötigt, es wurde für alle Entity-Objekte nur der lesende Zugriff per „getrec“ aktiviert (s.

Abb. 6.1 und Abb. 6.3).

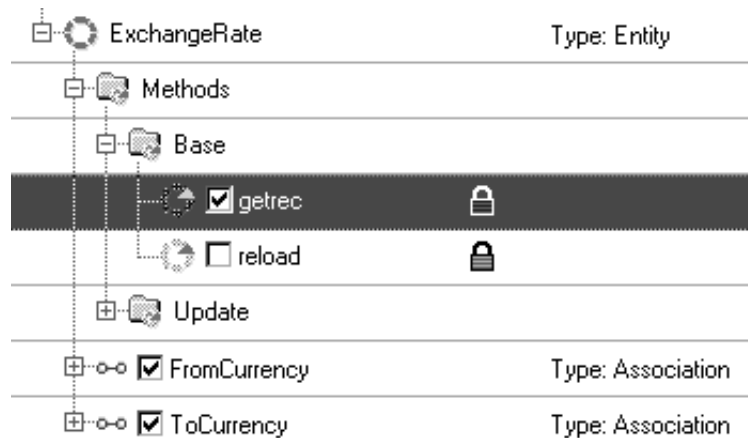


Abbildung 6.3.: Entity-Datenobjekt *ExchangeRate*, abgeleitet aus *IMPORT-CURRENCY\_PERFECT*

Die Methodenparameter der Methode „getrec“ eines Entity-Objekts repräsentieren die Tabellenspalten. Zusätzlich gibt es Parameter für eventuell vorhandene Null-Indikatoren (s. Abb. 6.4).

Record Data Elements		
<input checked="" type="checkbox"/>	WHRNR	Decimal (6)
<input checked="" type="checkbox"/>	WHRNR2	Decimal (6)
<input checked="" type="checkbox"/>	VALUE	Decimal (6)
<input checked="" type="checkbox"/>	VALUE-NI	Indicator
<input type="checkbox"/>	DATUM	Date
<input type="checkbox"/>	DATUM-NI	Indicator

Abbildung 6.4.: Methodenparameter zu *ExchangeRate.getrec*

Die benötigten Parameter werden per CheckBox aktiviert. Das Entity-Datenobjekt für die View *IMPORT-CURRENCY\_PERFECT* benötigt beispielsweise die Spalte *DATUM* nicht. Die Spalte *VALUE* ist als „nullable“ definiert, wird aber benötigt. Die anderen beiden Spalten *WHRNR* und *WHRNR2* sind der Primärschlüssel der View. Entsprechend wurden die Methodenparameter der *getrec*-Methode selektiert (s. Abb. 6.4). Die Methodenparameter besitzen in SCORE DBMS-unabhängige Datentypen. Hier sind, motiviert durch MINT, einige Datentypen hinzugekommen, z. B. „Date“.

Sind Beziehungen („Relationships“) für die Tabellen oder Views definiert worden, stehen zu den betreffenden Entity-Objekten Association-Objekte zur Verfügung. In diesem Beispiel gibt es zwischen der Tabelle *WAEHRUNG* und der View *IMPORT-CURRENCY\_PERFECT* zwei auf Fremdschlüsseln basierende 1:n-Beziehungen, die

jeweils in Richtung des Fremdschlüssel-Zugriffs (also von *ExchangeRate* auf *Currency*) von *ChartProduction* verwendet werden. Die benötigten Association-Objekte wurden aktiviert, s. Abb. 6.5.

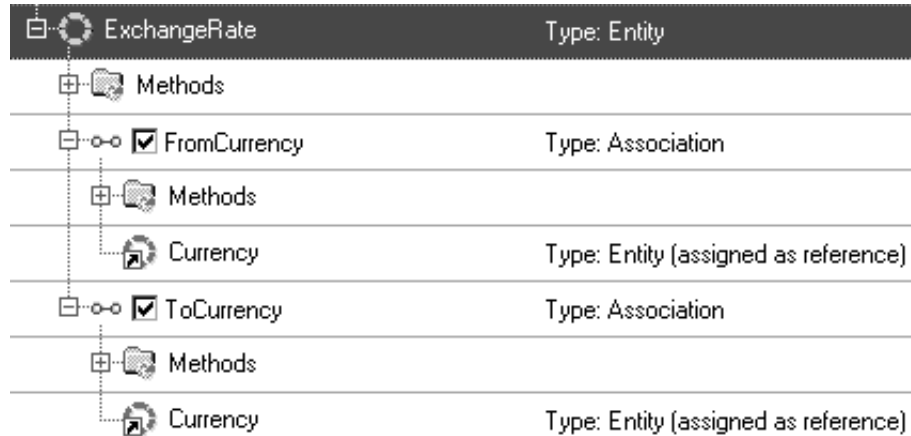


Abbildung 6.5.: Association-Objekte zu *ExchangeRate*

Unterhalb der Association-Objekte befindet sich jeweils ein Referenzobjekt als Platzhalter für das betreffende Entity-Objekt. In diesem Zusammenhang fanden im SCORE Composition Manager mehrere Verbesserungen statt: Neben den Referenzobjekten selbst, die eine bessere Handhabung in den Definitionen erlauben (z. B. bei Mehrfach-Referenzierungen), wurde auch die Definition zyklischer Referenzen eingeführt.

Zu jedem Entity-Objekt gehört in SCORE mindestens ein Objekt des Typs „Navigator“. In diesem Fall wurde genau ein Navigator pro Entity definiert: Ohne Navigator wäre die Entity nicht referenzierbar, und mehr als ein Navigator-Objekt ist nur für nicht objektorientierte Umgebungen interessant, die keine Instantiierung (einer Navigator-Klasse) ermöglichen.

Die Navigator-Objekte enthalten jeweils eine Load-Methode, die den Primärschlüsselzugriff auf das verknüpfte Entity-Objekt ermöglicht (s. Abb. 6.6). Die Parameter der Load-Methode sind genau die Primärschlüssel-Spalten, erweitert um einen Returncode-Parameter (s. Abb. 6.7).

Darüber hinaus kann es in den Navigator-Objekten noch weitere Selektionsmethoden geben, um selektive Zugriffe (mit mehr als einem Resultat) zu ermöglichen. Ein Beispiel ist das auf der Tabelle *BEZUGSINDIZES* basierende Entity-Objekt *StockIndex*. Aus der Tabelle werden drei Spalten benötigt, von denen eine – *IDENTIFIER* – kein Bestandteil des Primärschlüssels ist (s. Abb. 6.8).

Genau für diese Spalte *IDENTIFIER* wird in *ChartProduction* ein selektiver Zugriff durchgeführt. Das entsprechende Navigator-Objekt besitzt die zugehörige Selektionsmethode, s. Abb. 6.9.

Die Spezifikation solcher anwendungsspezifischer Selektionsmethoden, die nicht unmittelbar aus den Templates abgeleitet werden können, schließt die Definition der Datenzugriffsschicht ab.



Abbildung 6.6.: Navigator-Objekt zu *WAEHRUNG*

	Name	Type	Scope / Length
1	WHRNR	<b>Composite</b>	Method
2	WHRNR	Decimal	6
3	retcode	<b>Composite</b>	Method
4	retcode	Integer	

Abbildung 6.7.: Methodenparameter zu *NavCurrency.load*

Record Data Elements		
<input checked="" type="checkbox"/>	BEZEICHNUNG	Varchar (100)
<input checked="" type="checkbox"/>	PROVIDERID	Decimal (6)
<input type="checkbox"/>	ART	Varchar (20)
<input type="checkbox"/>	WHRBASIS	Varchar (20)
<input checked="" type="checkbox"/>	IDENTIFIER	Varchar (20)
<input type="checkbox"/>	ID	Decimal (4)

Abbildung 6.8.: Methodenparameter zu *StockIndex.getrec*

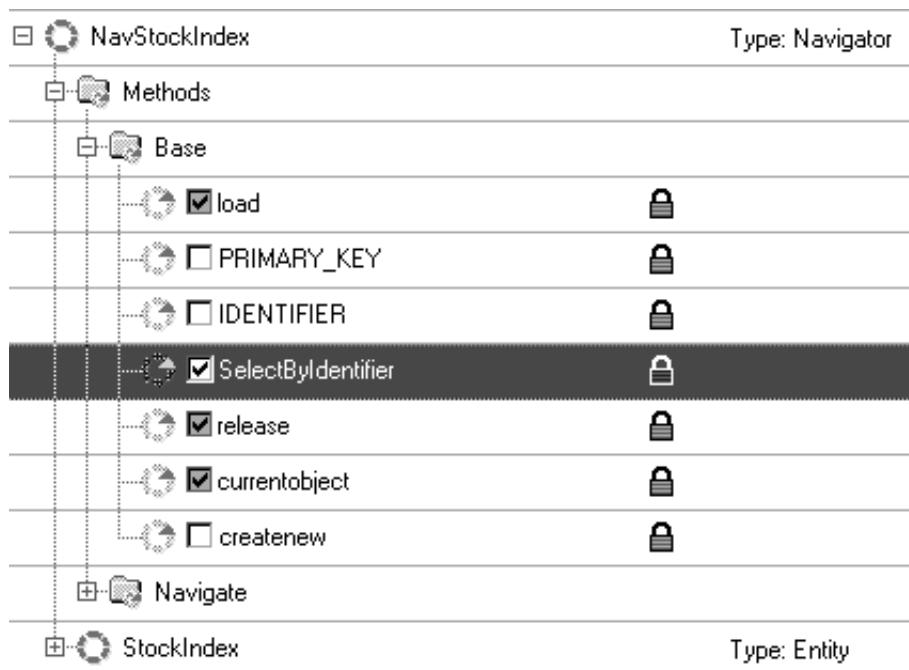


Abbildung 6.9.: Navigator-Objekt zu *BEZUGSINDIZES*



## 6.5. Die Service-Schicht

Die Datenzugriffsschicht isoliert zwar das eingesetzte DBMS und die verwendete Middleware, bietet aber eine rudimentäre, an den DB-Tabellen orientierte, Schnittstelle an. Die Aufgabe der Service-Schicht ist es, die Datenzugriffe in einer Form zu bündeln, die genau auf die Erfordernisse der Anwendung zugeschnitten ist. In unserem Fall geht es darum, eine objektorientierte Sicht auf Daten zu ermöglichen, die in einer relationalen Datenbank verwaltet werden. Zu dieser konzeptionellen, durch unterschiedliche Paradigmen erforderlichen Abstraktion, kommt noch eine semantische hinzu: Spezifisch für die Anwendung sollen die Datenzugriffe gebündelt werden. Dabei sind beide Schichten, die Service- und die Datenzugriffsschicht, spezifisch auf die jeweilige Anwendung zugeschnitten. Die Datenzugriffsschicht implementiert eine technische Abstraktion, die Service-Schicht eine semantische.

⊕ → DAICartProductionApplicationFacade	Type: Standard
⊕ → DAICurrency	Type: Standard
⊕ → DAExchangeRate	Type: Standard
⊕ → DAICompany	Type: Standard
⊕ → DAIShare	Type: Standard
⊕ → DAIShareIdentifier	Type: Standard
⊕ → DAIShareIdentifierType	Type: Standard
⊕ → DAISharePrice	Type: Standard
⊕ → DAIStockIndex	Type: Standard

Abbildung 6.10.: Das Service Interface für *ChartProduction*

Die oberste Ebene der Service-Schicht ist das „Service Interface“. Die Vorlage für das Service Interface sind im Fall von *ChartProduction* neun C#-Interfaces. Das Service Interface besteht aus mehreren Interfaces, die jeweils mehrere Operationen umfassen. In unserem Fall entspricht ein Interface des Service Interface genau einem C#-Interface, und eine Interface-Operation genau einer Methode in einem C#-Interface. Abb. 6.10 zeigt die Interfaces für *ChartProduction*. Eines besitzt eine hervorgehobene Rolle: *DAICartProductionApplicationFacade*. Es ist die zentrale Schnittstelle von *ChartProduction*. Die weiteren Interfaces definieren die Schnittstellen von Objekten, die von der Fassade zurückgeliefert werden – und somit das Service Interface erweitern.

Ein Beispiel für die letztgenannten Interfaces ist *DAICurrency*. Die „Interface Properties“ von *DAICurrency* umfassen mit „Implements“ die Spezifikation des zugehörigen C#-Interface und mit „Extends“ die – eventuell – benötigte Angabe einer Parent-Klasse. In diesem Fall gewährleistet *MarshalByRefObject* den sicheren und ef-

Name	DAICurrency
Description	
<b>Configuration</b>	
Type	Standard
Refers to Object	Currency
Extends	MarshalByRefObject
Implements	ICurrency
<b>Production</b>	
<b>Administration</b>	
Granted to	Composition Manager

Abbildung 6.11.: „Interface Properties“ von *DAICurrency*

fizienten Transport der Instanzen über Prozessgrenzen hinweg. Beide Angaben sind OO-bezogene Neuerungen in SCORE, die im Rahmen von MINT eingebaut wurden.

DAICurrency	Type: Standard
Interface Working Data	
get_Id	Version: 1 Method: getrec
set_Id	Version: 1 Method: setrec
get_Name	Version: 1 Method: getrec
set_Name	Version: 1 Method: setrec

Abbildung 6.12.: Operationen des Interface *DAICurrency*

Das Interface *DAICurrency* besteht aus den Get- und Set-Operationen für *Id* und *Name* (s. Abb. 6.12). Zu den einzelnen Operationsparametern wurden jeweils der Name, der Datenfluss („In“, „Out“, „InOut“ oder „Return“) sowie der Datentyp definiert. Beispielsweise enthält die Operation *DAICurrency.get\_Name* genau einen Return-Parameter des Typs „string“ (s. Abb. 6.13).

	Name	Data Flow	Type	Length
1	Name	Return	Var. Characters	3

Abbildung 6.13.: Parameter der Operation *DAICurrency.get\_Name*

Die meisten Interfaces besitzen Get- und Set-Operationen mit lediglich einem Parameter, während die Operationen des Fassaden-Interfaces (s. Abb. 6.14) komplexer sind. Ein Beispiel ist die Operation *GetExchangeRate*, s. Abb. 6.15. Alle ihre Parameter sind vom Typ eines anderen Interfaces – eine durch MINT motivierte Neuerung in SCORE.

DAIChartProductionApplicationFacade		Type: Standard
Interface Working Data		
getShares	✓	Version: 1 Method:
getSharePrices	✓	Version: 1 Method:
getShareRelativePerformance	✓	Version: 1 Method:
getShareIdentifiers-1	✓	Version: 1 Method:
getShareIdentifierForType	✓	Version: 1 Method:
GetShareIdentifierType	✓	Version: 1 Method:
GetLanguage	✓	Version: 1 Method:
getShareIdentifiers	✓	Version: 1 Method:
GetExchangeRate	✓	Version: 1 Method:
FindSharePriceByShareAndDate	✓	Version: 1 Method:
DAIChartProductionApplicationFacade	✓	Version: 1 Method:

Abbildung 6.14.: Operationen des Interface *DAIChartProductionApplicationFacade*

Ebenfalls im Rahmen von MINT neu eingeführt wurde die Möglichkeit, zusätzlich zu vordefinierten SCORE-Datentypen und existierenden Interfaces auch plattform-spezifische und projektspezifische Datentypen zu definieren. Dies geschieht jeweils per Konfiguration mit einer XML-Datei. Zum Beispiel enthält die Operation *getShares* des Fassaden-Interface mit *IList* einen C#-spezifischen Datentypen (s. Abb. 6.16). Die Operation *get\_Recommendation* des Interface *DAIShare* hingegen liefert ein Beispiel für die Verwendung eines projektspezifischen Datentyps: *IRecommendation* ist ein Interface des Objektmodells, das zwar von *ChartProduction* nicht verwendet wird, jedoch im betreffenden C#-Interface *IShare* definiert ist (s. Abb. 6.17).

Generell sind alle C#-Interfaces des von den Testapplikationen verwendeten Objektmodells sehr stark miteinander verflochten. Um eine applikationsspezifische Definition zu erreichen, wurde die Fassade vollständig implementiert, und die weiteren damit verknüpften Interfaces nur insoweit, wie sie tatsächlich verwendet werden<sup>1</sup>.

<sup>1</sup>Damit verbindet sich eine gewisse Einschränkung bzgl. einer semantischen Abstraktion. Genau genommen müsste man die Operationen auf der Fassade und den von ihr gelieferten Objekten noch stärker bündeln, so dass das Service Interface „oberhalb“ der Fassade angesiedelt wäre. Eingriffe oberhalb der Fassade hätten jedoch die Anwendungen unmittelbar berührt, und wurden daher nicht durchgeführt.




	Name	Data Flow	Type
1	 fromCurrency	→ In	ICurrency
2	 toCurrency	→ In	ICurrency
3	 return	← Return	IExchangeRate

Abbildung 6.15.: Parameter der Operation *DAIChartProductionApplicationFacade.GetExchangeRate*



	Name	Data Flow	Type
1	 return	← Return	IList
2	 shareIdentifierType	→ In	IShareIdentifierType

Abbildung 6.16.: Parameter der Operation *DAIChartProductionApplicationFacade.getShares*


	Name	Data Flow	Type
1	 return	← Return	IRecommendation

Abbildung 6.17.: Parameter der Operation *DAIShare.get\_Recommendation*

### 6.5.1. Mappings

Die Interfaces und Operationen der Service-Schicht sind auf die Objekte und Methoden der Datenzugriffsschicht abzubilden. Die (gewünschte) „semantische Lücke“ zwischen beiden Schichten macht hier eine interaktiv definierte Abbildung erforderlich. Bei diesem „Mapping“ handelt es sich um eine unidirektionale Abbildung, d. h. die Datenobjekte besitzen keinerlei Information über das Service Interface. Umgekehrt sind bei der Mapping-Definition die Datenobjekte sichtbar und referenzierbar. Die Mapping-Definition findet in den Interface-Operationen des Service Interface statt, da diese die kleinste adressierbare Einheit des Service Interface darstellen.

#### Mapping-Regeln

Die Abbildungen von Operationsparametern auf Methodenparameter der Datenzugriffsschicht werden durch Mapping-Regeln beschrieben. Während im allgemeinen Fall SCORE die Definition mehrerer Mapping-Regeln pro Parameter erlaubt, macht dies bei einer objektorientierten Implementierungssprache keinen Sinn. Es genügt jeweils eine Regel.

In einfachen Fällen wird ein Operationsparameter auf einen Parameter einer Methode eines Datenobjekts abgebildet. In diesem Beispiel trifft dies auf die Operation *DAICurrency.get\_Name* zu. Sie besitzt genau einen Parameter. Für diesen ist eine Mapping-Regel definiert worden (s. Abb. 6.18). Dieser Parameter *return* wird als Ganzes auf ein Datenelement eines Datenobjekts abgebildet. Dazu ist für den „Operation Context“ die Einstellung „Data Elements“ ausgewählt worden. Weitere mögliche Einstellungen sind hier „Embedded Coding“ (s.u.) sowie „No Mapping“.

Item	Operation Context	Operation Value	Method Context	Method Value
→ <b>In Mapping</b>				
^ <b>Execute</b>	No method invoked			
← <b>Out Mapping</b>				
⊞ <b>Return</b>				
⊞ Parameter	return			
⊞ Rule	Data Elements		WAEHRUNG	
⊞ Field Map	Native Target	return	Field	WHRKURZBEZ

Abbildung 6.18.: Mapping eines Return-Parameters auf ein Datenelement eines Datenobjekts

Für den „Method Context“ ist im Datenobjekt *Currency* die gewünschte Methode ausgewählt worden. Handelt es sich, wie in diesem Fall, um die *getrec*-Methode, so steht die gesamte Tabelle zur Verfügung (s. Abb. 6.19) und wird als Method Context angezeigt.

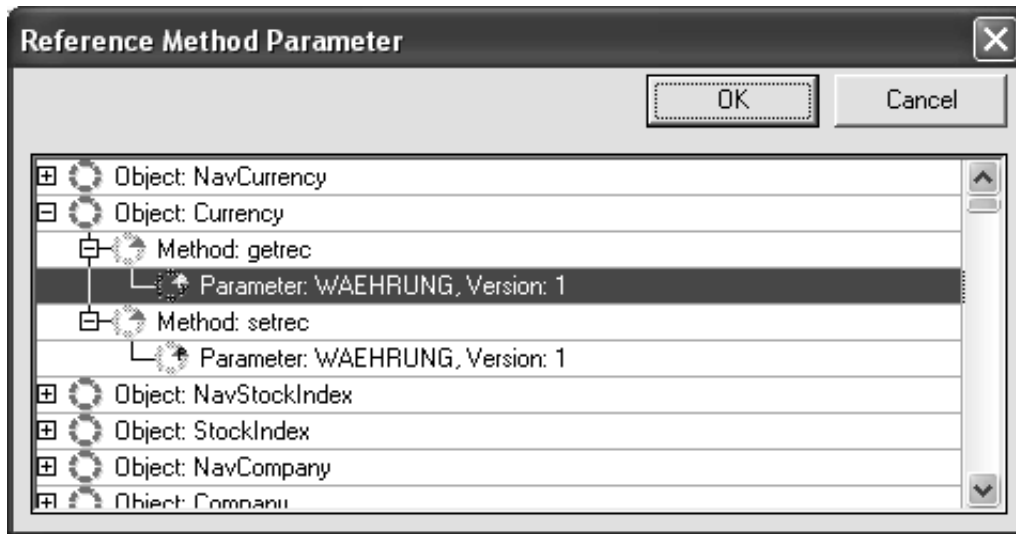


Abbildung 6.19.: Auswahl von Datenobjekt, Methode und Parameter

Wird eine Mapping-Regel mit dem Operation Context „Data Elements“ definiert – wie im Beispiel – so kann zusätzlich eine „Field Map“ definiert werden. Hier besteht die Field Map aus genau einem Ziel- und einem Quell-Feld. Der entsprechende Eintrag im Operation Context und im Method Context lautet daher „Field“. Alternativ kann man dort „Native Target“ bzw. „Native Source“ auswählen, was eine freiere Definition ermöglicht. Zusätzlich gibt es für den Method Context noch die Auswahlmöglichkeiten „Numeric Constant“ sowie „String Constant“, um einen Parameter mit dem Wert einer Konstanten zu versorgen.

Der „Operation Value“ ist der Name des Parameters, wenn es nur ein Feld gibt. Als „Method Value“ kann ein Parameter der Methode ausgewählt werden. Hier stehen die Spalten der Tabelle *WAEHRUNG* zur Verfügung (s. Abb. 6.20).

Method Context	Method Value
WAEHRUNG	
Field	WHRKURZBEZ Var. Charac
	WAEHRUNG WHRNR Packed Decimal (6,0) WHRKURZBEZ Var. Characters

Abbildung 6.20.: Feldauswahl im Methodenparameter

Die Mehrzahl der Interface-Operationen ist auf die hier am Beispiel von *DAICurrency.get\_Name* beschriebene Weise definiert worden. In einigen Fällen – und dies betrifft insbesondere die Operationen der oben genannten Fassade – ist die Abbildung auf die Datenzugriffsschicht so unregelmäßig und komplex, dass eine freiere Definition erforderlich ist.

## Komplexe Mappings

Ein Beispiel für ein komplexes Mapping ist die Operation *GetExchangeRate* aus dem Fassaden-Interface (s. Abb. 6.21).

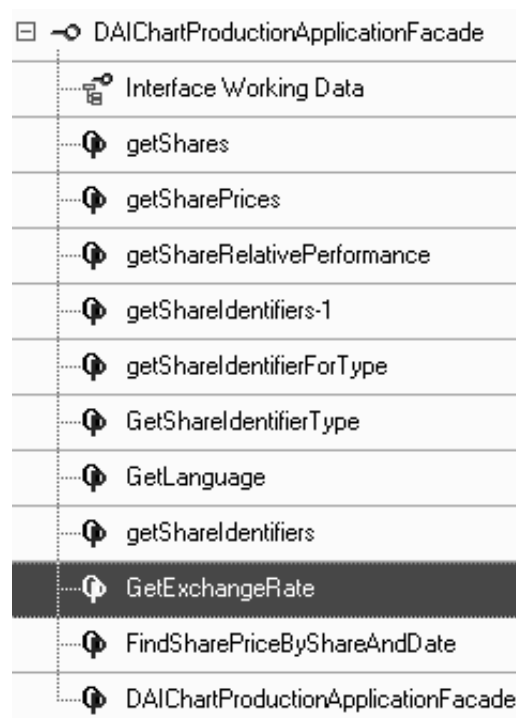


Abbildung 6.21.: Operationen des Interface *DAICurrencyApplicationFacade*

Für diese Operation ist eine „Execute“-Mapping-Regel mit dem Operation Context „Embedded Coding“ definiert worden (s. Abb. 6.22).

Das Embedded Coding wird in einem eigenen Edit-Dialog erfasst (s. Abb. 6.23). Dort ist ein freies Editieren möglich, wobei neben einem Chromacoding für C# insbesondere ein Model Browser das Eingeben von Datenobjekt-bezogenen Elementen unterstützt.

Der vorliegende Fall, bestehend aus dem Aufruf einer Load-Methode einer Navigatorklasse und der Rückgabe des Returnwerts, wobei ein Konstruktor aufgerufen wird, ist ein einfaches Beispiel für Embedded Coding. Die meisten Operationen im Fassaden-Interface sind deutlich umfangreicher.

Operation Parameters   Implementation   Association   Versions   Cross Reference   Checker   Output		
DAIChartProductionApplicationFacade.GetExchangeRate		
<input checked="" type="checkbox"/> Text format		
Item	Operation Context	Operation Value
→ In Mapping		
^ Execute	Embedded Coding	[DAIChartProductionApplicationFacade.GetExchangeRate]
← Out Mapping		
← Return		

Abbildung 6.22.: Mapping-Regel mit Embedded Coding

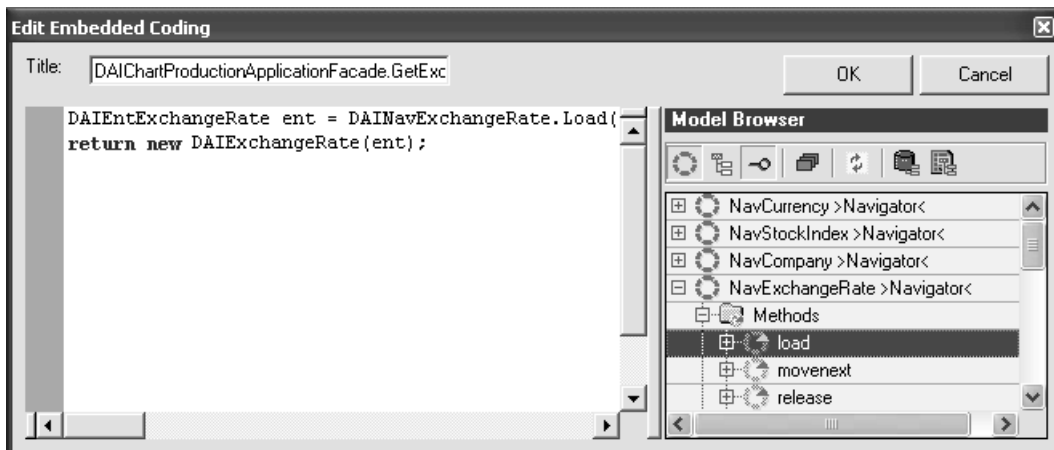


Abbildung 6.23.: Eingabe von Embedded Coding

### Mapping-Patterns

Eine dritte Variante der Mapping-Definition in SCORE stellen Mapping-Patterns dar. Mapping-Patterns ermöglichen die flexible Parametrisierung immer wiederkehrender Codefragmente. Sie bieten jeweils eine eigene Konfigurationsschnittstelle und sind an die Modelldaten des aktuellen Composition Repository gekoppelt. Technisch handelt es sich um jeweils einen Verbund aus 1–n HyperSenses-Patterns (s. Abschnitt 2.5.2). Die beteiligten HyperSenses-Patterns umfassen jeweils mindestens zwei Implementierungen, eine zur Definition der Konfigurationsschnittstelle sowie mindestens eine weitere zur zielplattformspezifischen Codeproduktion.

Für die im Rahmen von MINT untersuchten Beispiele wurden sechs Mapping-Patterns definiert:

- **CollectIntoList** – Selektiver Zugriff mit einer Liste als Resultat
- **GetRelatedToMany** – Zugriff entlang einer definierten Assoziation mit einer Liste als Resultat
- **GetRelatedToOne** – Fremdschlüssel-Zugriff



- **NotImplemented** – Operation besitzt keine Implementierung
- **SelectFirst** – Selektiver Zugriff mit mehreren Ergebniselementen, welcher nur das erste Resultat zurückliefert
- **SetRelatedToOne** – Setzen einer Fremdschlüssel-Beziehung

Im Folgenden werden die – in *ChartProduction* sehr häufig verwendeten – Mapping-Patterns *NotImplemented*, *GetRelatedToOne* und *GetRelatedToMany* genauer vorgestellt.

*NotImplemented* ist ein Pattern für nicht implementierte Operationen eines Service Interface. Im Fall der Zielplattform C# / ADO.NET / Oracle entspricht ein Service Interface einer C#-Klasse. Diese implementiert in den meisten Fällen ein Interface des Objektmodells. Manche Methoden des vorgegebenen C#-Interfaces werden jedoch nicht implementiert – andernfalls müssten in der Lösung aufgrund der Vernetzung der Interfaces des Objektmodells das gesamte Objektmodell implementiert werden, d. h. es gäbe dann keine anwendungsspezifischen Adapter mehr, sondern einen Objektmodell-Adapter. Da dieser Fall häufig auftaucht, macht eine Definition eines *NotImplemented*-Pattern Sinn.

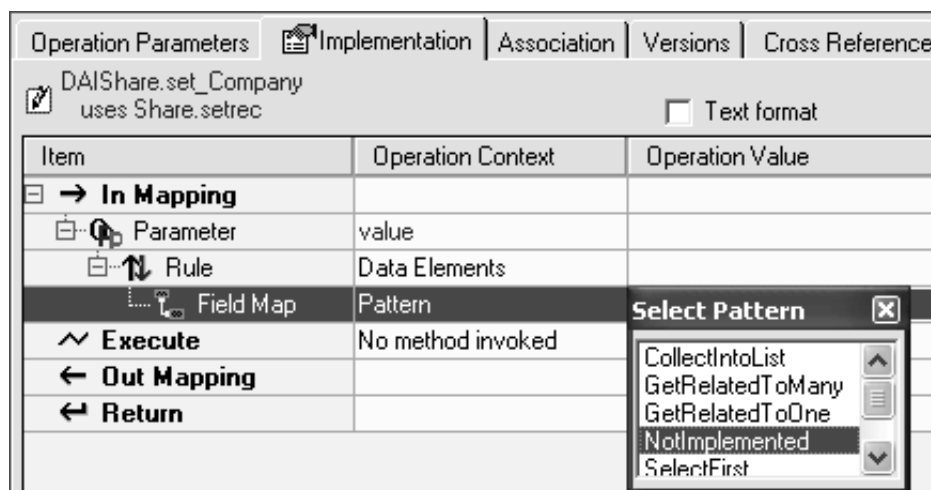


Abbildung 6.24.: Auswahl eines Mapping-Pattern

In der Mapping-Definition der betreffenden Methode erscheint bei der Angabe „Pattern“ als Operation Context eine Auswahlbox mit den zur Verfügung stehenden Mapping-Patterns (s. Abb. 6.24). Wird ein Pattern ausgewählt, erscheint im SCORE Composition Manager ein HyperSenses-Konfigurationsfenster. In unserem Fall gibt es für dieses Pattern jedoch nichts weiter zu konfigurieren – *NotImplemented* besitzt keine Parameter.

Das *NotImplemented*-Pattern dient außer zur Spezifikation der *NotImplemented*-Semantik auch zur Generierung des C#-Codes für das Auslösen einer entsprechenden Exception (s. „set“ in Abb. 6.25).

```

private ICompany company;
public ICompany Company
{
    get
    {
        if (company == null)
        {
            DAIEntCompany entCompany = entity.GetCompany();
            company = new DAICompany(entCompany);
        }
        return company;
    }
    set { throw new NotImplementedException(); }
}

```

Abbildung 6.25.: Property *Company* der C#-Klasse *DAIShare*

Das Mapping-Pattern *GetRelatedToOne* ermöglicht den Zugriff auf eine per Fremdschlüssel erreichbare Instanz eines benachbarten Interfaces. Diese Art des Zugriffs ist ein häufig auftauchender Standardfall. Für das Pattern ist lediglich der Entity-Name (*Company*) zu konfigurieren (s. „get“ in Abb. 6.25). Diese Konfiguration erfolgt interaktiv durch Popup-Menüs, die die Datenobjekte anzeigen (s. Abb. 6.26).

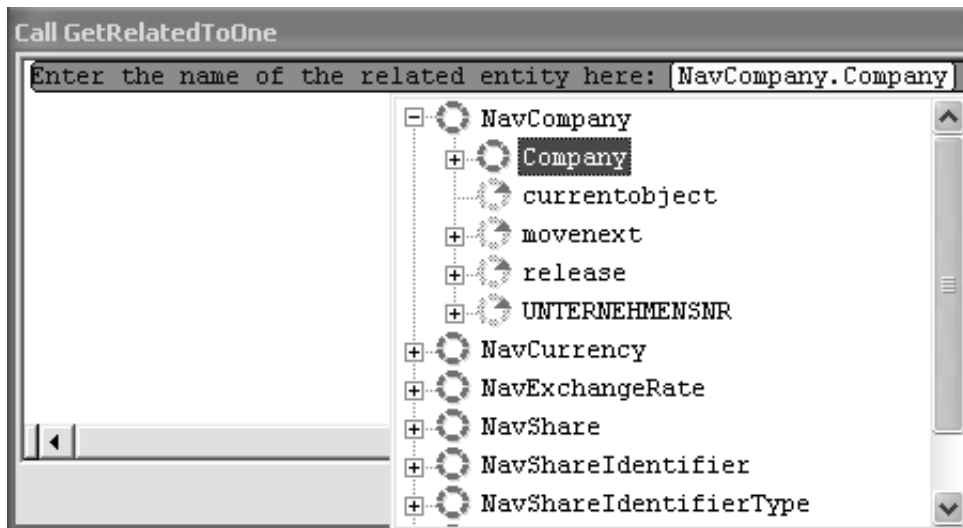
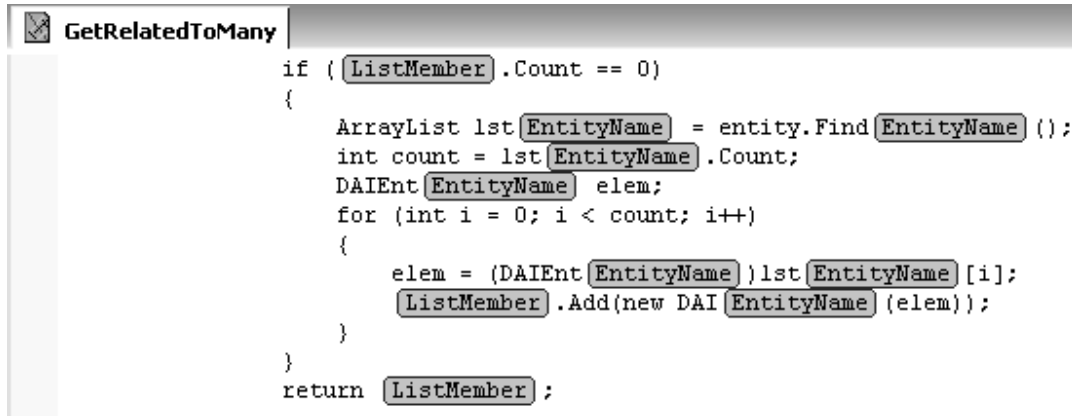


Abbildung 6.26.: Selektion des Entity-Namens im Mapping-Pattern *GetRelatedToOne*

Das Mapping-Pattern *GetRelatedToMany* schließlich realisiert eine typische Navigation „in Gegenrichtung“ bzw. in der zu-n-Richtung einer Relation bzw. Assoziation. Auch dieser Fall taucht mehrfach auf und ist ein Standardfall. An Stelle des Quellco-

des werfen wir hier einen Blick auf die C#-Implementierung des Patterns, also in die Definition des Patterns selbst (Abb. 6.27).



```

GetRelatedToMany
if ( ListMember .Count == 0)
{
    ArrayList lst EntityName = entity.Find EntityName ();
    int count = lst EntityName .Count;
    DAIEnt EntityName elem;
    for (int i = 0; i < count; i++)
    {
        elem = (DAIEnt EntityName )lst EntityName [i];
        ListMember .Add(new DAI EntityName (elem));
    }
}
return ListMember ;

```

Abbildung 6.27.: Produktions-Implementierung des Patterns *GetRelatedToMany* im Pattern-Editor

Hier gibt es zwei Slots (Parameter), *EntityName* und *ListMember*. Letzterer bezeichnet die Klassenvariable des Typs *IList*, die als Property definiert worden ist. *EntityName* bezeichnet das Ziel-Entity der Relation.

Hervorzuheben ist, dass Mapping-Patterns eine modellgebundene, kontextbezogene und domänenspezifische Konfiguration erlauben. Damit bieten sie, neben systematischer Wiederverwendung, entscheidende Vorteile gegenüber Mapping-Definitionen per Embedded Coding.

## 6.6. Adapter-Generierung

Der existierende Application Adapter-Generator wurde im Rahmen von MINT für die Zielplattform C# / ADO.NET / Oracle erweitert. Konkret betraf dies die Erstellung von HyperSenses-Code-Patterns. Neben neuen Patterns wurde zu existierenden Code-Patterns je eine Implementierung für die neue Zielplattform hinzugefügt. Diese Erweiterungen wurden nach der Pattern By Example-Methode (s. Abschnitt 2.5.2) durchgeführt, auf der Basis der bei DSTG manuell erstellten prototypischen Anwendungsadapter für die drei Testanwendungen.

Die Abbildung 6.28 zeigt eine Pattern-Implementierung, welche Routinen für die Navigation entlang der definierten Assoziationen generiert. Der Ausschnitt zeigt den optionalen Codeblock (s. 2.5.2) für die Fremdschlüssel-Zugriffsmethode. Dieser wird durch sieben Slots parametrisiert, von denen sechs konkrete Strings aufnehmen, während der gelb hinterlegte Slot einen Sub-Pattern-Aufruf, einen „Pattern Call“, darstellt. Aus den getesteten und lauffähigen Prototypen wurden Code-Fragmente extrahiert und mit Slot- und Block-Definitionen versehen, um solche neuen Pattern-Implementierungen zu erhalten.

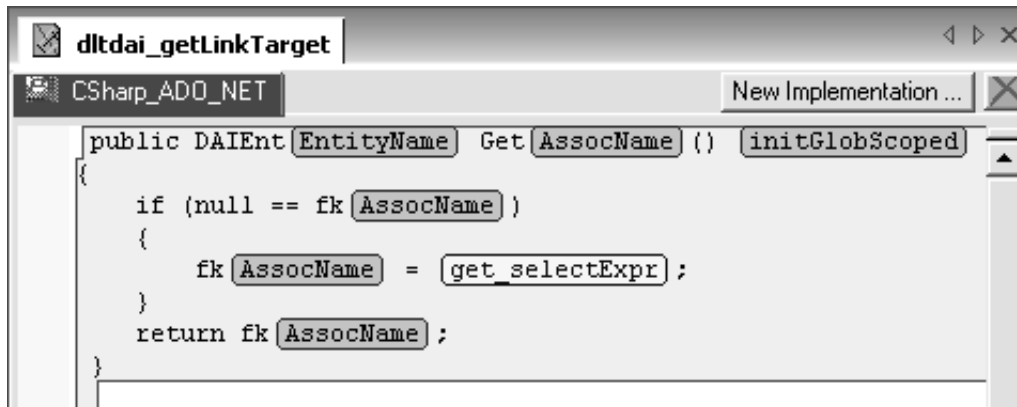


Abbildung 6.28.: Ausschnitt einer Pattern-Implementierung für den Fremdschlüssel-Zugriff

Die Erweiterungen in den Code-Patterns machten Anpassungen sowohl in der Technologie als auch bzgl. der Konfigurationsdaten notwendig:

- Die eingesetzte HyperSenses-Technologie hat zahlreiche, durch das MINT-Projekt motivierte, Erweiterungen erfahren: Es gibt eine neue Struktur der internen MOF-Implementierung, und es gibt neue Konzepte wie die Pattern-Vererbung.
- Für das während der Adapter-Generierung automatisch erzeugte Extract Model (s. Abschnitt 2.5.1) wurden in Bezug auf MINT inhaltliche Erweiterungen implementiert.

### 6.6.1. Der generierte Zielcode

Der generierte Zielcode für die neue Plattform C# / ADO.NET / Oracle besteht aus Zugriffsklassen und Adapterklassen. Die Zugriffsklassen wurden aus den Datenobjekten generiert, während die Adapterklassen aus Service Interfaces und Mapping-Definitionen erzeugt wurden.

#### Zugriffsklassen

Aus den Entity-Objekten des Composition Repository werden Entity-Klassen generiert. Eine Instanz stellt einen Datensatz dar. Eine Entity-Klasse implementiert jede (von der Anwendung verwendete) Tabellenspalte als C#-Property. Eine Ausnahme sind hier Null-Indikatoren: Sie werden nicht als eigene C#-Properties in den Entity-Klassen angeboten<sup>2</sup>. Jede Entity-Klasse enthält ADO.NET-Zugriffe und ist von einer gemeinsamen Basis-Entity-Klasse abgeleitet. „Identity Maps“ in den Entity-Klassen stellen sicher, dass keine Inkonsistenzen durch Duplikate entstehen.

---

<sup>2</sup>ADO.NET bietet hier einen System-DB-Nullwert an, so dass eigene Abfrage-Properties für Nullwerte nicht benötigt werden.

Aus den Navigator-Objekten des Composition Repository werden Navigator-Klassen generiert. Eine Instanz steht für einen Selektions- und Iterationsvorgang und entspricht einem Cursor. Es gibt eine Load-Methode für den Primärschlüsselzugriff, sowie 0-n Selektionsmethoden. Zusätzlich gibt es Navigationsmethoden für Fremdschlüssel und, falls spezifiziert, für die Navigation in der Gegenrichtung. Die Navigator-Klassen basieren auf den Entity-Klassen. Sie instantiiieren sie, und liefern Entity-Instanzen ggf. als Ergebnis zurück. Auch in den Navigator-Klassen sind ADO.NET-Zugriffe enthalten, und sie sind von einer gemeinsamen Navigator-Basisklasse abgeleitet.

Die Navigation in den Zugriffsklassen erfolgt entsprechend dem SCORE-Datenmodell (s. Abb. 2.12). Assoziationen werden nicht als eigene Klassen implementiert, sondern in Form von Zugriffsmethoden in den Entity-Klassen. Dabei wurde insbesondere „Lazy Loading“ implementiert. Auch Creator-Objekte werden als entsprechende Methoden in den Entity-Klassen realisiert. Außer Entity- und Navigator-Klassen gehört zu den Zugriffsklassen noch eine Session-Klasse, welche Connection- und Transaktionseigenschaften implementiert. Auch die Session-Klasse enthält sowohl ADO.NET- als auch Oracle-spezifischen Code.

Insgesamt orientieren sich die Zugriffsklassen inhaltlich und in ihrer Struktur an dem verwendeten Ausschnitt des DB-Schemas. Hier spielen Performance-Optimierungen eine große Rolle. Beispielsweise wird immer nur auf die tatsächlich benötigten Tabellenspalten explizit zugegriffen, ein „SELECT \*“ gibt es nicht. Ein weiterer Performance-Aspekt ist die Generierung von DBMS-spezifischem Code in den Zugriffsklassen, insbesondere durch die Verwendung Oracle-spezifischer ADO.NET-Klassenbezeichner. Ein Beispiel ist die Verwendung von *OracleConnection* in der Session-Klasse, was ein automatisches Connection Pooling ermöglicht. Die Zugriffsklassen kapseln im Anwendungsadapter die plattformspezifische Zugriffstechnologie: Oracle- und ADO.NET-spezifischer Code findet sich im Adapter nur dort.

## Adapterklassen

Die Adapterklassen repräsentieren den von der jeweiligen Anwendung verwendeten Ausschnitt des Objektmodells. Konkret wird für jedes verwendete Interface des Objektmodells – dies sind genau die Interfaces des Service Interface – eine implementierende Klasse generiert. Die Methoden dieser Klasse enthalten Aufrufe der Zugriffsklassen, entsprechend den Mapping-Definitionen. Die Adapterklassen isolieren die Anwendung von der Zugriffsschicht, d. h. die Anwendung „sieht“ nur die Adapterklassen.

Die Adapterklassen besitzen Referenzen auf Entity-Klassen, d. h. eine Adapter-Instanz ist an 0–n Entity-Instanzen gekoppelt. Referenzen auf Navigator-Klassen sind hingegen temporärer Natur, d.h. sie werden nur innerhalb einer Methode für die Dauer einer Iteration verwendet. Eine Sonderrolle nimmt die Session-Klasse ein, die in mindestens einer Adapter-Klasse bekannt sein muss, um die Session bei den Navigationen korrekt anzusprechen. Im Fall von *ChartProduction* besitzen die Adapterklassen jeweils eine Referenz auf eine Entity-Klasse. Die einzige Ausnahme ist die Fassadenklasse, welche eine Referenz auf die Session-Klasse enthält.

## 6.7. Zusammenfassung

Die von DSTG realisierte Lösung, die Persistenzadapter zu generieren, ist maßgeschneidert für die jeweilige Applikation und spezifisch für den verwendeten Datenbankausschnitt und das eingesetzte DBMS. Dies ermöglicht eine hohe Performance (s. Kap. 7.3, insbes. Tab. 7.7), und geht einher mit einer hohen Zuverlässigkeit des generierten Quellcodes sowie einem drastisch reduzierten Testaufwand. Die durchgehende Werkzeugunterstützung senkt den mit dem (genauer: jedem) Werkzeugeinsatz verbundenen Overhead – und damit die Entwicklungskosten – entscheidend ab. Der Break-Even wird um so eher erreicht, je größer das Projekt ist (vgl. dazu Tab. 8.2 in Kap. 8.2).

Hervorzuheben ist, dass während der Adapter-Erstellung mit SCORE sowohl das Objektmodell als auch die Testapplikationen unverändert blieben. Die generierten Anwendungsadapter wurden im Projekt vollständig aus den Daten des Composition Repository generiert. Der generierte Code wurde in einer (menschlich) gut lesbaren Form erzeugt. Obwohl technisch nicht notwendig, ermöglicht DSTG auf diese Weise seinen Kunden eine geringere Abhängigkeit vom Einsatz von Codegeneratoren.

Für DSTG ergibt sich aus den Erweiterungen im Produkt SCORE mit Schwerpunkt auf C# / ADO.NET / Oracle eine Kompetenzerweiterung: Zum einen wird eine Positionierung im Kontext OR-Mapping ermöglicht, zum anderen unterstreicht das MINT-Projekt die Verwendbarkeit von SCORE für unterschiedlichste Umgebungen. Dies zeigt insbesondere die Breite des potentiellen Einsatzbereiches sowie der zugrundeliegenden Konzepte.

# 7. Evaluierung der Ergebnisse

Thomas Goldschmidt

Die Aufgabenstellung, die für Arbeitspaket 3 (AP3) definiert wurde, befasst sich mit der Validierung der gewonnenen Ergebnisse. Hierfür soll eine Testumgebung zur Verfügung gestellt werden (siehe Abschnitt 2.8). Weiterhin soll ein Vergleich verschiedener Kopplungstechniken durchgeführt und der MINT Ansatz mit bestehenden Kopplungstechniken verglichen werden. Als Ergebnisse hiervon sollen Best Practices, Antipatterns und ein Verfahren zur Unterstützung von Entwurfsentscheidungen herausgearbeitet werden. Diese werden später in Kapitel 8 beschrieben. Die Ergebnisse der Evaluierung wurden in den folgenden internationalen Publikationen veröffentlicht: [26], [25].

## 7.1. Das Vorgehen

In diesem Abschnitt soll das Vorgehen beschrieben werden, das zu den Ergebnissen geführt hat. Das Vorgehen wird in Abbildung 7.1 skizziert. Die Ausgangssituation lässt sich folgendermaßen beschreiben.

Es gibt eine seit ungefähr einer Dekade existierende Applikationssuite (*MESCOR*) und eine ebenso alte, dazugehörige Datenbank. Eine erste Analyse zeigte, dass die Datenbank eher suboptimal gestaltet wurde. Es wurde an vielen Stellen nicht normalisiert und in einigen Tabellen liegt keine Primärschlüsseldefinition vor. Dazu gehören zwei Serverapplikationen, die Anfragen der Clients entgegennehmen, evtl. modifizieren und danach die Kommunikation mit der Datenbank durchführen. “Oben” im Schichtenmodell gibt es diverse Clientapplikationen, die für viele Bereiche des Wertpapiergeschäftes genutzt werden können. Aus dieser Menge von Applikationen wurde die Testumgebung extrahiert (siehe Abschnitt 2.8). Die vorhandene Serverschicht wurde nicht mit in die Testumgebung miteinbezogen, stattdessen wird diese umgangen und es wird direkt mit der Datenbank kommuniziert.

Zu Beginn wurde eine Migration der Chartproduction durchgeführt. Diese Applikation wurde in der Vergangenheit mit Borland Delphi entwickelt und nun nach Microsoft .NET migriert. Anhand der Datenbank wurde per *bottom up* Analyse ein wertpapierorientiertes Objektmodell erstellt. Eine zentrale Applikation in der *MESCOR* Applikationssuite ist das sogenannte *Accounting*. Diese Applikation ermöglicht es einem Analysten Informationen über Wertpapiere zu sammeln und Empfehlungen auszusprechen. Diese Applikation wurde durchanalysiert und daraus ein allgemeines wertpapier- und unternehmensorientiertes Objektmodell abgeleitet (siehe: 7.4 und 7.3). Diese Analyse

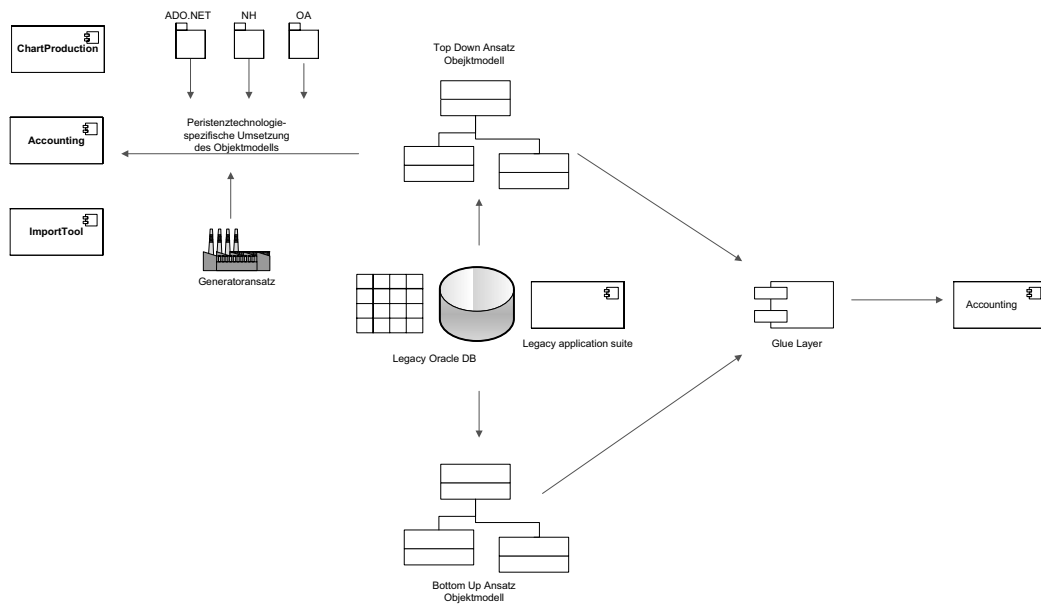


Abbildung 7.1.: Schematische Darstellung - das Vorgehen

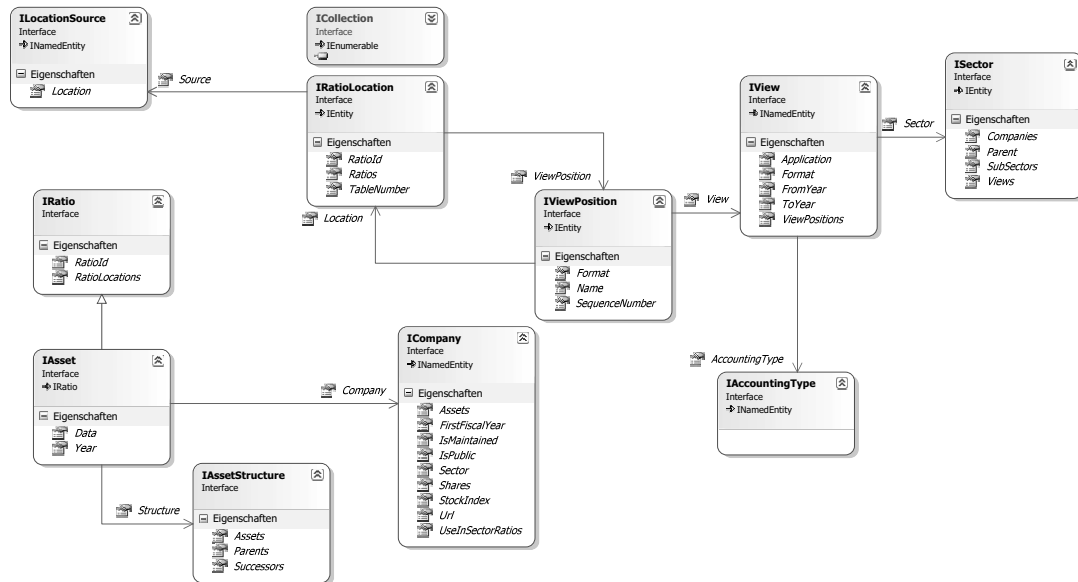


Abbildung 7.2.: Accounting - Objektmodell



erfolgte nicht wie zuvor bottom up, sondern top down, also von der Applikationsseite her betrachtet. Nachdem das Objektmodell erstellt wurde, ging man nochmals auf die Chartproduction und auf die einzelnen Persistenztechnologien ein. Für jede Persistenztechnologie wurde ein spezifisches Objektmodell abgeleitet. Es wurde auch der Versuch unternommen, das Objektmodell für die Chartproduction durch einen Generator zu erzeugen. Dies konnte erfolgreich mit openArchitectureWare (OAW) durchgeführt werden. Weiterhin wurden diverse andere Ansätze zur modellgetriebenen Entwicklung evaluiert. Einige von ihnen (AndroMDA<sup>1</sup>, Constructor MDRAD<sup>2</sup>) lieferten nicht die erhofften Ergebnisse, da entweder ihr Entwicklungsparadigma nicht für die Integration von Altanwendungen geeignet ist bzw. sie dafür nicht genug Flexibilität boten. Gerade bei der Anpassung auf ein vorhandenes Datenbank-Schema, welches nicht immer perfekt normalisiert wird, erwies sich AndroMDA als nicht flexibel und anpassbar genug. Der Ansatz von Constructor MDRAD sah zwar zunächst recht vielversprechend aus, jedoch wurde die Integration in die Testumgebung dadurch stark erschwert, dass hier noch zusätzlich ein manuell geschriebener Zusatzadapter von Nöten war um die benötigten Interfaces anbieten zu können. Als alternativer Ansatz zur Generatorentwicklung wurde auch der von Delta Software Technologies entwickelte Generator HyperSenses getestet. Dieser bot gerade durch seinen Pattern-By-Example Entwicklungsansatz 2.5.2 Vorteile bezüglich der initialen Entwicklung der Templates. Da sich der Generator jedoch zum Zeitpunkt der Evaluierung noch in der Entwicklungsphase befand wurde diese Evaluierung ausser Konkurrenz durchgeführt. Um den Vergleich mit existierenden Ansätzen herzustellen, wurde der von Delta Software Technology im MINT-Kontext entwickelte Ansatz SCORE (siehe Kapitel 6) ebenfalls mit in die Evaluierung mit einbezogen.

Bei der Analyse der Accounting-Applikation zeigte sich, dass der Stammdatenverwaltungsteil der Applikation sich hervorragend für eine Aufnahme in die Testumgebung eignet, da hier sowohl Lese- als auch Schreibzugriffe auf die Datenbank zum Einsatz kommen. Weiterhin kann hier der Mehrfachzugriff getestet werden. Ausserdem werden hierbei für eine Unit of Work mehrere Daten verändert und diese nicht einzeln, sondern nach Betätigen des Speichern-Knopfes, in einer Transaktion, in die Datenbank persistiert. Für die Accounting-Applikation wurde, genauso wie bei der ChartProduction-Applikation, auch jeweils ein persistenztechnologie-spezifisches Objektmodell erstellt und implementiert, so dass die Applikation mit drei verschiedenen Persistenztechnologien auf die Datenbank zugreifen kann. Ebenso wie für die Chartproduction, wurde auch das für die Accounting-Applikation benutzte Objektmodell mit Hilfe eines Generators erzeugt. Als letzte Applikation wurde das Importtool umgesetzt. Hierbei liegt der Fokus darauf, möglichst performant Daten für die Chartproduction in die DB zu schreiben. Das Lesen kann in diesem Kontext vernachlässigt werden. Der Import in die Datenbank erfolgt mittels XML-Daten. Auch hierfür wurde eine Implementierung des Objektmodells von Hand mit ADO.NET und mit den Persistenzframeworks durchgeführt. Für die Stammdatenverwaltung (Accounting) wurde auch ein Double-Mapping

---

<sup>1</sup><http://www.andromda.org/>, letzter Zugriff: 15.12.2008

<sup>2</sup><http://www.i3design.co.uk/constructor/mdrad>, letzter Zugriff: 15.12.2008

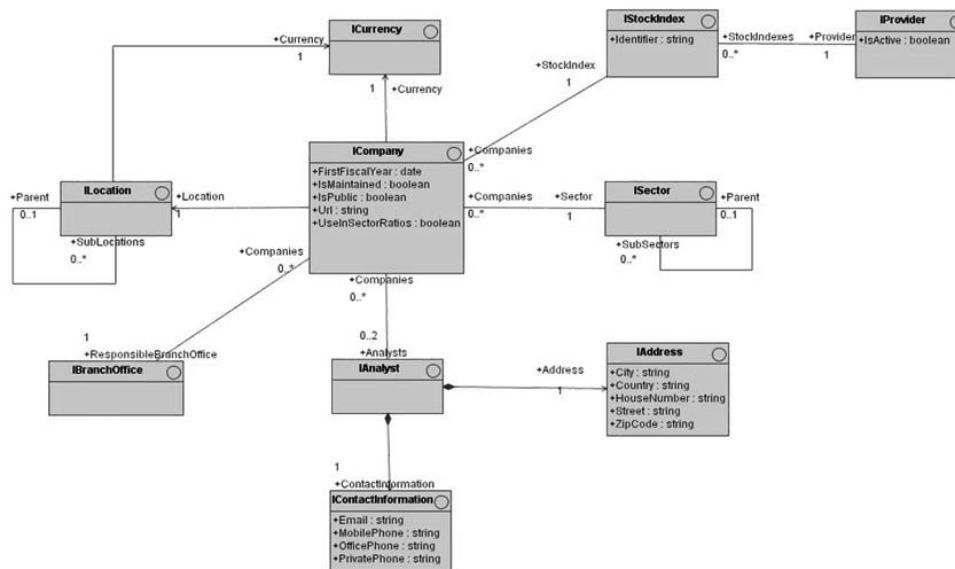


Abbildung 7.3.: Ausschnitt aus Objektmodell - unternehmenszentriert

Ansatz umgesetzt. Hierbei wurde bottom up ein Persistenzmodell erstellt und dieses mittels einer Zwischenschicht (glue layer) mit dem top down entworfenen Objektmodell (BOM) verbunden.

**Einsatz von Remoting** Um ein weiteres *Real World Szenario* nachzustellen, wurden zwei der im Projektkontext betrachteten Applikationen als verteilte Anwendung konzipiert. Die Applikation Chartproduction und Stammdatenverwaltung (Accounting) wurde für die Verwendung mit .NET Remoting implementiert. Hierzu wurde per Persistenztechnologie-spezifischem Remotingserver eine Session angeboten, die von der Applikation zur Kommunikation benutzt werden kann. Bei der nicht-Remoting Implementierung kann man in der Konfigurationsdatei *app.config* festlegen, welche Variante (ADO.NET, NHibernate, OpenAccess, ScoreDAI) zur Kommunikation mit der Datenbank benutzt werden soll. Bei diesem Vorgehen gab es eine Projektreferenz in Form einer .DLL-Datei. Im Gegensatz dazu läuft der spezifische Remotingserver als eigenständige Applikation. In diesem Bereich werden Messungen bzgl. des Overheads, der durch das Serialisieren erzeugt wird, durchgeführt und einem Vergleich unterzogen.

## 7.2. Der Goal Question Metric (GQM)-Plan

Ein Ansatz zur Realisierung eines Messprozesses ist durch das Goal-Question-Metric (GQM)-Paradigma [3] gegeben. Der Ansatz ist so aufgebaut, dass die Planung des Messprozesses in strukturierter Weise Metriken erstellt, die bezüglich ihrer Ziele interpretiert werden. GQM besteht aus einer konzeptionellen, einer operationalen und einer quantitativen Ebene, die das Grundkonstrukt zur Erstellung des Messprozesses

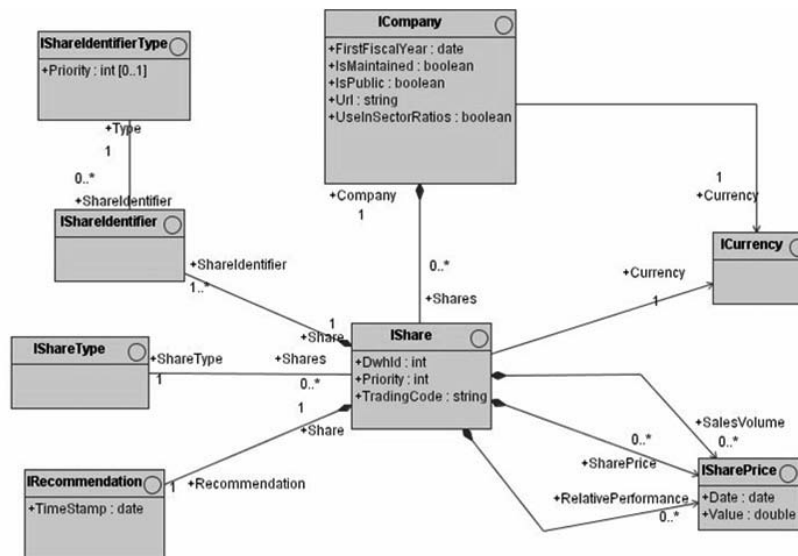


Abbildung 7.4.: Ausschnitt aus Objekmodell - wertpapierzentriert

darstellen. In der konzeptionellen Ebene werden Ziele (Goals) definiert. Sie sind durch die Fokussierung auf einen definierten Sachverhalt oder Angelegenheit charakterisiert. Weiterhin wird festgelegt, welche Objekte betrachtet werden. Dies können Produkte, Prozesse oder Ressourcen sein. Die Perspektive, aus der das Ziel betrachtet wird, ist weiterer Bestandteil des Ziels. Mögliche Perspektiven können beispielsweise Kunden, Entwickler oder Führungspersonal sein. Der Zweck, zu dem das Ziel definiert wird, ist die letzte Eigenschaft eines Ziels. Auf operationaler Ebene werden zu den Zielen Fragen (Questions) erstellt, die einzelne Qualitätsaspekte eines Objektes heraus greifen und es charakterisieren, wobei Fragen natürlichsprachlich formuliert sind. Die letzte Ebene beantwortet die erstellten Fragen durch Metriken (Metric) quantitativ. Die erhaltenen Maßzahlen können ausschließlich von dem Objekt abhängen oder zusätzlicher subjektiver Beeinflussung unterliegen. Grundsätzlich können einzelne Ziele durch mehrere Fragen adressiert werden, die wiederum durch multiple Metriken beantwortet werden. Metriken können zur Beantwortung mehrerer Fragen herangezogen werden und sind ihnen nicht fest zugeordnet. Abbildung 7.5 verdeutlicht diesen Zusammenhang für zwei Ziele. Im GQM-Ansatz sind Metriken integraler Bestandteil, was dazu führt, dass Metriken anhand von Zielen erstellt und nicht willkürlich definiert werden. Das reduziert die Datenmenge der Messwerte und bindet sie gleichzeitig an deren Interpretation.

Speziell im Kontext der in MINT durchgeführten Evaluierung bietet die GQM-Methode Vorteile, da hier speziell für eine Domäne (Erstellung einer Datenanbindungsschicht) ganz gezielt Metriken ausgewählt und deren Einfluss auf die zu evaluierenden Ziele klar dargestellt werden kann. Die spezifischen Qualitätseigenschaften Performance und Wartbarkeit dieser Domäne werden von den folgenden zwei Teilen des erstellten

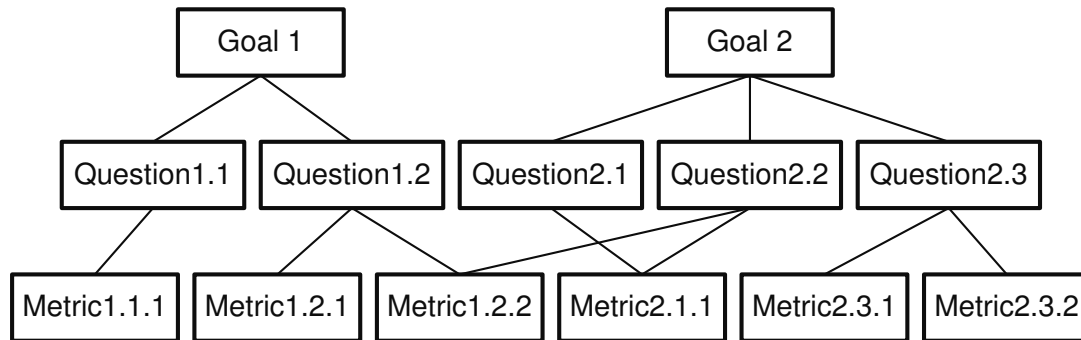


Abbildung 7.5.: Die Struktur eines GQM-Planes (von Basili[3])

GQM-Plans abgedeckt.

### 7.2.1. GQM-Plan für Wartbarkeit

Die Wartbarkeit eines Softwaresystems ist mit empirischen Mitteln durch die vielen schwankenden Einflussfaktoren (so wie Erfahrung der Entwickler, Unterschiede im Entwicklungsparadigma, usw.) nur schwer zu messen. Dies ist der Hauptgrund warum es so schwierig ist analytische Metriken zu validieren. Im Gegensatz zu anderen Wartbarkeitsevaluierungen, welche auf einer direkten Analyse des Source Codes basieren (wie z. B. in [51]) muss eine Evaluierung, welche unterschiedliche Entwicklungsparadigmen, wie in diesem Fall modellgetriebene und herkömmliche manuelle Entwicklung, miteinander vergleicht, auf abstraktere Metriken zurückgreifen. Wartungsaktionen auf Modellen können gegenüber solchen Aktionen auf Quellcode nicht über Metriken verglichen werden welche nur auf Quellcode bezogen sind. In einer modellgetriebenen Entwicklungsumgebung werden Wartungsaktionen hauptsächlich entweder durch Änderungen am Applikationsmodell oder in manchen Fällen durch Änderungen am Generator durchgeführt. Andererseits werden Änderungen in nicht modellgetriebenen Umgebungen durch direkte Änderungen am Quellcode durchgeführt. Da in modellgetriebenen Umgebungen der Code generiert wird, hat die Komplexität dieses Codes keinen Einfluss auf die Wartbarkeit. Daher können Metriken wie Codekomplexität oder Größe nicht sinnvoll zum Vergleich herangezogen werden.

Eine Lösung, die einen Vergleich ermöglicht, ist es daher projektspezifische, empirische Metriken zu definieren. Solche Metriken können mit Hilfe der GQM-Methode durch die Erstellung von Fragen, welche spezifische Wartungsszenarien und Aufgaben die in diesem speziellen Projekt gefunden werden können, definiert werden. Der größte Nachteil dieser Metriken besteht jedoch darin, dass sie natürlich dann auch nur für die speziellen Ziele dieses Evaluierungsplans valide Ergebnisse liefern.

Um die Wartbarkeit von Persistenztechniken zu messen wurde folgendes Ziel definiert:

**Goal 1: Purpose:** Vergleich

**Object:** Verschiedene Persistenztechniken

**Issue:** Wartbarkeit

**Viewpoints:** Das Entwicklungs- bzw. Wartungsteam

Folgende Fragen wurden erarbeitet um dieses Ziel abzudecken:

Ein wichtiger Aspekt einer solchen Technologie ist der initiale Aufwand, der investiert werden muss um dessen Konzepte und die Syntax zu verstehen sowie die Steilheit der Lernkurve um diese Erkenntnisse zu erlangen.

**Question 1.1:** Wie groß ist der initiale Aufwand um die Technologie zu verstehen sowie ein Design zu erstellen wie diese Technologie in das bestehende System integriert werden kann?

Um diese Fragen zu beantworten wurden folgende Metriken aufgestellt:

**M1.1.1:** *Personentage* um die Abbildung für eine spezielle Applikation (Chartproduction) zu entwerfen und zu implementieren. Die Zeit, die gebraucht wird um die jeweilige Technologie zu erlernen und initiale Arbeiten, so wie das Erstellen eines Generators für einen modellgetriebenen Ansatz, sind hier ebenfalls mit einzubeziehen.

**M1.1.2:** *Die Anzahl unterschiedlicher Aspekte*, welche von Hand implementiert werden mussten (so wie z. B. Transaktionen, Caching, Queries oder referentielle Integrität). Einige dieser Aspekte wurden durch die Analyse der Patterns, die bei der Implementierung eines Persistenzadapters nur mit Hilfe einer Standard API (wie z. B. ADO.net in unserem Falle) gebraucht werden, definiert. Für eine vollständige Erklärung der untersuchten Patterns siehe [16]. Die evaluierten Aspekte waren im einzelnen:

1. O/R mapping (Row Data Gateway, Active Record oder Data Mapper)
2. Lazy loading
3. Referential integrity
4. Identity map
5. Unit of work

**M1.1.3:** *Die Anzahl der Workarounds*, die benötigt wurden um eine Technologie in das System vollständig zu integrieren.

**M1.1.4:** Der Zeitbedarf, gemessen in *Personentagen*, welcher aufgebracht wurde um diese Workarounds in das System zu implementieren.

Da unsere Beispielapplikation aus mehreren unabhängigen Teilen besteht, wurde der initiale Entwicklungsaufwand für jede dieser Applikationen separat gemessen. Dies sollte auch Aufschluss über die Steilheit der Lernkurve für die jeweiligen Technologien geben.

**Question 1.2:** Wie hoch ist der Aufwand um weitere Teile des Systems zu implementieren?

**M1.2.1:** Anzahl der *Personentage* die aufgebracht werden mussten um die Abbildungen der jeweiligen Applikation zu implementieren.

**M1.2.2:** Zeit (in *Stunden*) die für das Testen und Debuggen der Applikationen gebraucht wurde.

Um feingranulare Aussagen über die Wartbarkeit der einzelnen Persistenztechniken zu machen wurden die wichtigsten und am häufigsten durchgeführten Wartungsaufgaben identifiziert. Da die Persistenz-Framework-Schicht (der Einfachheit halber hier nur Persistenzschicht genannt) das Bindeglied zwischen Datenbank und Objektmodell ist, schlagen sich Änderungen in den beiden anderen angrenzenden Schichten meist ebenfalls auf die Persistenzschicht nieder. Der Hauptbestandteil der Änderungen ist hier meist entweder das Hinzufügen, Ändern oder Löschen von persistenten Klassen sowie Relationen zwischen diesen. Die daraus abgeleiteten Fragen für den GQM-Plan sind wie folgt:

**Question 1.3:** Wie groß ist der Aufwand um die Persistenzschicht um eine neue persistente Klasse zu erweitern sowie diese in das existierende Objektmodell zu integrieren (einschließlich der Modifikation eventuell betroffener Klassen und Relationen)?

Die folgenden Metriken sollen erhoben werden um Frage 1.3 zu beantworten

**M1.3.1:** Durchschnittliche Zeit, die benötigt wurde um die Änderung durchzuführen in *Minuten*.

**M1.3.2:** Durchschnittliche *Anzahl von Dateien bzw. Modellen*, die dabei geändert werden mussten.

**M1.3.3:** Durchschnittliche *Anzahl der Test- und Debug-Läufe*, die benötigt wurden um alle Tests, nach einer Änderung, zu erfüllen.

**M1.3.4:** Durchschnittliche *Zeit in Minuten*, die benötigt wurde um eventuelle Fehlerkorrekturen durchzuführen nachdem eine Änderung durchgeführt wurde.

Nicht nur Änderungen im Bezug auf die statische Struktur der Applikation werden erfasst sondern auch Erweiterungen und Änderungen an der Geschäftslogik sollen untersucht werden:

**Question 1.4:** Wie groß ist der Aufwand die Funktionalität einer Geschäftslogikfassage zu ändern oder zu erweitern?

Die gleichen Metriken wie für Frage 1.3 treffen auch hier zu, nummeriert als **M1.4.1**, **M1.4.2**, **M1.4.3** und **M1.4.4**.

Für die Fragen 1.3 und 1.4 wurden diverse Änderungsszenarien definiert, welche diese Änderungen nach sich ziehen. Um der Frage der externen Validität der Evaluierung (siehe Sektion 7.4) nachzukommen wurden diese Szenarios basierend auf wirklichen Änderungsanforderungen an das Altsystems erarbeitet. Auch der Typ der Änderungen wurde variiert. Sowohl Änderungen am Datenbank-Back-End, der Abbildungsdefinition als auch der Geschäftslogik wurden untersucht. Da hier keine ausführliche

Beschreibung der Altanwendung präsentiert werden kann, sei hier nur ein kurzer Überblick über die Szenarien gegeben. Diese Szenarien wurden anhand der Chartproduction Applikation durchgeführt:

1. Änderungsszenario 1. Änderung des Datenbank Back-Ends von Oracle 9i zu Microsoft SQL Server.
2. Änderungsszenario 2. Hinzufügen eines neuen Features (Währungsumrechnung) zur Applikation. Dies schließt das Hinzufügen von zwei neuen, persistenten Klassen, zwei neuen Relationen, einer neuen Geschäftslogikmethode sowie die Änderung einer weiteren Methode der Geschäftslogik ein.
3. Änderungsszenario 3. Ein neues Kriterium (Der Aktienindex einer Aktie) für die Erstellung der Kursdiagramme soll eingeführt werden. Dies bedeutet, dass eine neue, persistente Klasse sowie eine neue Relation hinzugefügt werden muss. Diverse Methoden der Geschäftslogik müssen ebenfalls geändert werden.

Für jedes dieser Szenarien wurden die folgenden Fragen/Metriken gesammelt:

**Question 1.5:** Wie groß ist der Aufwand um alle Änderungen des jeweiligen Szenarios durchzuführen?

**M1.5.1:** Zeit um die Änderung durchzuführen in *Minuten* (ohne Test/Debug-Zeit).

**M1.5.2:** *Anzahl der Dateien/Modelle* die angefasst werden mussten.

**M1.5.3:** *Anzahl der Debug- und Testläufe* um alle Tests nach einer Änderung wieder erfolgreich durchlaufen zu können.

**M1.5.4:** Zeit in *Minuten*, die für Test- und Debug-Läufe aufgewendet werden musste.

### 7.2.2. GQM-Plan für Performance

Zusätzlich zu Wartbarkeitsaspekten ist meist die Performance einer Persistenztechnologie einer der wichtigsten Belange. Um die Performance einer Persistenztechnologie zu messen wurde das zweite Meß-Ziel mit Hilfe der GQM-Methode folgendermaßen definiert:

**Goal 2: Purpose:** Vergleich

**Object:** Verschiedene Persistenztechnologien

**Issue:** Performance

**Viewpoints:** Entwicklungs- und Wartungsteam

Dieses Ziel enthält Fragen generischer Art. Sie können für nahezu alle Client-Applikationen des Beispielsystems gestellt werden. Da jedoch jede der Applikationen unterschiedliche Performance-Anforderungen hat unterscheidet, sich die Meßmethode abhängig vom Use-Case-Szenario in welchem sie gesammelt wurden.

**Question 2.1:** Welche Auswirkungen hat die Wahl des Persistenzsystems auf die Startzeit der Anwendung?

Um dies zu messen wurden die folgenden Metriken benutzt:

**M2.1.1:** Zeit um die Anwendung zu initialisieren, gemessen in *Millisekunden*. Diese Metrik mißt die Zeit, die benötigt wird um die Applikation vollständig zu starten. Dies bezieht sich auch auf die Initialisierung von Bibliotheken, das Öffnen der Datenbankverbindungen sowie das Laden von initialen Daten von der Datenbank.

**M2.1.2:** Der initiale Speicherverbrauch nachdem die Anwendung komplett gestartet wurde (Gemessen in *Megabytes*).

Die Interpretation der Ergebnisse dieser Metriken basieren auf der Annahme, dass kurze Initialisierungszeiten besser sind als lange. Allerdings ist es sehr schwer zu sagen, ob eine breite oder eine enge Verteilung der Werte besser ist. Ein geringerer Speicherverbrauch hingegen ist ein Indikator für eine bessere Start-Performance.

Ein weiterer wichtiger Punkt ist es, das spezifische Performance-Verhalten einer Persistenztechnologie zu identifizieren. Einige sind schneller bei der Ausführung von Queries haben jedoch, z. B. durch das Vorhandensein eines grösseren Caches, einen höheren Speicherverbrauch. Um diese Charakteristiken zu identifizieren wurden die folgende Frage abgeleitet:

**Question 2.2:** Was sind die spezifischen Performance-Charakteristiken des jeweiligen Persistenzsystems für die Anwendung?

Die Metriken, welche helfen diese Frage zu beantworten, messen die Ressourcenauslastung während der Ausführung von vordefinierten, repräsentativen Use-Case-Szenarios. Diese Metriken wurden während der automatischen Ausführung von Szenarien gesammelt, welche Aufgaben der jeweiligen Applikation repräsentieren.

**M2.2.1:** CPU Auslastung durch die Applikation. Gemessen in *Prozent*.

**M2.2.2:** CPU Auslastung durch die Datenbank. Gemessen in *Millisekunden* welche in den Operationen der Datenbank verbraucht werden.

**M2.2.3:** (a) Zeit welche für eine spezifische Aufgabe benötigt wird (Gemessen in *Millisekunden*). (b) *Durchsatz Aufgaben pro Minute*. (c) *Zeit für den Commit* der Änderungen zur Datenbank.

**M2.2.4:** Speichernutzung (in *Megabytes*) während der Ausführung des Szenarios. Im speziellen das Maximum an benutztem Speicher während der Testläufe.

Nahezu alle Anwendungen des Beispielsystems haben eine direkte Interaktionsschnittstelle mit dem Benutzer. Deswegen ist es wichtig zu erfassen, wie responsiv das System ist.

**Question 2.3:** Wie gut ist die Reaktionsfähigkeit des Systems, aus Sicht des Clients?

Alle Operationen, welche durch die Applikationen ausgeführt werden können sind in einer entsprechenden Geschäftslogikfassade gebündelt. Deshalb kann man anhand der Antwortzeit dieser Fassadenmethoden einen Eindruck der Gesamt-Responsivität der Hauptfunktionalitäten der Anwendung gewinnen.



**M2.3.1:** Antwortzeit in *Millisekunden* der Methoden der Geschäftslogikfassaden. Gemessen pro Methode mit wechselndem Benutzungsprofil.

Um realistische Aussagen über die Performance der einzelnen Persistenzmechanismen machen zu können wurde für jede Geschäftslogikfassade eine spezielle Implementierung geschaffen, welche die spezifischen Querymechanismen der Technologien entsprechend nutzt. Allerdings ist es ebenso wichtig, die Performance der Ansätze bei reiner Benutzung von Objektnavigationen zu messen. Dies wird ermöglicht durch eine zusätzliche Implementierung der Fassade welche die gleiche Funktionalität wie die Query-Mechanismen hat, allerdings nur direkte Navigation auf den Objekten durchführt.

**Question 2.4:** Wie gut ist die Responsivität des Systems bei Benutzung einer Geschäftslogikfassade, welche nur generische Objektmodellnavigation durchführt?

**M2.4.1:** Antwortzeit in *Millisekunden* der Methoden der **generischen** Geschäftslogikfassade. Gemessen pro Methode mit wechselndem Benutzungsprofil.

## 7.3. Ergebnisse der Evaluierung

Für die Evaluierung von Performance und Wartbarkeit der verschiedenen Persistenzsysteme wurden zwei Applikationen implementiert, welche auf dem bereits erwähnten MESCOR-System basieren.

### 7.3.1. Aufsetzen des Experiments

Für das Sammeln der Wartbarkeitsmetriken wurden verschiedene Entwicklergruppen zusammengestellt, welche die Aufgabe hatten, die unterschiedlichen Systeme zu implementieren. Insgesamt arbeiteten sechs Entwickler an dem System. Jeder von ihnen hatte bereits Erfahrung in mindestens einem der eingesetzten Persistenzsysteme und/oder der Zieldomäne der entwickelten Applikation bzw. der Applikation selbst. Die Entwicklungsarbeit selbst wurde in Pairprogramming-Sessions mit wechselnden Paaren durchgeführt um den Einfluss der unterschiedlichen Erfahrungsstände möglichst gering zu halten. Die Teams wurden zwischen den Entwicklern der andrena objects ag und dem FZI gemischt um ein möglichst balanciertes Team bezüglich der unterschiedlichen Erfahrungen zu erhalten.

### 7.3.2. Ergebnisse für Goal 1: Wartbarkeit

#### Client Anwendung 1: Chartproduction

Nicht alle nötigen Features um die komplette Bandbreite an Funktionalität der evaluierten Applikation zu implementieren wurden durch die eingesetzten Frameworks komplett unterstützt. Hauptsächlich die Tatsache, dass die Altanwendung ein lang gewachsenes System ist, das auch nicht immer gleichartig und konsequent weiterentwickelt wurde, verursachte einiges an Zusatzaufwand. Metrik M1.1.3 und M1.1.4 zeigen

die Anzahl der Workarounds sowie die benötigte Zeit diese zu implementieren. Da die ADO.Net Implementierung des O/R Mappers von Grund auf von Hand geschrieben wurde, konnte hier speziell auf die Eigenheiten der Altdatenbank eingegangen werden. Die Implementierungen der beiden O/R Mapping Frameworks (NHibernate und OpenAccess) benötigten teilweise aufwändige Workarounds um das Legacyschema auf das Objektmodell abbilden zu können.

Die Chartproduction Anwendung diente nicht nur als initiale Anwendung um die diversen Persistierungstechniken anzuwenden, sondern ebenso als Basis für das Erstellen eines Generators, welcher auf dem openArchitectureWare Generatorframework basiert. Wie in Tabelle 7.1 dargestellt betrug der zusätzliche Aufwand zur Erstellung dieses Generators drei Personentage. Allerdings wurde während der Erweiterung des Systems um weitere Applikationen diverse Features benötigt, die noch nicht im Generator vorgesehen waren. Während der Änderungsszenarien zwei und drei (siehe Tabelle 7.2) wurde zusätzliche Zeit benötigt um den Generator mit den notwendigen Änderungen zu versehen.

Der erste Versuch AndroMDA innerhalb dieser Evaluierung als Representant der Out-of-the-box-Generatoren Klasse zu benutzen führte zu dem Ergebnis, dass der von AndroMDA bereitgestellte C# und NHibernate Generator bei weitem nicht flexibel genug war um die Altdatenbank in annehmbarem Maße abbilden zu können. Des Weiteren war es nicht möglich den Generator so anzupassen, dass er zu den bereitgestellten Interfaces passt. Somit lässt sich als Ergebnis festhalten, dass Out-of-the-box-Generatoren für diese Art der Erstellung von Objektrelationellen Abbildungen nur bedingt geeignet sind.

Eine Evaluierung der Wartbarkeitsmetriken für die im MINT-Kontext entwickelten Tools von Delta Software Technology war zu diesem Zeitpunkt aufgrund der Fortgeschrittenheit der Tools nicht vergleichbar und wurde somit nur ansatzweise durchgeführt. Da sich jedoch zeigte, dass auch in einem frühen Stadium diese Art der Werkzeuge einen vielversprechenden Ansatz bescheinigen, ist davon auszugehen, dass sich die Ergebnisse wohl im Bereich der modellgetriebenen Entwicklung mit oAW bewegen.

Wie in Tabelle 7.1 dargestellt, lässt sich erkennen, dass der initiale Entwicklungsaufwand für die Chartproduction Anwendung recht stark in Abhängigkeit mit der eingesetzten Persistenztechnologie schwankt. Die Ergebnisse von Metrik 1.1.2 zeigen, dass state-of-the-art O/R Mapping Frameworks alle benötigten Aspekte der Persistenzschicht abdeckt, welche von der Evaluierungsapplikation benötigt werden. Aufgrund unserer Erfahrungen mit der ADO.Net Implementierung lässt sich sagen, dass für diese konkrete Art von Applikation, bei welcher das Schema der Altdatenbank erhalten bleiben soll, diverse Trade-Off-Entscheidungen zwischen Abstraktion und Flexibilität gemacht werden müssen. Einige Teile der Implementierung könnten in Framework-Komponenten ausgelagert werden, ganz so wie dies bereits bei O/R-Mapping-Frameworks geschieht. Allerdings gibt es auch einige Teile der Anwendung die nicht in ein generisches Schema passen und daher spezifische Aufmerksamkeit und Implementierungen benötigen. Dies wiederum resultiert in zusätzlichen Aufwänden für Workarounds, welche ebenso zu Performance-Verschlechterungen führen können. Abhängig vom Grad der Unkonventionalität der Altdatenbank könnte es daher ebenso eine sinn-

Tabelle 7.1.: Resultate der Wartbarkeitsevaluierung

Metrik[Einheit]	ADO	NH	OA	OAW
1.1.1[PD]	12	9	<b>6</b>	6 + 3 <sup>a</sup>
1.1.2	5	<b>0</b>	<b>0</b>	n.a.
1.1.3	1	1	1	n.a.
1.1.4[PD]	<b>0,5</b>	2	1	n.a.
1.3.1[m]	19,75	25	31,5	<b>18,75</b>
1.3.2	3	4	4	<b>1 - 2<sup>b</sup></b>
1.3.3	2,25	4,25	2,5	<b>1,25<sup>c</sup></b>
1.3.4[m]	89,25	66,5	48,25	<b>4,5<sup>d</sup></b>
1.4.1[m]	<b>5</b>	14	11,5	14

<sup>a</sup>6 Tage für den NHibernate Prototypen, 3 Tage um den Generator aus dem Prototypen abzuleiten

<sup>b</sup>Abhängig vom Bedarf für zusätzliche Query-Methoden in den Manager-Klassen

<sup>c</sup>plus zusätzlich 1,75 Stunden um Fehler am Generator zu beheben (verursacht durch den sehr frühen Stand des Generators)

<sup>d</sup>plus zusätzlich 11 Minuten um Fehler am Generator zu beheben (verursacht durch den sehr frühen Stand des Generators)

volle Entscheidung sein eine manuelle Implementierung anzustreben.

An den Ergebnissen zu Änderungsszenario 1, welches das Datenbankmigrationsszenario ist, lässt sich erkennen (siehe Tabelle 7.3), dass der Zeitaufwand für ADO.Net immernoch relativ hoch ist, obwohl diese schon eine gewisse Abstraktion vom Datenbankzugriff bieten. Allerdings ist diese Abstraktion noch nicht hoch genug um die Datenbank austauschen zu können ohne dazu noch datenbankspezifische Elemente so wie z. B. Query-Parameter anpassen zu müssen. Durch eine entsprechende Abstraktion der Query-Sprache (wie z. B. OQL bei OpenAccess oder HQL bei NHibernate) lässt sich eine solche Migration mit deutlich weniger Zeitaufwand durchführen.

Für die Szenarios 2 und 3 ergaben sich fast die erwarteten Ergebnisse bezüglich der Implementierungszeit (M1.5.1): Die Implementierungen mit Hilfe der O/R Mapping-Frameworks können deutlich schneller evolviert werden als diejenige mit ADO.Net. Des Weiteren verkürzt der Einsatz von modellgetriebenen Techniken die Implementierungszeit zusätzlich. Der schlechte Wert von OpenAccess in Szenario 2 resultiert aus Problemen das OpenAccess eigene Reverse-Engineering Werkzeug mit weiteren, tiefgreifenderen Features zu benutzen. Eine weitere Auffälligkeit ist der hohe Zeitbedarf für das Testen und Debuggen des NHibernate Mappings. Dieses entstand hauptsächlich durch die sehr ungenauen Fehlermeldungen von NHibernate. Es war mehr als nur einmal von Nöten, das NHibernate-Forum zu konsultieren um Erklärungen und Lösungen für bestimmte Probleme zu erhalten.

## Client Anwendung 2: Accounting

In Tabelle 7.3 sind die Ergebnisse der Wartbarkeitsevaluierung für die Accounting-Applikation dargestellt. Verglichen mit der Implementierung der Chartproduction-

Tabelle 7.2.: Ergebnisse der Evaluierung der Evolutionsszenarien

Metrik[Einheit]	ADO	NH	OA	OAW
Szenario 1				
1.5.1[m]	40	<b>10</b>	<b>10</b>	<b>10</b>
1.5.2	7	<b>1</b>	<b>1</b>	<b>1</b>
1.5.3	5	<b>1</b>	<b>1</b>	<b>1</b>
1.5.4[m]	30	<b>0</b>	<b>0</b>	<b>0</b>
Szenario 2				
1.5.1[m]	78	55	82	<b>39</b>
1.5.2	7	10	8	<b>3(+1)<sup>a</sup></b>
1.5.3	<b>5</b>	13	<b>5</b>	2(+6) <sup>b</sup>
1.5.4[m]	108	29	<b>6</b>	3(+31)
Szenario 3				
1.5.1[m]	78	56	47	<b>39</b>
1.5.2	6	8	7	<b>3(+1)<sup>c</sup></b>
1.5.3	4	4	5	3(+1)
1.5.4[m]	249	237	187	<b>15(+12)</b>

<sup>a</sup>Hiermit sind 3 Quelldateien sowie eine Modelldatei gemeint.

<sup>b</sup>Die zusätzlichen Werte in Klammern beziehen sich auf Fehler im Generator

<sup>c</sup>Hiermit sind 3 Quelldateien sowie eine Modelldatei gemeint.

Anwendung lässt sich hier feststellen, dass die Aufwände, welche sich für die OpenAccess bzw. NHibernate Implementierung ergaben, nun die Reihenfolge getauscht haben. Dies resultiert hauptsächlich aus den unterschiedlichen Entwicklungsansätzen welche für die Frameworks eingesetzt werden. OpenAccess bietet einen Wizard und Editorunterstützung für seine Mappings, welche dem Entwickler helfen, initiale Mappings sehr schnell zu erstellen. Allerdings nimmt dieser Vorteil ab, sobald der Entwickler sich in das Erstellen der Mappings eingearbeitet hat. Hier geht die Erstellung der Mappings durch direktes Editieren der Mapping-Dateien meist deutlich schneller voran. Im speziellen auch dadurch, dass die Mapping-Beschreibung von NHibernate einfacher zu lesen und zu verstehen sind als die von OpenAccess.

Der Aufwand um die ADO.Net-Mappings zu implementieren sowie die Zeit diese zu Testen und zu Debuggen war deutlich höher als bei den anderen Ansätzen. Dies erklärt sich hauptsächlich durch die Komplexität, die durch die Kombination der verschiedenen implementierten Muster (wie z. B. Lazy Loading oder Unit of Work) in die Mappings hineinkommt. Die modellbasierte Technik zeigt hier ganz klar ihre Vorteile in Hinsicht auf Entwicklungsaufwand sobald der Generator eine gewisse Reife erlangt hat. Speziell die Zeit welche für das Debuggen und Testen nötig war verkleinerte sich merklich. Einer der wichtigsten Einflüsse hier war die Elimination von systematischen Fehlern durch eine Behebung im Generator, anstelle des Suchens und Behebens innerhalb des gesamten Codes.

Tabelle 7.3.: Ergebnisse der Warbarkeitsevaluierung

Metrik[Einheit]	ADO	NH	OA	OAW
1.2.1[PD]	4,2	3	3,25	<b>0,9</b>
1.2.2[PD]	8,1	3,2	4,2	<b>1,5</b>

Tabelle 7.4.: Ergebnisse der Performance-Evaluierung für Frage 2.1 und Anwendung 1: Chartproduction

Metrik[Einheit]	ADO	NH	OA	SCORE
M2.1.1[ms]	5331,3	6211,2	6961,4	<b>1462,5</b>
M2.1.2[ms]	<b>51,8</b>	56,1	51,9	52,9

### 7.3.3. Ergebnisse für Goal 2: Performance

Da der oAW-Generator im wesentlichen den gleichen Code bzw. die gleichen Mapping-Files wie die manuelle NHibernate-Implementierung erzeugt, wurden die Performance-Tests nur für ADO.Net, OpenAccess sowie die manuelle Implementierung der NHibernate-Mappings durchgeführt.

#### Client Anwendung 1: Chartproduction

Die ersten Testläufe um die Performance-Metriken zu sammeln zeigten, dass die Techniken nahezu gleich schnell bezüglich der Chartproduction-Anwendung sind. Allerdings bleibt hervorzuheben, dass der Caching-Mechanismus von NHibernate einen kritischen Aspekt in diesem Szenario darstellt. Die Chartproduction-Anwendung hat einen speziellen Stapelverarbeitungsmodus, in welchem sie für alle Aktien innerhalb des Repositories Verlaufskurven erzeugt. Während dieses Laufes stieg die Zeit, die für die Erstellung einer einzelnen Kurve benötigt wurde, linear an, was in einer bis zu 15 mal höheren Laufzeit des Gesamtprozesses im Vergleich zur ADO.Net Implementierung resultierte. Durch Änderung der NHibernate-Implementierung, bei der der Sitzungs-Cache nach der Erstellung einer Kurve gleich wieder gelöscht wird, waren beide Technologien erneut relativ gleich schnell. Es ist weiterhin wichtig zu erwähnen, dass die Performance dieser ersten Implementierungen nur fast gleich war für diesen speziellen Anwendungsfall. Weitere Tests zeigten, dass die Performance der unterschiedlichen Implementierungen in anderen Szenarien recht stark von einander abweicht. Tabelle 7.4 zeigt die Mittelwerte für alle gesammelten Performance-Metriken. In den meisten Fällen können diese Werte als repräsentativ angesehen werden. Allerdings ist für manche Metriken in einigen Fällen der Mittelwert nicht genug um die eigentlichen Ergebnisse ausreichend darzustellen. Diese Fälle sind speziell markiert und werden in einem eigenen Abschnitt erklärt.

**Metrik M2.1.1 und M2.1.2:** Die Initialisierungszeit der Anwendung variiert für die verschiedenen Persistenztechnologien. Die Initialisierung von OpenAccess bei-

Tabelle 7.5.: Ergebnisse der Performance-Evaluierung für Frage 2.2 und Anwendung 1: Chartproduction (Mittelwerte bis auf M2.2.4)

Metrik	ADO	NH	OA	SCORE
M2.2.1[%]	86,5	88,5	<b>74,2</b>	87,3
M2.2.2[%]	<b>3,3</b>	5,8	7,5	4,1
M2.2.3a[ms]	87,2	194,0	114,8	<b>84,1</b>
M2.2.3b[1/m]	11,5	5,15	8,7	<b>11,9</b>
M2.2.4[mb](max.)	<b>2,2</b>	6,5	24,0	<b>2,3</b>

spielsweise braucht fast ein Drittel länger als die der ADO.Net Implementierung. Die ADO.Net-Implementierung ist, durch ihre Schlankheit und ihren stark projektspezifischen Charakter, leicht schneller als die schwergewichtigen O/R-Frameworks. Der ebenfalls projektspezifisch generierte Adapter von SCORE, welcher mit den Delta Software Technology Werkzeugen erzeugt wurde, zeigt hier eine extrem gute Performance was die Aufstartzeit angeht.

Der initiale Speicherverbrauch ist bei Hibernate ebenso höher. Im Kontext dieser Applikation ist dies zwar kein Problem, wenn man allerdings kleinere Anwendungen betrachtet, die öfter gestartet und geschlossen werden, spielen die Startzeit sowie der Speicherverbrauch eventuell eine entscheidendere Rolle. SCORE ist hier ähnlich schlank wie ADO.

**Metric M2.2.1 bis M2.2.4:** Um die Werte für diese Metriken zu sammeln wurde das primäre Anwendungsszenario der Applikation verwendet, welches eine Stapelverarbeitung zur Erstellung der Verlaufskurven ist. Aktuell befinden sich rund 500 Aktien in der Datenbank. Während eines Stapelverarbeitungslaufes werden die Kurven für alle diese erzeugt. Die Zeiten für die Erstellung jeder einzelnen Kurve wurden aufgezeichnet. Da es sich hierbei schon um eine weit verteilte Menge an Werten handelt, wurde zur Darstellung in Tabelle 7.5 nicht die Gesamtheit der Läufe zusammengefasst, sondern ein repräsentativer Lauf ausgewählt.

Abbildung 7.6 zeigt die Häufigkeitsverteilung der Zeiten, die für die Erstellung der Charts innerhalb eines Stapelverarbeitungslaufes benötigt wurden (M2.2.3). Es zeigt sich, dass in diesem Szenario ADO.Net und SCORE ein wenig schneller sind als die OpenAccess-Implementierung, wobei NHibernate langsamer als die anderen Ansätze ist. Es ist allerdings auch sehr interessant, dass es, wenn auch nur einige wenige, Ausreißer innerhalb der Werte für das OpenAccess-Framework gibt. In Szenarien, in welchen eine garantierte maximale Ausführungszeit erwartet wird, könnte dies ein entscheidender Punkt für oder gegen den Einsatz einer Persistenztechnologie sein. Da der durch SCORE generierte Adapter auf einem manuell entwickelten ADO-Adapter basierte, sind hierfür die Werte fast deckungsgleich mit denen der Handimplementierung und deshalb in diesem Diagramm nicht noch einmal separat aufgeführt. Die Durchschnittswerte lassen sich aus Tabelle 7.5 herauslesen.

Der Verbrauch des Heap-Speichers der untersuchten Frameworks (M2.2.4) während

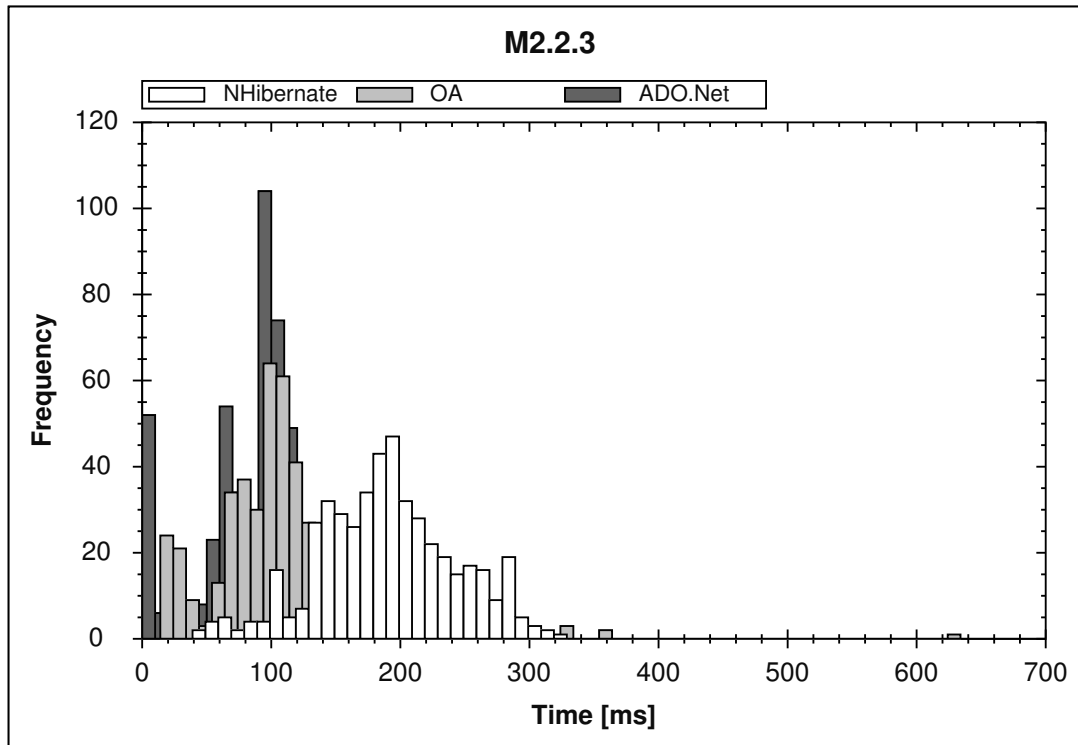


Abbildung 7.6.: M2.2.3: Häufigkeitsverteilung der Zeiten für die Erstellung von Verlaufskurven.

des Stapelverarbeitungslaufes ist in Abbildung 7.7 dargestellt. Obwohl OpenAccess NHibernate bezüglich der Ausführungszeit übertrumpft, ist das Verhältnis bezüglich des Speicherverbrauchs umgedreht. Nicht nur der Gesamt Speicherverbrauch von OpenAccess ist drei mal höher als der von NHibernate, es gibt zusätzlich auch Spitzen, welche nochmal so hoch sind wie das Maximum das von NHibernate verbraucht wird. In speicherkritischen Systemen könnte dies ein entscheidender Punkt bezüglich einer Entscheidung für NHibernate gegenüber OpenAccess sein.

**Metrik M2.3.1** Wenn man die Resultate in Tabelle 7.6 mit denen des Stapelverarbeitungsszenarios vergleicht, erkennt man die große Diskrepanz zwischen beiden Szenarien. Obwohl die meisten Fassadenoperationen langsamer mit ADO bzw. SCORE sind als mit den anderen Technologien, ist die Gesamt-Performance innerhalb des Stapelverarbeitungslaufes dennoch besser. Dies resultiert hauptsächlich aus der besseren Performance der Operationen `GetSharePrices` and `GetShareRelativePerformance`.

**Metrik M2.4.1** Wie in Tabelle 7.7 zu sehen ist, weichen die Werte bei Verwendung von reiner Objektmodellnavigation gegenüber dem Einsatz von zusätzlichen Query-Mechanismen (M2.3.1) deutlich ab. Vergleicht man die Ergebnisse für diese Metrik mit Tabelle 7.6 so lässt sich erkennen, dass der handgeschriebene ADO.Net Adapter bzw. der auf ähnlicher Ebene agierende generierte Adapter von SCORE Vorteile durch ihre

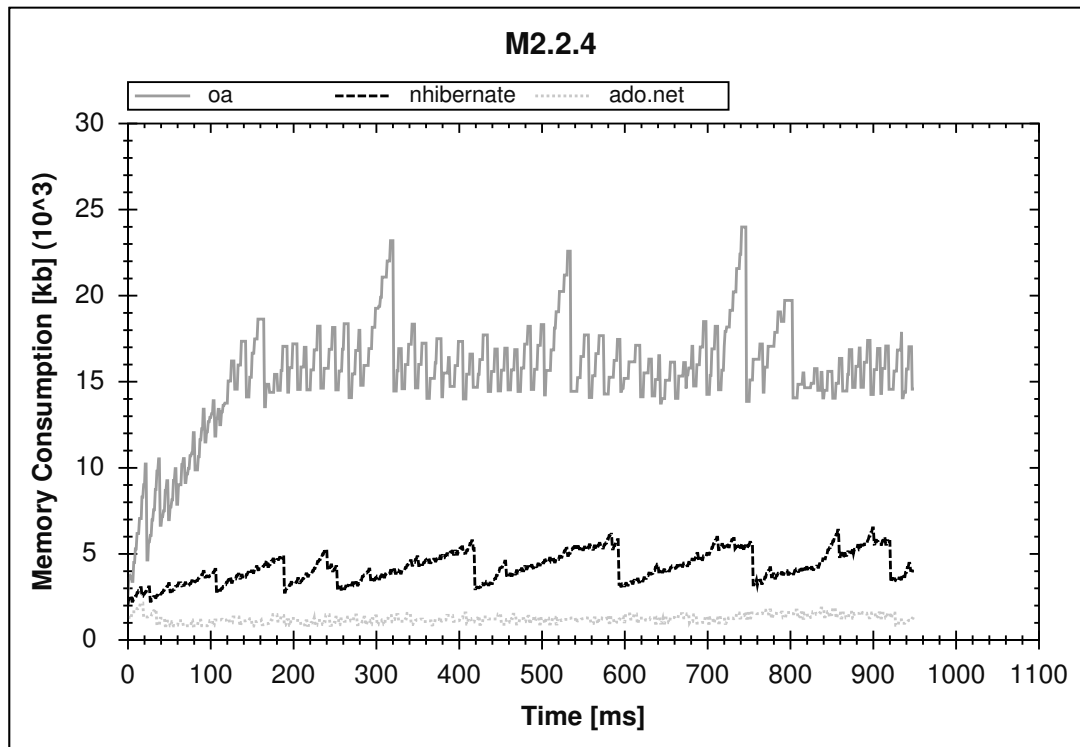


Abbildung 7.7.: M2.2.4: Speicherverbrauch während der Erstellung der Verlaufskurven.

Tabelle 7.6.: M2.3.1 Mittelwerte der Ausführungszeiten der Chartproduction-Fassade in Millisekunden (ms).

Metrik	ADO	NH	OA	SCORE
GetCompany	9,5	12,1	<b>8,5</b>	8,9
GetExchangeRate	<b>9,6</b>	103,9	59,4	10,2
GetShareIdentifierForType	23,0	27,3	<b>9,5</b>	21,7
GetShareIdentifiers	279,0	1858,1	<b>44,0</b>	264,2
GetSharePrices	49,8	118,7	64,3	<b>47,4</b>
GetShareRelativePerformance	<b>52,0</b>	128,7	60,6	54,1
GetShares	1279,4	90,3	<b>38,4</b>	1165,5
GetStockindex	85,1	<b>58,8</b>	80,8	79,4
<b>Overall</b>	1787,4	2398,0	<b>365,6</b>	1643,2



Tabelle 7.7.: M2.4.1 Mittelwerte der Ausführungszeiten der *generischen* Chartproduction-Fassade in Millisekunden (ms)

Metrik	ADO	NH	OA	SCORE
GetCompany	8,2	10,2	<b>6,4</b>	8,1
GetExchangeRate	<b>10,6</b>	100,4	62,1	<b>12,1</b>
GetShareIdentifierForType	23,0	21,3	<b>12,8</b>	20,5
GetShareIdentifiers	1285,6	2073,4	<b>266,9</b>	1155,9
GetSharePrices	387,6	938,5	818,7	<b>370,9</b>
GetShareRelativePerf.	<b>98,4</b>	258,2	208,3	102,1
GetShares	270,2	<b>37,5</b>	1156,8	266,4
GetStockindex	84,4	<b>54,6</b>	77,9	81,2
<b>Overall</b>	2162,8	3403,9	2559,0	<b>2017,2</b>

Tabelle 7.8.: Ergebnisse der Performance-Evaluierung von Frage 2.1 für Applikation 2: Accounting in Millisekunden (ms)

Metrik[Einheit]	ADO	NH	OA	SCORE
M2.1.1[ms]	1633,8	<b>1613,3</b>	1647,5	1624,2
M2.1.2[mb]	<b>7,6</b>	14,6	8,5	<b>7,8</b>

Low-Level-Query-Mechanismus zu haben scheinen. Bei reiner Objektmodellnavigation hingegen sind beide O/R Mapper wiederum deutlich schneller als die ADO/SCORE-Implementierungen.

### Client Anwendung 2: Accounting

Da die Accounting-Anwendung ein Lese-/Schreib-Szenario ist, in welchem Objektstrukturen erzeugt, editiert und gelöscht werden, spielen hier andere Aspekte eine Rolle als im nur-lesenden Chartproduction-Szenario.

**Metrik M2.1.1 und M2.1.2:** Während des Startens der Applikation wird, abgesehen von der generellen Initialisierung der Datenbankverbindung und der Manager-Klassen, ein initiales Set von Unternehmen und Sichten von der Datenbank geladen. Verglichen mit den Resultaten von Frage Q2.1 der Chartproduction-Applikation, kann man erkennen, dass die höhere Speicherauslastung für Objekte in OpenAccess im initialen Memory-Footpring der Applikation reflektiert ist. Zusätzlich ist nun auch die Aufstartzeit mit OpenAccess höher als mit NHibernate.

Für die Performance-Messung der Accounting-Applikation wurden zwei lesende, aktualisierende sowie schreibende Szenarios definiert. Eines mit einigen wenigen selektiven Änderungen (single edit) sowie ein anderes in dem mehrere Unternehmen inklusive einige ihrer Werte gelesen und editiert wurden (multiple edits). Um ein statistisch einigermaßen aussagekräftiges Ergebnis zu erhalten wurde jedes der Szenarien zehn mal ausgeführt und gemessen. Die Durchschnittswerte sind in Tabelle 7.9 zu sehen.

Tabelle 7.9.: Ergebnisse der Performance-Evaluierung für Frage Q2.2 für Applikation 2: Accounting (alles Durchschnittswerte ausser M2.2.4) in Millisekunden (ms)

	Multiple Edits			
Metriken[unit]	ADO	NH	OA	SCORE
M2.2.1[%]	84,6	77,2	70,5	83,1
M2.2.2[%]	6,4	0,07	5,5	6,6
M2.2.3a[ms]	<b>3687,3</b>	56589,4	6688,9	2834,7
M2.2.3c[ms]	<b>49,6</b>	149,9	67,4	50,1
M2.2.4(max.)[mb]	23,8	22,8	<b>16,3</b>	21,9
	Single Edit			
Metriken[unit]	ADO	NH	OA	SCORE
M2.2.1[%]	94,3	90,7	64,8	93,1
M2.2.2[%]	0,7	0,13	8,1	0,8
M2.2.3a[ms]	<b>820,4</b>	11230,1	2187,7	845,7
M2.2.3c[ms]	40,13	143,6	<b>13,7</b>	37,6
M2.2.4(max.)[mb]	12,8	12,7	<b>6,3</b>	13,1

**Metrik M2.2.1 bis M2.2.3:** Innerhalb des Lese/Schreib-Szenario stellte sich heraus, dass das NHibernate Mapping signifikant langsamer ist, als das entsprechende OpenAccess-, ADO-, bzw. SCORE-Mapping. Höchstwahrscheinlich ist der Grund hierfür die schlechtere Performance der Objektmodellnavigation, wie schon in Tabelle 7.7 dargelegt wurde. Da das Accountingszenario sehr stark auf der Navigation zwischen Unternehmen, Aktien, verschiedenen Sichten sowie den Unternehmensdaten basiert, ist dieser Einfluß entsprechend groß. Ein weiterer interessanter Punkt ist, dass OpenAccess während der Navigation auf dem Objektmodell mehr Last auf den Datenbankserver gibt und weniger auf den Client (siehe M2.2.2 in Tabelle 7.9). Trotzdem ist es langsamer als ADO bzw. SCORE. Wiederum lässt sich erkennen, dass SCORE eine recht ähnliche Performance wie die handgeschriebene ADO-Implementierung bezüglich dieses Szenarios hat.

**Metrik M2.2.4** Ein interessantes Ergebnis ist die maximale Auslastung des Heap-Speichers der Applikation. Vergleicht man die Ergebnisse mit denen der vorangegangenen Applikation, zeigt sich, dass diese Frameworks, abhängig vom Einsatzszenario, ein sehr unterschiedliches Speicherverhalten aufweisen. Wenn Speicher ein entscheidender Belang eines Projekts ist, könnte es unausweichlich sein, den Speicherverbrauch für jedes spezifische Einsatzgebiet zu evaluieren. Für diesen speziellen Use-Case der Accounting-Applikation scheint OpenAccess die beste Speichernutzungseffektivität zu haben.

## 7.4. Einschränkungen und Validität der Evaluierung

Es gibt zwei Arten von Validität bezüglich der Ergebnisse einer empirischen Studie: die *interne* und die *externe* Validität [19]. Die interne Validität beschreibt die Qualität der Ergebnisse im Bezug auf eine einzelne empirische Untersuchung (z. B.: eine Fallstudie oder ein Experiment). Dies beinhaltet die Qualität der Schlussfolgerungen welche von den unabhängigen zu den abhängigen Variablen der Untersuchung gezogen wurden. Mit anderen Worten, dies beschreibt zu welchem Maße der Einfluss von Störfaktoren ausgeschaltet werden konnte. Die externe Validität bezieht auf das Maß, zu welchem die Ergebnisse über die einzelne Fallstudie bzw. das Experiment verallgemeinert werden können. Meist verhalten sich diese beiden Arten gegenläufig zueinander.

Als Testbett wurde eine real existierende multi-tier Geschäftsanwendung gewählt in welcher die verschiedenen Persistenztechniken deployed wurden. Dies ist der einzige Variationspunkt der Applikation. Um die externe Validität zu unterstützen, wurde die vorgestellte Architektur absichtlich als eine "typische Applikation" dargestellt: von einem technischen Standpunkt aus gesehen, könnte diese Architektur für viele verschiedene Geschäftsanwendungen, über diese spezielle Domäne hinaus, verwendet werden. Um eine hohe interne Validität zu erhalten wurde das Testbett so designed, dass die spezifischen Persistenztechniken auf bestmöglichem Wege eingesetzt werden können. Zum Beispiel mussten in EJB 1.x und 2.x einige Muster berücksichtigt werden um eine zufriedenstellende Performance in bestimmten Situationen zu liefern. Allerdings sind solche Muster speziell für eine bestimmte Persistenztechnik ausgelegt und somit nicht Teil des Testbetts, welches so designed werden sollte, dass es den Austausch der Persistenzschicht erlaubt. Deshalb könnte man die interne Validität der Ergebnisse in Frage stellen, indem man argumentiert, dass die Art wie die jeweiligen Systeme eingesetzt wurden nicht unbedingt ihrem optimalen Einsatz entspricht. Dies würde dann wiederum die externe Validität reduzieren. Dem sind allerdings folgende Argumente entgegenzuhalten:

- Ein technischer Belang wie Persistenz sollte sowieso nicht das Design der domänenorientierten Geschäftslogik beeinflussen. Falls eine Persistenztechnik ein solch spezielles Design des Mittel-Tiers erfordert, ist dies definitiv ein großer Nachteil der Technologie und sollte als Vorbedingung für dessen erfolgreiche Anwendung explizit gemacht werden. Darüber hinaus muss eine Technologie evaluiert und mit konkurrierenden Ansätzen verglichen werden ohne einen spezifischen technischen Belang im Mittel-Tier realisieren zu müssen. Die Flexibilität einen Austausch der Mapping-Technologie durchführen zu können ohne ein anderes Tier ändern zu müssen, ist eine wichtige Anforderung, welche in vielen Projekten vorherrscht.
- Alle Ansätze die hier evaluiert wurden sind dafür bekannt eine akzeptable Performance zu liefern ohne Modifikationen am Geschäfts-Tier vornehmen zu müssen.

Bezüglich der Validität der Wartbarkeitsmessungen ist es wichtig, die persönlichen Erfahrungen der Entwickler zu berücksichtigen. Zusätzliche Einflüsse, so wie Lerneffekte müssen ebenso berücksichtigt werden. Es wurde deshalb versucht diese Effekte

durch den Einsatz von Gruppen, die sowohl aus sehr erfahrenen als auch unerfahrenen Entwicklern bestanden, zu minimieren. Bezüglich der Lerneffekte könnte man argumentieren, dass sie in einer solchen Evaluierung nie ausgeschaltet werden können. Allerdings sind solche Effekte ebenso in realen Szenarios immer vorhanden, wie in jeder empirischen Studie. Außerdem sind solche Ergebnisse besser als überhaupt keine Hinweise. Des Weiteren wurden alle Änderungsszenarien, die verwendet wurden um die Metriken zu erheben, aus realen Anforderungen an die Altanwendung abgeleitet. Dies hilft auch die externe Validität der Wartbarkeitsmessungen zu sichern.

# 8. Entwurfsentscheidungsunterstützung beim Entwurf von Integrationssystemen

*Thomas Goldschmidt, Thomas Kühn, Jochen Winzen*

## 8.1. Best Practices und Anti-Patterns

### 8.1.1. Übersicht

Während des Projektverlaufs haben sich, wie in der Vorhabensbeschreibung vorgegeben, als Ergebnis diverse Best Practices und Anti-Patterns herauskristallisiert. Diese werden nun in den nachfolgenden Abschnitten erläutert.

### 8.1.2. Best Practices - allgemeine Erkenntnisse

Bei der Bearbeitung des Projekts haben sich einige allgemeine Erkenntnisse ergeben, die nachfolgend erläutert werden. Diese Erkenntnisse beziehen sich auf die Auswahl eines geeigneten Persistenzframeworks, sowie auf geeignete Entwurfsmuster, die die Entwicklung effizienter gestalten.

#### **Auslagerung der Persistenzschicht**

Die Auslagerung der Persistenzschicht ermöglicht eine Entkopplung der Persistenzschicht und schafft saubere Schnittstellen zwischen den Subsystemen. Dies erhöht die Austauschbarkeit der Persistenzschicht. Dadurch ergibt sich eine bessere Wartbarkeit / Erweiterbarkeit. Durch *Separation of Concerns* findet eine allgemeine Architekturverbesserung statt.

#### **Unit-Tests**

Der Einsatz von Unit-Tests erhöht das Vertrauen des Entwicklers in das zu entwickelnde System. Diese sollten daher immer zur Absicherung der Funktionalität des Quelltextes implementiert werden. Weiterhin entsteht durch einen Testfall eine Dokumentation des getesteten Objekts.

Durch Unit-Tests kann die zu entwickelnde Software in kleinen Inkrementen getestet werden. Wird dies ordentlich und richtig durchgeführt, hat man zu jeder Zeit einen

Überblick, ob die Software noch korrekt funktioniert. Bei der Test-First-Entwicklung ist vorgesehen, dass der Applikations Quelltext testgetrieben entsteht, d. h. dass zuerst der Test für die zu entwickelnde Klasse / Methode formuliert wird. Weiterführende Literatur zu dieser Thematik findet sich in [36].

### **Unabhängige Datenmodellierung**

Eine unabhängige Modellierung des Objektmodells ist sinnvoll, um die Transparenz des Objektmodells zu erhöhen und um Domänenwissen in die Datenmodellierung zu integrieren.

Daraus ergeben sich höhere Anforderungen an den Mapper / die Mappingschicht, da im schlimmsten Fall eine große Lücke zwischen zu Grunde liegender Altdatenbank und unabhängig modelliertem Objektmodell besteht. Dieses Vorgehen ist empfehlenswert für die Kommunikation mit den Domänenexperten, da die Kommunikation unter Verwendung von Begriffen aus der spezifischen Domäne stattfinden kann, die den Domänenexperten geläufig sind. Im Projektverlauf wurde zuerst versucht, ein Objektmodell anhand der vorhandenen Datenbank zu erstellen. Dies stellte sich als nicht empfehlenswert heraus, da die Datenbank nicht unter objektorientiert gedachten Gesichtspunkten erstellt wurde. Daher wurde von der Applikationsseite ausgehend ein Objektmodell erstellt, das die oben aufgeführten Vorteile mit sich bringt.

### **8.1.3. Best Practices - Entwurfsmuster**

Bei der Kopplung von objektorientiert modellierter Geschäftslogik mit einem bestehenden (Alt-)Datenbanksystem hat sich im Projektverlauf der Einsatz von diversen Entwurfsmustern bewährt. Dabei handelt es sich um etablierte Entwurfsmuster aus dem Bereich des Software-Engineering. Diese werden in den nachfolgenden Abschnitten erläutert. Ganz allgemein kann man sagen, dass Entwurfsmuster die Wartbarkeit eines Softwaresystems vereinfachen, sobald diese richtig eingesetzt werden.

#### **Fassade (facade)**

Hat ein vorhandenes System (viele) Subsysteme mit unterschiedlichen Schnittstellen, bietet sich der Einsatz des Fassaden-Entwurfsmusters an. Das Fassaden-Entwurfsmuster ist ein Strukturmuster. Eine Fassade bietet eine einheitliche und meist vereinfachte Schnittstelle zu einer Menge von Schnittstellen eines Subsystems. Wenn ein Subsystem viele technisch orientierte Klassen enthält, die selten von außen verwendet werden, ist es empfehlenswert eine Fassade zu verwenden. Die Fassade ist eine Klasse, die ausgewählte Methoden enthält, die eine häufig benötigte Untermenge an Funktionalität des Subsystems zusammenfasst. Hier wird die Funktionalität an andere Klassen des Subsystems delegiert und vereinfacht dadurch den Umgang. Weitere Informationen hierzu finden sich in [20].

Die Umsetzung im Projekt sieht folgendermaßen aus: Für jede der drei Applikationen aus der Testumgebung wurde jeweils eine Persistenztechnologie-spezifische Fassade implementiert. Dadurch konnten die allgemein formulierten Unit-Tests für die jeweilige

Applikation von allen benutzten Persistenz-Implementierungen verwendet werden und mussten somit nicht für jede Persistenz-Implementierung neu implementiert werden.

### **Das Datenzugriffsobjekt (DAO - data access object)**

Im Kontext des Zugriffs auf eine Datenquelle (Datenbank, Datei, etc.) bietet sich der Einsatz des Datenzugriffsobjekt-Entwurfsmusters (DAO - Data Access Object) an, um eine Austauschbarkeit der Datenquelle zu gewährleisten. Dadurch soll die eigentliche Programmlogik von technischen Details zur Persistierung der Daten befreit werden und flexibler einsetzbar sein. Das DAO-Muster ist ein Muster für die Gestaltung von Programmierschnittstellen (APIs [application programming interfaces]). Weiterführende Informationen zu diesem Thema finden sich beispielsweise in [16].

### **Unit of Work (UoW)**

Das *Unit of Work* Entwurfsmuster wurde im Kontext der *Accounting*-Applikation quasi bei allen Ansätzen verwendet. Hierbei geht es darum, nicht jede einzelne, kleine Änderung an einem Datum sofort in die Datenbank zu persistieren, sondern, sozusagen summiert, alle kleinen Änderungsschritte auf einmal, in einer Transaktion, an die Datenbank zu übermitteln. Dies bringt als Vorteil eine deutliche Performance-Steigerung. Als weiterführende Literatur zu diesem Thema empfiehlt sich [16].

### **Beobachter Entwurfsmuster - observer pattern**

Bei der Stammdatenverwaltung (*Accounting*) kam das Beobachter-Entwurfsmuster zum Einsatz. Dies gehört zu der Kategorie der Verhaltensmuster (behavioural patterns). Es dient zur Weitergabe von Änderungen an einem Objekt an von diesem Objekt abhängige Strukturen. Die *Accounting*-Applikation besteht hauptsächlich aus einem sogenannten *Grid*, welches wiederum aus einzelnen Zellen besteht (man kann sich das wie Microsoft Excel vorstellen). Da im Falle des *Accountings* viele Zellen in einem Grid vorhanden sind, empfiehlt sich hier der Einsatz dieses Entwurfsmusters. Weiterführende Information zu diesem Thema finden sich in [20].

### **Fabrik Muster - factory pattern**

Bei den verschiedenen Persistenzframework-spezifischen Implementierungen wird zur Erzeugung der Managerklassen das Fabrikmuster (factory pattern) eingesetzt. Dadurch werden dedizierte Schnittstellen zum Erzeugen eines Objekts geschaffen. Dies kapselt eine komplexe Objekterzeugung. Weiterführende Informationen hierzu finden sich in [20].

#### 8.1.4. Best practices für die verschiedenen Szenarien

Nachfolgend werden die gewonnenen Erkenntnisse über Best Practices im Projektumfeld aufgeführt. Dies soll zeigen, für welches Szenario welches Verfahren besonders gut geeignet ist.

##### Manuelle Implementierung

Eine manuelle Implementierung empfiehlt sich bei einer sehr großen Diskrepanz zwischen Objektmodell und der zu Grunde liegenden Altdatenbank. Dadurch entsteht eine hohe Flexibilität um Anpassungen an der Implementierung vorzunehmen. Im Projektverlauf hat sich gezeigt, dass sich bei einer manuellen Implementierung der Einsatz von diversen Entwurfsmustern zur Komplexitätsreduzierung bewährt. Dies wären das Ghostpattern, die Identity Map und das Factory Pattern zur Objekterzeugung. Es wird hierbei nicht weiter auf die einzelnen, verwendeten Entwurfsmuster eingegangen. Weiterführende Informationen zum Identity Map und Ghost (Lazy Loading) Entwurfsmuster finden sich in [16], zum Fabrik Entwurfsmuster in [20].

Es hat sich außerdem gezeigt, dass in diesem Fall die Performance besser ist, als bei der Verwendung eines Persistenzframeworks. Dies lässt sich dadurch erklären, dass aufgrund der Altdatenbank häufig Workarounds bei der Verwendung von Persistenzframeworks implementiert werden müssen.

##### Persistenzframeworks - allgemein

Der Einsatz eines Persistenzframeworks empfiehlt sich, wenn die Diskrepanz zwischen Objektmodell und Datenbank nicht allzu groß ist. Weiterhin ist es in diesem Umfeld möglich, das Mapping automatisiert durch einen Generator erstellen zu lassen. Der Einsatz eines Mapping-Werkzeugs ist auf jeden Fall empfehlenswert, da dies der Fehlerminimierung bzw. der Fehlervermeidung dient. Bei der Benutzung der Persistenzframeworks fiel auf, dass diese nicht immer mit der gegebenen Altdatenbank der Testumgebung zurecht kommen. In solch einem Fall wurden vereinzelt und strukturiert Workarounds eingesetzt.

##### Das Persistenzframework OpenAccess

**Workaround für Mapping von zwei Klassen auf eine Tabelle** Im speziellen Fall der Accounting Anwendung sollen zwei Klassen (*RatioLocation* und *ViewPosition*) auf die gleiche Tabelle "ANSICHTENPOSITION" gemappt werden. Da dies mit OpenAccess in der verwendeten Version nicht möglich ist, wurde folgender Workaround implementiert: Eine zusätzliche Klasse *OAViewPositionAndRatioLocation* wird auf die Tabelle gemappt, welche beide Interfaces implementiert. Die jeweiligen Getter "Location" und "ViewPosition" liefern dann einfach "this" zurück. Dies funktioniert nur in diesem speziellen Fall, da es sich hierbei auch um eine eins-zu-eins Beziehung handelt.



**Assistent / Wizard - Benutzung der Reverseengineeringfunktion** OpenAccess bringt von Haus aus eine Reverseengineeringfunktionalität mit. Damit können recht einfach mit Annotationen versehene POCOs mitsamt Mapping erzeugt werden. Dies geschieht unter Verwendung eines Assistenten. Dies ist bei einem Bottom-Up-Design, bzw. bei einem Double-Mapping-Ansatz sinnvoll, da sich hierbei recht schnell die benötigten Datenbankzugriffsobjekte erzeugen lassen. Bei einem Top-Down-Ansatz ist dies weniger empfehlenswert, da in diesem Fall das Objektmodell nicht von der Datenbank ausgehend erstellt wird.

### **Modellgetriebener Ansatz**

Ein modellgetriebener Ansatz ist eine geschickte Ergänzung beim Einsatz eines Persistenzframeworks. Der Ansatz lohnt sich ab einem Applikationsumfang von ca. 15 bis 30 Tabellen, in Abhängigkeit der dahinterliegenden Komplexität. Der Einsatz vorgefertigter Generatoren ist meist nicht sinnvoll, da diese oft nicht mit der Altdatenbank zurecht kommen. Es ist zusätzlicher Aufwand für die Spezialfallbehandlung notwendig.

Im Falle des verwendeten AndroMDA zeigte sich, dass es von Haus aus einige fertige Generatoren mitbringt. Diese sind aber für die Spezialfallbehandlung nicht unbedingt geeignet und nicht intuitiv anpassbar. OpenArchitectureWare (oAW) bringt wenige fertige Generatoren mit, dafür ist die Generatorsprache intuitiv und leicht erlernbar und somit lassen sich die benötigten Generatoren leicht selbst entwickeln.

Der von Delta Software Technology propagierte Ansatz des *Pattern By Example* 2.5.2 ist ein sehr nützlicher Ansatz, da hier zuerst eine prototypische Implementierung erfolgt und von dieser dann abstrahiert wird. Somit wird auch der Generator auf vorhandenem, im Idealfall qualitativ hochwertigem Quelltext entwickelt. Letztendlich könnte man die Behauptung aufstellen: ist der Original-Quelltext qualitativ hochwertig, so ist auch der daraus abgeleitete Generator qualitativ hochwertig.

### **Double-Mapping**

Ein Double Mapping ergänzt den Einsatz von Persistenzframeworks und Generatoren. Es gibt zwei Objektmodelle, das Persistenzobjektmodell (POM) und das objektorientiert modellierte Businessobjektmodell (BOM). Diese sind durch eine sogenannte Glue-Schicht verbunden. Diese Schicht ist für die Kommunikation zwischen POM und BOM zuständig. Im Projektverlauf stellte sich heraus, dass solch ein Vorgehen ab ca. 15 Tabellen und einer gewissen Lücke zwischen der Altdatenbank und Objektmodell Vorteile bringt. Ein Vorteil hierbei ist, dass man die Geschäftslogik unabhängig von der Persistenzschicht halten kann. Ein weiterer Vorteil ist, dass das Objektmodell unabhängig vom Persistenzmapping gehalten wird.

Eine schematische Darstellung findet in Abbildung 8.1 statt. Außerdem ist ein unabhängig modelliertes Businessobjektmodell geschickter für die Kommunikation mit einem Domänenexperten. Weiterhin ist ein Double Mapping bei einer Teilmigration empfehlenswert, da in diesem Fall eine *smooth migration* durchgeführt werden kann, bei der sowohl der neu entworfene Teil, als auch das Altsystem gleichzeitig benutzt



Abbildung 8.1.: Schematische Darstellung - double mapping

werden können. *Smooth migration* bedeutet, dass nicht die komplette Anwendung auf einen Schlag ausgewechselt wird, sondern Anwendungsteile nach und nach ersetzt werden. Hierbei findet eine Koexistenz von Alt- und Neusystem statt.

### 8.1.5. Anti-Patterns für die verschiedenen Szenarien

Nachfolgend werden die gewonnenen Erkenntnisse über Anti-Patterns im Projektumfeld aufgeführt.

#### ADO.NET Tableadapter

Von der Verwendung von ADO.NET Tableadaptern ist abzuraten, da diese von Haus aus kein Transaktionsmanagement mitbringen und über unflexible, vorgefertigte und schlecht wartbare Zugriffsmethoden verfügen.

Tableadapter führen SQL-Anweisungen und gespeicherte Prozeduren gegen Datenbanken aus, und ermöglichen so den Datenaustausch zwischen Anwendungen und Datenbanken. Tableadapter sind der von Microsoft vorgegebene Standardmechanismus zur Kommunikation mit einer Datenbank. Weiterhin werden anstatt Businessobjekten Datatables zurückgeliefert. Dies erfordert wiederum Aufwand, um aus den einzelnen Zeilen der zurückgelieferten Datatable wieder Businessobjekte zu erzeugen.

#### ADO.NET - Parametrisierung von SQL-Statements

Bei der Migration von der Oracle-Datenbank nach Microsoft SQL Server 2005 (siehe auch: Datenbanken) tritt das Problem auf, dass beide unterschiedliche Zeichen für das Escaping von Parametern in SQL-Strings verwenden. Oracle verwendet hier den Doppelpunkt (":") wobei der SQL-Server ein "@" erwartet. Hierdurch müssen zusätzlich zum Connection String auch noch alle parametrisierten SQL-Statements angepasst werden.

#### Kombination mehrerer Entwurfsmuster

Es hat sich als nicht empfehlenswert herausgestellt eine Fülle von Entwurfsmustern gleichzeitig einzusetzen, da diese durch Interferenzen die Wartbarkeit verschlechtern und die Applikation dadurch fehleranfälliger werden kann als beim gezielten Einsatz eines einzelnen Entwurfsmusters. Dies zeigte sich im Projektverlauf bei der gemeinsamen Implementierung der Identity Map und des Ghost Patterns. Beide Entwurfsmuster sind in den einzelnen Geschäftsobjekten angesiedelt und müssen deshalb für jedes

einzelne Objekt implementiert werden. Hierbei schlichen sich häufig Fehler ein, die die Implementierungszeit verlängerten.

Weiterhin ist dadurch der eigentliche Quelltext deutlich schwieriger zu verstehen, da man zunächst identifizieren muss welcher Quelltext von welchem Entwurfsmuster benötigt wird, bzw. für das jeweilige Entwurfsmuster geschrieben werden muss.

### Persistenzframeworks

**NHibernate - zusammengesetzte Primärschlüssel** Im speziellen Fall von NHibernate (hier vorliegend in Version 1.2.0.3001) hat sich schon von Beginn an gezeigt, dass es ein Problem gibt, wenn eine Tabelle einen zusammengesetzten Primärschlüssel (composite key) besitzt. Diese Problematik wurde durch spezielle Primärschlüsselklassen umgangen.

**NHibernate - Remoting** Remoting scheint nur schwer mit dem Lazy-Loading Mechanismus von NHibernate vereinbar zu sein. Da beim Remoting lediglich Proxies für die Objektlisten vom Datamapper-Server zum Client übertragen werden und dann, sobald dort ein Zugriff erfolgt, versucht wird eine Anfrage (Query) für diese Objektliste abzusetzen kommt es zu einem Fehler. Die NHibernate Session läuft nicht auf dem Client sondern auf dem jeweiligen Server und somit konnten die Objekte nicht gefunden werden. Es gibt hier die Möglichkeit eines Workarounds, indem man einfach bei allen remote verfügbaren Objektlisten das Attribut von *lazy* auf *false* setzt. Hiermit hat man dann allerdings die entsprechenden Performance-Einbußen. Eine weitere Möglichkeit besteht darin eine Handimplementierung für das Serialisieren einer solchen Objektliste zu erstellen. Dies bedeutete allerdings einen erheblichen Mehraufwand für die Entwicklung und Wartung.

**NHibernate - keine Primärschlüssel** Bei der gegebenen Altdatenbank gab es auch Tabellen, die komplett ohne Primärschlüssel definiert wurden. Auch damit kam NHibernate in der verwendeten Version nicht zurecht. Dies führte zur Änderung einiger Tabellen in der Datenbank. Hier wurden die Primärschlüsseldefinitionen nachträglich durchgeführt. Des Weiteren zeigte sich beim Entwickeln mit NHibernate, dass die auftretenden Fehlermeldungen oftmals an der eigentlichen Ursache des Fehlers vorbeizugehen und somit überhaupt nicht hilfreich sind.

**Vanatec OpenAccess** Beim kommerziell vertriebenen Persistenzframework OpenAccess (OA) des Herstellers Vanatec ist es in der hier verwendeten Version (4.3.11.601) nicht möglich, ein einfaches Löschen großer Tabellen in der mitgelieferten Query Language zu formulieren. Statt dessen müssen Objekte der Daten erzeugt werden und diese dann durch Iterieren einzeln herausgelöscht werden.

### Modellgetriebener Ansatz

Hierbei muss auf jeden Fall mit zusätzlichem Aufwand zur Erlernung der Generatorsprache gerechnet werden um ein Profil/Generator zu erstellen. Der Break-Even-Point muss vorher bestimmt werden, um abschätzen zu können, ob der Mehraufwand lohnenswert ist. Weiterhin hat sich herausgestellt, dass mehr Ausnahmen im Profil bzw.

Generator notwendig sind, je "eigenartiger" die Altapplikation, bzw. Altdatenbank, ist. In diesem Kontext ist ein Einsatz von vorgefertigten Generatoren wenig sinnvoll. Für die Spezialfallbehandlung wäre zusätzlicher Mehraufwand einzuplanen. Bei den betrachteten Generatoren lieferte openArchitectureWare (oAW) die zufriedenstellendsten Ergebnisse, da hier die Generatorsprache recht einfach erlernbar und flexibel ist. Allerdings muss auch festgehalten werden, dass bei hochgradig anpassbaren Generatorsystemen (siehe Kapitel 6) der Break-Even eher erreicht wird. Somit ist der Grad der Anpassbarkeit und somit auch die Möglichkeit vorhandene Generatoren wiederzuverwenden und den projektspezifischen Bedürfnissen anzupassen ein entscheidender Punkt in der Wahl des Ansatzes.

### Datenbanken - Migration

Es wurde während des Projektverlaufs eine Migration der Datenbank durchgeführt. Es wurde von einer Oracle 9.2 Datenbank auf einen Microsoft SQL Server 2005 migriert. Dies wurde mit Hilfe eines Migrationswerkzeugs von Microsoft erledigt.

Bei diesem Umzug gab es einige Probleme. Bei Oracle und dem SQL Server gibt es jeweils verschiedene, reservierte Schlüsselwörter. Vor allem der in der Altdatenbank oft benutzte Spaltenname *value* ist im Microsoft Umfeld ein reserviertes Wort und sorgt somit für Probleme. Ein Umbenennen von *value* nach *wert* konnte hierbei Abhilfe schaffen, sorgte jedoch für nachträglichen Wartungsaufwand in den implementierten Mappings / den Managern die das Objektmodell benutzen.

Das Grundproblem stellt die Datenbank selbst dar, da diese relational und nicht objektorientiert entwickelt wurde und ausserdem historisch gewachsen ist. Des Weiteren ist die Datenbank hoch generisch was den Umgang gewöhnungsbedürftig macht. Beispielsweise wurden in manchen Tabellendefinitionen keine Primärschlüsselspalten definiert oder es gab nicht intuitiv nachvollziehbare Relationen zwischen Tabellen oder es wurde eine nicht überschaubare Menge an Spalten in die Primärschlüsseldefinition mit einbezogen.

## 8.2. Entwurfsentscheidungsmatrix

Die Entwurfsentscheidungsmatrix soll eine Hilfestellung geben, bei welchem Einsatzszenario welche Technologie bevorzugt zum Einsatz kommen kann. Die Ergebnisse beziehen sich auf die Aspekte Performance und Wartbarkeit der Applikation. Die aufgezeigten Ergebnisse in der Matrix (siehe 8.2) beruhen auf den erarbeiteten Forschungsergebnissen.

Die Ergebnisse der Matrix dürfen nicht als absolut betrachtet werden, eher als eine Empfehlung, in welche Richtung die Entwicklung / Migration gehen soll. Dies wird anhand der untersuchten Eigenschaften des Systems festgelegt.

Auf der Ordinate werden der Projekttyp, die Projektgröße und das Nutzungsprofil abgetragen, auf der Abszisse die verschiedenen Technologien. Die Projekttypen werden hierbei unterschieden in:

- Neuentwicklung
- Wartung eines Altsystems
- (Teil-)Migration eines Altsystems

Die Projektgröße wird in groß und klein unterteilt, wobei hier die Unterteilung anhand dreier Faktoren bestimmt werden kann. Bei einer Neuentwicklung steht die aus der Spezifikation gewonnene Komplexität sowie die Anzahl an Datenbanktabellen im Vordergrund. Daraus lassen sich aber auch Rückschlüsse auf die zukünftige Anzahl an Klassen, bzw. Lines of Code ziehen.

Bei Wartungsarbeiten an einem Altsystem ist die Komplexität (die Lines of Code sowie die Anzahl der vorhandenen Tabellen in der Datenbank) ausschlaggebend zur Bestimmung. Als *Richtwerte* für die Anzahl an Tabellen in der Datenbank kann man grob sagen, dass ein Projekt mit kleiner gleich 10 Tabellen eher als klein einzustufen ist, und eine höhere Anzahl an Tabellen ein Indikator für ein eher großes Projekt darstellt. In diesem Kontext darf natürlich die Komplexität nicht vernachlässigt werden. Auch eine geringe Anzahl an Tabellen kann, je nach zu Grunde liegendem Entwurf, eine hohe Komplexität beherbergen.

Im Falle einer Teilmigration / Migration läuft die Bestimmung, ob groß oder klein, ähnlich wie bei den zuvor aufgeführten Wartungsarbeiten. Hier spielt die Anzahl der Lines of Code, die Anzahl der vorhandenen Tabellen sowie die vorhandene Komplexität eine Rolle.

Das Nutzungsprofil unterscheidet die Richtung des Datenzugriffs (lesen und schreiben), die Art des Zugriffs (Einfach- oder Mehrfachnutzung) sowie die Attribute Zuverlässigkeit, Sicherheit und Echtzeitfähigkeit.

Auf der Abszisse wird die Technologie abstrahiert. Hier wird zwischen einer manuellen Implementierung, dem Einsatz eines OR-Mappers (NHibernate, Vanatec OpenAccess), dem Einsatz von Wizards, modellgetriebener Entwicklung (MDA), Adaptergenerierung (MINT Ansatz) sowie einem sogenannten Double Mapping unterschieden.

Je nachdem, welche Attribute auf der Ordinate ausgewählt sind, gibt es Erfahrungswerte, welche Technologie bei welchem Einsatzszenario zu bevorzugen ist.

Den Anmerkungen mit \*, \*\*, \*\*\* und \*\*\*\* ist folgende Bedeutung zugeordnet:

*	Sofern eine hohe Diskrepanz zwischen Altsystem und migriertem System vorhanden ist.
**	Im Falle der Performance von einzelnen Aktionen.
***	Viele OR Mapper sind nicht Thread-safe.
****	Mehrfachaufwand durch double mapping, aber auch Vorteile durch Zwischenspeicherung (Caching).

Zur Auswertung, bzw. Empfehlungsaussprechung wird ein 5-stufiges Bewertungssystem mit den Werten -, -, neutral, + und ++ zu Grunde gelegt, wobei - als nicht empfehlenswert interpretiert wird und ++ als vorziehbar gegenüber eines anderen Ansatzes. Im Falle der im Projektumfeld betrachteten Applikationen kamen alle drei

		Technologie					
		manuell	OR-Mapper	Wizards	MDA	Adapter-generierung (MINT)	double Mapping
<b>Einsatzszenario</b>	<b>Projekttyp</b>						
	Neuentwicklung	-	+	+	++		
	Wartung Altsystem (Teil-)Migration Altsystem	+	-	-	-	+	-
			+	++	+	++	+*
	<b>Projektgröße</b>						
	klein	++		+	-	-	-
	groß	-	+	+	++	++	+
	<b>Nutzungsprofil</b>						
	lesen	+	-			+	-
	schreiben	-	+			+	+**
	Einzelbenutzer	+	+			+	-
	massiver paralleler Zugriff	-	-***		+	+	+
	Sicherheit	-	-		++	++	
	Zuverlässigkeit	-		+	+	+	+
Echtzeitfähigkeit	+	-			+	+/_****	

Abbildung 8.2.: Entwurfsentscheidungsmatrix

Projekttypen zum Einsatz. Bei der Chartproduction erfolgte eine Migration eines kleinen Projekts, bei der Accountingapplikation erfolgte eine Teilmigration eines großen Projekts und das Importtool repräsentiert quasi die Neuentwicklung eines Miniprojekts.

Im Zuge der Durchführung der Implementierung dieser drei Applikationen haben sich die in der Matrix aufgezeigten Ergebnisse entwickelt.

**Migration - Chartproduction** Die Chartproduction-Applikation repräsentiert im Projektkontext eine Migration einer Altanwendung. Da dies ein relativ kleines Projekt darstellt, wurde zuerst eine Handimplementierung für die Datenbankanbindung durchgeführt. Die Applikation liest nur Daten aus der Datenbank. Dies erwies sich als recht einfach und gut durchführbar.

Im nächsten Ansatz wurde der Versuch unternommen, die Datenbankanbindung mit Hilfe eines OR-Mappers zu erstellen. Hierbei zeigte sich, dass es einen gewissen Mehraufwand gibt, um den OR-Mapper zum Funktionieren zu bewegen. Da in dieser Anwendung aber nur aus der Datenbank gelesen wird, ist der Mehraufwand für die Implementierung mit OR-Mapper nicht unbedingt den Aufwand wert. Im Falle der (Teil-)Migration eines großen Projekts, wie bei der Accounting-Anwendung, ist der Einsatz eines OR-Mappers sinnvoller, da der durch den OR-Mapper erzeugte Overhead an Lines of Code nicht so sehr ins Gewicht fällt, im Vergleich zu der (Teil-)Migration eines kleinen Projekts.

Auch ist der MDA-Ansatz für die Migration eines kleinen Projekts nicht empfehlenswert, da hierbei sowohl der Zusatzaufwand zur Erlernung der Generatorsprache als

auch die Zeit zur Erstellung eines Generators, auch wenn die Generatorsprache schon erlernt wurde, zu groß ist. Dies äußerte sich in der Zeit, die für die Implementierung benötigt wird.

Der MINT-Ansatz ist diesem Kontext ähnlich ungeeignet wie der MDA-Ansatz, da es auch hier zuviel Zusatzaufwand gibt.

Das Double-Mapping ist bei der Migration eines kleinen Projektes nur empfehlenswert, falls eine *sehr große* Diskrepanz zwischen Altsystem und migriertem System vorhanden ist. Der Einsatz von sogenannten Wizards (Assistenten) ist hier auch sinnvoll. Der kommerzielle OR-Mapper OpenAccess bringt ein Werkzeug mit, mit dem man ein Reverseengineering einer Datenbanktabelle durchführen kann. Aus den aus der Datenbank gewonnenen Informationen wird ein mit Annotationen versehenes POCO (plain old C# objects) samt Mapping erstellt. Die Mappingdatei ist, genau wie bei NHibernate, im XML Format.

**Teilmigration - Accounting-Anwendung** Die Accounting-Applikation repräsentiert eine Teilmigration eines großen Projekts. Es ist vorab anzumerken, dass hierbei eine sehr hohe Komplexität der verwendeten Tabellen der Altdatenbank vorliegt. Es wurde auch nicht der komplette Teil des Accountings der Altdatenbank umgesetzt, da dies sehr wahrscheinlich den Projektumfang gesprengt und keine nennenswerten Ergebnisse für das MINT-Forschungsprojekt geschaffen hätte.

Eine Implementierung ohne jegliche Hilfsmittel (OR-Mapper, Wizard, MDA) ist nur bedingt empfehlenswert, da die gemessene Implementierungszeit ohne Hilfsmittel größer war, als die Implementierungszeit mit Hilfsmitteln. Weiterhin spricht die Tatsache, dass auch in die Datenbank geschrieben werden soll, für die zur Hilfenahme eines OR-Mappers, bzw. für die Adaptergenerierung, da es sich gezeigt hat, dass es in diesem Kontext bei einer Handimplementierung öfters zu Fehlern bei der Wartung / Erweiterung kommt, als bei der zur Hilfenahme bspw. eines OR-Mappers.

Weiterhin zeigte sich in diesem Kontext, dass es bei den verwendeten OR-Mappern, in den vorliegenden Versionen, Probleme beim Parallelzugriff gibt, da diese nicht Thread-safe zu sein scheinen. Möglicherweise ist dieses Verhalten in späteren Programmversionen korrigiert.

Das Doublemapping wurde in dem Kontext dieser Applikation evaluiert. Es stellte sich als empfehlenswert im Falle von Performance von einzelnen Aktionen heraus. Außerdem wird hiermit Domänenwissen in das Objektmodell integriert. Dies dient zur besseren Kommunikation mit den Entscheidungsträgern auf Kundenseite, sowie dem Anwender aus der spezifischen Domäne.

**Neuentwicklung / Prototyping - ImportTool** Das ImportTool repräsentiert eine Neuentwicklung eines kleinen Projekts. Diese Applikation soll XML-Daten für die ChartProduction-Applikation in die Datenbank schreiben. Der Hauptfokus liegt hierbei auf der Performance des Schreibens vieler Daten. Außerdem wurde der Versuch unternommen, von mehreren Instanzen der Applikation gleichzeitig XML-Daten in die Datenbank zu schreiben.

Zu Beginn wurde eine Handimplementierung evaluiert, was in diesem Kontext als sinnvoll erachtet wurde, da nur eine kleine Menge von Tabellen aus der Datenbank beteiligt sind. Der Einsatz eines OR-Mappers erzeugte in diesem Kontext Mehraufwand,

genauso wie der MDA-Ansatz.

Durch Abspalten mehrerer Threads mit jeweils einer Instanz des ImportTools wurde der Parallelzugriff beim Schreiben auf die Datenbank evaluiert. Beim Einsatz der OR-Mapper zeigte sich, dass die beiden eingesetzten OR-Mapper in den vorliegenden Versionen nicht Thread-safe sind. Bei einer Neuentwicklung wie dieser, ist der Einsatz eines Double-Mappings nicht wirklich sinnvoll, da hierbei das Domänenwissen schon bei der Entwicklung des neuen Objektmodells einfließen sollte.



## 9. Fazit

*Cord Giese, Thomas Goldschmidt, Steffen Kruse, Ralf Reussner,  
Ulrike Steffens, Niels Streckmann, Malte Zilinski*

Der vorliegende Bericht beschreibt die Ergebnisse des MINT-Projekts, das von den Partnern andrena objects AG, BTC AG, Delta Software Technology, FZI Forschungszentrum Information, OFFIS und der Universität Oldenburg durchgeführt und vom Bundesministerium für Bildung und Forschung gefördert wurde. Ziel des Projekts war die Entwicklung und Evaluierung modellgetriebener Methoden für die Integration von Informationssystemen. Schwerpunkte des Projekts waren die fachliche Modellierung von Integrationsaufgaben, die Entwicklung von Verfahren und Werkzeugen zur modellgetriebenen Integration auf Prozess- und Datenebene sowie die Evaluierung dieser Verfahren und die Ableitung von Best Practices und Mustern zur Unterstützung von Entwurfsentscheidungen.

Zur fachlichen Modellierung von Geschäftsprozessen mit Schwerpunkt auf der Integration von bestehenden Systemen wurde eine domänenspezifische Sprache namens MINT-XL (siehe Kapitel 3) entworfen und entsprechende Werkzeuge für die Nutzung der Sprache implementiert. Neben dem Kern der Sprache, der die Domäne *Prozessintegration* unterstützt, liegt der Hauptvorteil der Sprache auf der Erweiterbarkeit für verschiedene Anwendungsdomänen. Diese Erweiterbarkeit gewährleistet, dass die Anforderungen von Fachexperten der jeweiligen Domänen in einer auf diese angepassten Sprache aufgenommen werden können und damit Kommunikationsprobleme und Missverständnisse, die erheblichen Einfluss auf die spätere Entwicklung haben können, vermieden werden. Prototypisch wurde eine solche Erweiterung für die Domäne *Störungsmanagement* durchgeführt.

Zusätzliche Erweiterungen für andere Anwendungsdomänen sind für zukünftige Arbeiten denkbar. Eine Möglichkeit für eine solche Erweiterung, die auch bereits teilweise begonnen wurde, ist die in Abschnitt 2.7.2 beschriebene Instandhaltungsdomäne. Des Weiteren sind für den produktiven Einsatz weitere Evaluierungen der Werkzeuge nötig. Insbesondere ist hierbei die Erweiterbarkeit der bestehenden Implementierung des MINT-PIE-Editors auf neue Domänen zu nennen. Zusätzlich wäre eine integrierte Lösung zur Unterstützung bei der Modellierung mit diversen Mustern für den produktiven Einsatz bei der Weiterentwicklung des MINT-PIE-Editors sinnvoll. Hierfür kommen sowohl allgemeine Prozessmuster als auch die in Kapitel 4 verwendeten kognitiven Muster in Frage. Eine erste Evaluation der möglichen Ansätze hat jedoch gezeigt, dass der Arbeitsaufwand für eine zufriedenstellende Lösung nicht unterschätzt werden darf, da komplexe Aufgaben im Bereich der Mustererkennung und Graphtransforma-

tionen anfallen. Hier kann jedoch ein lohnenswerter Ansatzpunkt für weitere Arbeiten gesehen werden.

Neben der MINT-XL wurden in MINT Methoden entwickelt, die kognitive Muster aus dem Bereich der Künstlichen Intelligenz nutzen, um Fachexperten bei der Modellierung ihrer Anforderungen zu unterstützen (Kapitel 4). Diese Unterstützung wurde als Vorgehensmodell für die Modellierung und Umsetzung von wissensintensiven Prozessen im Rahmen der MDA vorgestellt. Es wurde gezeigt, wie Metamodelle für die verschiedenen MDA-Ebenen erzeugt werden können und wie diese iterativ verfeinert werden, so dass sie den gewünschten Anforderungen genügen. Im Anwendungsbeispiel der Studienplanung wurde der vorgestellte Prozess bis zum Zielsystem durchgeführt.

Es stellte sich die Herausforderung, die Probleme auf PIM-Ebene als generische Scheduling-Probleme zu repräsentieren. Eine allgemeine Transformation für die entsprechende Scheduling-Klasse ist möglich, erzeugt aber ein berechnungskomplexes Problem. Als Anknüpfungspunkt für weitere Arbeiten wäre eine Bibliothek von Transformationen sinnvoll, die eine Menge von Domäneneigenschaften berücksichtigt und somit Heuristiken für die automatische Lösung bereitstellen könnte. Dafür müsste allerdings eine Zielplattform gewählt werden, die eine Steuerung der Suche zulässt (zum Beispiel Constraint Satisfaction Problems).

Eine Besonderheit für die Lösung von Planungsproblemen und somit eine Anforderung für das Zielsystem ist, dass sich das Scheduling-Modell erst zur Laufzeit vervollständigen lässt und aufgrund dieser Tatsache dann zu einer PDDL-Spezifikation transformiert wird. Es ist zu beachten, dass verschiedene Klassen von Scheduling-Problemen existieren, für die gegebenenfalls sehr unterschiedliche Transformationen definiert werden müssen. So unterscheidet man zum Beispiel grundsätzlich präemptives und nicht-präemptives Scheduling. Für eine Lösung von nicht-präemptiven Scheduling-Problemen kann - wie gezeigt - PDDL verwendet werden, während es sich für präemptive Probleme weniger gut eignet, da „Actions“ im Vorfeld als Zerlegungen entsprechend der Zeitschlitze definiert werden müssen. Insgesamt wäre eine Beobachtung für mögliche alternative Zielplattformen auf PSM-Ebene für Scheduling-Modelle interessant.

Für andere wissensintensive Prozesse (siehe Abbildung 4.1) wäre eine Konkretisierung für die Nutzung von Mustern mit entsprechender Transformation über die verschiedenen MDA-Ebenen sicherlich wertvoll und könnte ein Thema eines Nachfolgeprojekts darstellen. So auch in der Windparkdomäne (siehe Abschnitt 2.7.2), da auch in dieser wissensintensive Probleme im Bereich der Planung auftreten.

Das entwickelte Vorgehensmodell zur modellgetriebenen Prozessintegration (siehe Kapitel 5) baut auf der MINT-XL als Sprache für die fachliche Modellierung auf und nutzt die darin beschriebenen Modelle als Basis für die modellgetriebene Entwicklung eines Integrationssystems. Diese Modelle werden (angelehnt an bestehende Standards) in technische Architektur-, Prozess- und Abbildungsmodelle überführt. Hieraus lassen sich über mehrere Schritte hinweg in einem iterativen Prozess Softwareartefakte wie ausführbare Prozessbeschreibungen und Adapter für bestehende Systeme generieren. Neben den fachlichen Modellen fließen in diesen Generierungsprozess auch Modelle der Schnittstellen der bestehenden Systeme ein, auf die in den genannten Modellen Bezug genommen wird, um die Integration der Systeme auf Modellebene beschreiben

---

zu können.

Anknüpfungspunkte für zukünftige Arbeiten sind vor allem in der Anpassung auf weitere domänenspezifische Besonderheiten und Plattformen zu sehen. Um diese in einer produktiven Umgebung ideal unterstützen zu können, müssen auf den Kontext angepasste UML-Profile für die technischen Architekturmodelle erstellt und die bestehenden Modell-zu-Modell-Transformationen um die entsprechende Logik erweitert werden. Entsprechende Erweiterungsmechanismen sind in der bestehenden prototypischen Implementierung bereits vorgesehen. Daneben sind Templates für die Code-Generierung für weitere Plattformen und ggf. zusätzliche Software-Artefakte neben den bereits bestehenden Implementierungen für z. B. ausführbare Prozessbeschreibungen und Webservice-Adapter notwendig.

Das zweite Anwendungsszenario des Projekts bildet die modellgetriebene Datenintegration (vgl. Kapitel 6). Das hierbei entwickelte Verfahren baut ebenfalls auf der Model Driven Architecture (MDA) der OMG auf, und hat die Generierung von Persistenzadaptern für Datenbanken zum Ziel. Im Projekt lag der Schwerpunkt auf der Kopplung objektorientierter Client-Applikationen an relationale Legacy-Datenbanken. Im Detail implementieren die generierten Adapter die Abbildung von (anwendungsorientierten) Service Interfaces auf DB-Operationen. Die Adapter selbst bestehen aus zwei Schichten: einer Datenzugriffsschicht und einer Service-Schicht. Die Datenzugriffsschicht kapselt die technischen Eigenheiten des DBMS und der eingesetzten Zugriffstechnik bzw. Middleware und bietet eine (interne) Schnittstelle, die sich am verwendeten Ausschnitt des DB-Schemas orientiert. Die Service-Schicht implementiert die semantische Abbildung (das „Mapping“) einer objektorientierten Service-Schnittstelle auf Operationen der Datenzugriffsschicht. Die Anwendung kennt nur die Service-Schnittstelle. Auf diese Weise realisieren die generierten Adapter eine technische und eine semantische Abstraktion. Während die zu dem Verfahren gehörende Werkzeugkette einen möglichst hohen Automatisierungsgrad anstrebt, gibt es auf der PIM- und der PSM-Ebene jeweils manuelle Eingriffsmöglichkeiten, die praktischen Aspekten geschuldet sind. Dies betrifft den Import vorhandener DB-Schemata sowie Details der semantischen Abbildung in der Service-Schicht.

Im Projekt wurde dieses Datenintegrationsverfahren anhand mehrerer Applikationen und einer produktiv eingesetzten Datenbank evaluiert. Dabei zeigte sich, dass die – spezifisch für die jeweilige Anwendung und die eingesetzte Datenbank – generierten Adapter sehr leistungsfähig sind (vgl. Kapitel 7). Es ist hinzuzufügen, dass im Projekt weder die Anwendungen, noch das verwendete Objektmodell noch das DB-Schema für die Integration geändert werden mussten.

Da sowohl das Verfahren als auch die entsprechenden Werkzeuge (Abschnitt 2.5) einschließlich der Adapter-Generierung während des Projekts weiterentwickelt worden waren, konnten noch keine exakten Aussagen zum Entwicklungsaufwand, zur Wartbarkeit und zum Testaufwand beim praktischen Einsatz gemacht werden. Hier könnte ein künftiger Vergleich mit konkurrierenden generischen Ansätzen die vielversprechenden Aussichten verifizieren.

Die Wartung hat im Rahmen der Softwareentwicklung großen Einfluss auf die Kosten eines Softwareprodukts. Oft wird die notwendige Qualität aufgrund von Zeitmangel

bei Fertigstellung des Produkts nicht erreicht, was zu erheblichem Wartungsaufwand nach dessen Auslieferung führt. Die modellgetriebene Softwareentwicklung ist ein viel versprechender Ansatz, der auf höherer Abstraktionsebene als quelltextgetriebene Softwareentwicklung den Entwurf von Softwareprodukten erlaubt. Die starke Integration der Artefakte, die während der Entwicklung entstehen, vermindert das Risiko von Inkonsistenzen während der Entwicklung. Trotz der inhaltlichen Verbesserungen können jedoch auch Softwareprodukte, die modellgetrieben erstellt wurden, qualitativ minderwertig sein, was zu erheblichen Kosten der Wartung führen kann.

In MINT wurden bereits einige Ansätze zur Evaluierung der Wartbarkeit in der modellgetriebenen Entwicklung untersucht. Allerdings stellte sich heraus, dass es schwierig ist vorhandene Metriken (z. B. von objektorientierter Softwareentwicklung) auf dieses Paradigma zu übertragen. Somit ist hier ein weiteres Forschungsgebiet eröffnet worden. Hierbei dienen die in MINT gewonnenen Ergebnisse als Grundlage für weitere Forschungsarbeit. Ein erster Schritt in diese Richtung wurde bereits während des Projekts im Kontext einer Diplomarbeit durchgeführt[35].

Die Evaluierung der Vorgehens (siehe Kapitel 7) konzentrierte sich auf das Anwendungsszenario der Datenintegration. Hier wurde der beschriebene MINT-Ansatz mit anderen zur Verfügung stehenden Konzepten und Werkzeugen zur Datenintegration verglichen. Zu diesem Zweck wurde ein GQM-Plan erstellt, auf dessen Basis ein systematischer Vergleich durchgeführt wurde. Anhand der erzielten Ergebnissen und der Erfahrung bei der Umsetzung der verschiedenen Methoden an einem Realsystem aus dem Finanzbereich, wurden Best Practices und Muster abgeleitet (vgl. Kapitel 8). Diese und eine aus den Ergebnissen der Evaluation erstellte Entwurfsentscheidungsmatrix sollen Architekten und Entwickler, die vor dem Problem der Auswahl eines Ansatzes für die Datenintegration stehen, unterstützen einen geeigneten Ansatz auszuwählen.

## A. Veröffentlichungen

### 2006

Ralf Reussner, Ulrike Steffens und Niels Streekmann. *MINT - Modellgetriebene Integration von betrieblichen Informationssystemen*. In: Tagungsband zur Statuskonferenz der Forschungsoffensive „Software Engineering 2006“, Leipzig, Juni 2006.

Niels Streekmann, Ulrike Steffens, Claus Möbus und Hilke Garbe. *Model-Driven Integration of Business Information Systems*. In: Softwaretechnik-Trends, 26(4):9-13, November 2006.

Jürgen Meister und Heiner Feislachen. *Evaluation von MDA-Werkzeugen* (Technischer Bericht). Oldenburg, 2006.

### 2007

Mathias Uslar, Niels Streekmann und Sven Abels. *MDA-basierte Kopplung heterogener Informationssysteme im EVU-Sektor*. In: eOrganisation: Service-, Prozess-, Market-Engineering, 8. Internationale Tagung Wirtschaftsinformatik, 2: 965-982, Karlsruhe, Februar 2007.

Thomas Goldschmidt, Jochen Winzen und Ralf Reussner. *Evaluation of Maintainability of Model-driven Persistency Techniques*. In: IEEE CSMR 07 - Workshop on Model-Driven Software Evolution (MoDSE2007), Amsterdam, März 2007.

Ulrike Steffens, Jan Stefan Addicks und Niels Streekmann (Hrsg.). *MDD, SOA und IT-Management (MSI 2007)*. GITO-Verlag, April 2007.

### 2008

Michael Goedicke, Maritta Heisel, Sascha Hunold, Stefan Kühne, Matthias Riebisch und Niels Streekmann. *Workshop Modellgetriebene Softwarearchitektur - Evolution, Integration und Migration*. In: Software Engineering 2008. Februar 2008.

Michael Goedicke, Maritta Heisel, Sascha Hunold, Stefan Kühne, Matthias Riebisch und Niels Streekmann. *Workshop Modellgetriebene Softwarearchitektur - Evolution, Integration und Migration*. In: Software Engineering 2008 - Beiträge zu den Workshops. März 2008.

Dieter Hildebrandt, Michael Gründler und Heiner Feislachen. *Modellgetriebene Integration von Altsystemen*. In: Software Engineering 2008 - Beiträge zu den Workshops. März 2008.

Steffen Kruse, Malte Zilinski, Hilke Garbe und Claus Möbus. *MDA und KI: Domänenspezifische Modellierung und Umsetzung wissensintensiver Prozesse*. In: Software Engineering 2008 - Beiträge zu den Workshops. März 2008.

Niels Streekmann und Wilhelm Hasselbring. *Towards Identification of Migration Increments to Enable Smooth Migration*. In: Proceedings of the First International Workshop on Model-Based Software and Data Integration (MBSDI 2008). Springer-Verlag, April 2008.

Thomas Goldschmidt, Ralf Reussner und Jochen Winzen. *A Case Study Evaluation of Maintainability and Performance of Persistency Techniques*. In: Proceedings of the Proceedings of the 30th International Conference on Software Engineering (ICSE 2008): 401-410. ACM, April 2008.

Sven Abels, Wilhelm Hasselbring, Niels Streekmann und Mathias Uslar. *Model-Driven Integration In Complex Information Systems: Experiences From Two Scenarios*. In: Model-Driven Software Development: Integrating Quality Assurance. IGI Global, August 2008.

Ulrike Steffens, Jan Stefan Addicks und Niels Streekmann (Hrsg.). *MDD, SOA und IT-Management (MSI 2008)*. GITO-Verlag, September 2008.

Achim Baier, Steffen Becker, Martin Jung, Klaus Krogmann, Carsten Röttgers, Niels Streekmann, Karsten Thoms und Steffen Zschaler. *Modellgetriebene Software-Entwicklung*. In: Handbuch der Software-Architektur, 2. Auflage, dpunkt.verlag, Dezember 2008.

Wilhelm Hasselbring, Stefan Krieghoff, Ralf Reussner und Niels Streekmann. *Migration der Architektur von Altsystemen*. In: Handbuch der Software-Architektur, 2. Auflage, dpunkt.verlag, Dezember 2008.

Wilhelm Hasselbring, Stefan Krieghoff, Ralf Reussner und Niels Streekmann. *Migration eines Altsystems zu einer Java Enterprise Architektur*. In: Handbuch der Software-Architektur, 2. Auflage, dpunkt.verlag, Dezember 2008.

Dieter Hildebrandt, Oliver Holschke, Philipp Offermann und Ulrike Steffens. *Grundlagen serviceorientierter Architekturen*. In: Handbuch der Software-Architektur, 2. Auflage, dpunkt.verlag, Dezember 2008.

# Abbildungsverzeichnis

1.1. Arbeitspakete des MINT-Projekts . . . . .	5
1.2. Übersicht zum MINT-Ansatz . . . . .	8
2.1. MDA-Überblick . . . . .	12
2.2. Erweiterter MDA-Ansatz . . . . .	13
2.3. Das Dublo-Muster [27] . . . . .	14
2.4. Der BALES-Ansatz [30] . . . . .	16
2.5. Der BALES-Ansatz und MDA [28] . . . . .	17
2.6. Überblick über oAW . . . . .	22
2.7. Workflow DSL Tools [39] . . . . .	24
2.8. Schichtenarchitektur von SCORE . . . . .	29
2.9. Datenfluss des Schema-Imports . . . . .	30
2.10. Datenfluss der Adapter-Definition . . . . .	30
2.11. Datenfluss der Adapter-Generierung . . . . .	31
2.12. Das SCORE-Datenmodell . . . . .	32
2.13. Abstraktion und Scope von Generatoren . . . . .	33
2.14. Codegenerierung und MOF . . . . .	34
2.15. HelloWorld-Implementierung für C . . . . .	35
2.16. SAP Netweaver Komponentenansicht (Quelle: SAP AG) . . . . .	37
2.17. Architektur von SAP Visual Composer for NetWeaver (Quelle: [9]) . . . . .	39
2.18. Source iView . . . . .	42
2.19. Nested iView . . . . .	43
2.20. Ansicht „Customer“ . . . . .	44
2.21. Ansicht „Address“ . . . . .	45
2.22. Architektur der Testumgebung . . . . .	50
3.1. <i>MINT-XL Sprachstruktur</i> . . . . .	55
3.2. <i>MINT-XL Struktur-Paket</i> . . . . .	56
3.3. <i>Process Oriented Integration Package</i> . . . . .	57
3.4. <i>Erweiterungsmechanismus der MINT-XL</i> . . . . .	58
3.5. Mappings im MINT-XL Metamodell . . . . .	60
3.6. Beispiel für ein Mapping in der MINT-XL . . . . .	61
3.7. Das <i>Fault Support</i> -Paket . . . . .	62
3.8. Anwendungsbeispiel Störungsmanagement I . . . . .	64
3.9. Anwendungsbeispiel Störungsmanagement II . . . . .	65
3.10. Anwendungsbeispiel Störungsmanagement III . . . . .	66
3.11. MINT-PIE Editor . . . . .	69

---

4.1. Übersicht CommonKADS: wissensintensive Prozesse [64, S. 125] . . . . .	72
4.2. Geschäftsprozess der Studienplanung [34] . . . . .	73
4.3. Instanziierung des abstrahierten TKM Planning in MINT-XL [34] . . . . .	74
4.4. Metamodellerzeugung . . . . .	75
4.5. Metamodell zur Modellierung der statischen Aspekte der Studienordnung	76
4.6. Scheduling Metamodell . . . . .	78
4.7. Zielsystem . . . . .	80
4.8. Web-Client Studienplanung . . . . .	81
5.1. Übersicht zum MINT-Vorgehensmodell . . . . .	83
5.2. Fachliches Modell im MINT-Vorgehensmodell . . . . .	84
5.3. Beispiel für ein fachliches Mapping . . . . .	85
5.4. Auszug aus dem plattformabhängigen Architekturmodell . . . . .	89
5.5. Auszug aus dem Altsystemmodell . . . . .	92
5.6. Das technische Mapping im MINT-Vorgehensmodell . . . . .	93
5.7. ILF-Paket des Metamodells für die technischen Mappings . . . . .	93
5.8. Beispiel für ein technisches Mapping . . . . .	94
5.9. Übersicht der CIM-zu-PIM-Transformation . . . . .	98
5.10. Der Workflow vom fachlichen Modell zum plattformunabhängigen Ar- chitekturmodell . . . . .	101
5.11. Die Transformation vom CIM zum PIM im MINT-Vorgehensmodell . . . . .	102
5.12. Übersicht der PIM-zu-Annotationsmodell-Transformation . . . . .	103
5.13. Übersicht der PIM zu PSM Transformation . . . . .	104
5.14. Der Workflow vom plattformunabhängigen zum plattformabhängigen Architekturmodell . . . . .	105
5.15. Die Transformation vom plattformabhängigen Modell zu Code . . . . .	105
6.1. Entity-Datenobjekt <i>Currency</i> . . . . .	112
6.2. Anzeige der „Cross Reference“ zum Entity-Datenobjekt <i>Currency</i> . . . . .	112
6.3. Entity-Datenobjekt <i>ExchangeRate</i> , abgeleitet aus <i>IMPORT_CURRENCY_-</i> <i>PERFECT</i> . . . . .	113
6.4. Methodenparameter zu <i>ExchangeRate.getrec</i> . . . . .	113
6.5. Association-Objekte zu <i>ExchangeRate</i> . . . . .	114
6.6. Navigator-Objekt zu <i>WAEHRUNG</i> . . . . .	115
6.7. Methodenparameter zu <i>NavCurrency.load</i> . . . . .	115
6.8. Methodenparameter zu <i>StockIndex.getrec</i> . . . . .	115
6.9. Navigator-Objekt zu <i>BEZUGSINDIZES</i> . . . . .	116
6.10. Das Service Interface für <i>ChartProduction</i> . . . . .	117
6.11. „Interface Properties“ von <i>DAICurrency</i> . . . . .	118
6.12. Operationen des Interface <i>DAICurrency</i> . . . . .	118
6.13. Parameter der Operation <i>DAICurrency.get_Name</i> . . . . .	118
6.14. Operationen des Interface <i>DAIChartProductionApplicationFacade</i> . . . . .	119
6.15. Parameter der Operation <i>DAIChartProductionApplicationFacade.Get-</i> <i>ExchangeRate</i> . . . . .	120



---

6.16. Parameter der Operation <i>DAIChartProductionApplicationFacade.getShares</i>	120
6.17. Parameter der Operation <i>DAIShare.get_Recommendation</i>	120
6.18. Mapping eines Return-Parameters auf ein Datenelement eines Datenobjekts	121
6.19. Auswahl von Datenobjekt, Methode und Parameter	122
6.20. Feldauswahl im Methodenparameter	122
6.21. Operationen des Interface <i>DAIChartProductionApplicationFacade</i>	123
6.22. Mapping-Regel mit Embedded Coding	124
6.23. Eingabe von Embedded Coding	124
6.24. Auswahl eines Mapping-Pattern	125
6.25. Property <i>Company</i> der C#-Klasse <i>DAIShare</i>	126
6.26. Selektion des Entity-Namens im Mapping-Pattern <i>GetRelatedToOne</i>	126
6.27. Produktions-Implementierung des Patterns <i>GetRelatedToMany</i> im Pattern-Editor	127
6.28. Ausschnitt einer Pattern-Implementierung für den Fremdschlüssel-Zugriff	128
7.1. Schematische Darstellung - das Vorgehen	132
7.2. Accounting - Objektmodell	132
7.3. Ausschnitt aus Objekmodell - unternehmenszentriert	134
7.4. Ausschnitt aus Objekmodell - wertpapierzentriert	135
7.5. Die Struktur eines GQM-Planes (von Basili[3])	136
7.6. M2.2.3: Häufigkeitsverteilung der Zeiten für die Erstellung von Verlaufskurven.	147
7.7. M2.2.4: Speicherverbrauch während der Erstellung der Verlaufskurven.	148
8.1. Schematische Darstellung - double mapping	158
8.2. Entwurfsentscheidungsmatrix	162



# Tabellenverzeichnis

2.1. Bewertung der MDA-Werkzeuge . . . . .	27
2.2. Informationsquellen . . . . .	41
2.3. Evaluationsergebnis . . . . .	46
5.1. Abbildung der UML-Elemente des Komponentendiagramms auf C#- Elemente . . . . .	98
5.2. Mapping der MINT-XL-Elemente auf UML für Aktivitätsdiagramme .	99
5.3. Abbildung von UML-Elementen der dynamischen Sicht des plattform- abhängigen Architekturmodells auf BPEL . . . . .	106
7.1. Resultate der Wartbarkeitsevaluierung . . . . .	143
7.2. Ergebnisse der Evaluierung der Evolutionsszenarien . . . . .	144
7.3. Ergebnisse der Warbarkeitsevaluierung . . . . .	145
7.4. Ergebnisse der Performance-Evaluierung für Frage 2.1 und Anwendung 1: Chartproduction . . . . .	145
7.5. Ergebnisse der Performance-Evaluierung für Frage 2.2 und Anwendung 1: Chartproduction (Mittelwerte bis auf M2.2.4) . . . . .	146
7.6. M2.3.1 Mittelwerte der Ausführungszeiten der Chartproduction-Fassade in Millisekunden (ms). . . . .	148
7.7. M2.4.1Mittelwerte der Ausführungszeiten der <i>generischen</i> Chartproduction- Fassade in Millisekunden (ms) . . . . .	149
7.8. Ergebnisse der Performance-Evaluierung von Frage 2.1 für Applikation 2: Accounting in Millisekunden (ms) . . . . .	149
7.9. Ergebnisse der Performance-Evaluierung für Frage Q2.2 für Applikation 2: Accounting (alles Durschnittswerte ausser M2.2.4) in Millisekunden (ms) . . . . .	150



# Literaturverzeichnis

- [1] ANDREAS RENTSCHLER: *Model-To-Text Transformation Languages*. Seminararbeit an der Universität Karlsruhe, 2006.
- [2] ANDREWS, TONY, FRANCISCO CURBERA, HITESH DHOLAKIA, YARON GOLAND, JOHANNES KLEIN, FRANK LEYMANN, KEVIN LIU, DIETER ROLLER, DOUG SMITH, SATISH THATTE, IVANA TRICKOVIC und SANJIVA WEERAWARANA: *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM, 2003.
- [3] BASILI, V., G. CALDEIRA und H. D. ROMBACH: *Encyclopedia of Software Engineering*, Kapitel The Goal Question Metric Approach. Wiley, 1994.
- [4] BECKER, STEFFEN, VIKTORIA FIRUS, SIMON GIESECKE, WILHELM HASSELBRING, SVEN OVERHAGE und RALF H. REUSSNER: *Towards a Generic Framework for Evaluating Component-Based Software Architectures*. In: TUROWSKI, KLAUS (Herausgeber): *Architekturen, Komponenten, Anwendungen - Proceedings zur 1. Verbundtagung Architekturen, Komponenten, Anwendungen (AKA 2004), Universität Augsburg*, Band 57 der Reihe *GI-Edition of Lecture Notes in Informatics*, Seiten 163–180, 2004.
- [5] BECKER, STEFFEN, SVEN OVERHAGE und RALF H. REUSSNER: *Classifying Software Component Interoperability Errors to Support Component Adaption*. In: CRNKOVIC, IVICA, JUDITH A. STAFFORD, HEINZ W. SCHMIDT und KURT C. WALLNAU (Herausgeber): *Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings*, Band 3054 der Reihe *Lecture Notes in Computer Science*, Seiten 68–83, Berlin, Heidelberg, 2004. Springer.
- [6] BERGSTRA, JAN A. und PAUL KLINT: *The ToolBus Coordination Architecture*. In: CIANCARINI, P. und C. HANKIN (Herausgeber): *Coordination Languages and Models*, Band 1061 der Reihe *LNCS*, Seiten 75–88. Springer-Verlag, Berlin, Germany, 1996.
- [7] BUSCHERMÖHLE, RALF und WILHELM HASSELBRING: *An Approach for a UML Profile for Software Development Process Modeling*. In: *NWUML2004: 2nd Nordic Workshop on the Unified Modeling Language*, Seiten 167–184, June 2004.
- [8] BÉZIVIN, JEAN, GUILLAUME HILLAIRET, FRÉDÉRIC JOUAULT, IVAN KURTEV und WILLIAM PIERS: *Bridging the MS/DSL Tools and the Eclipse Modeling Fra-*

- mework*. In: *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, 2005.
- [9] BÖNNEN, CARSTEN und MARIO HERGER: *SAP NetWeaver Visual Composer*. SAP PRESS, 2007.
- [10] DEURSEN, ARIE VAN und PAUL KLINT: *Little languages: little maintenance?* Journal of Software Maintenance, 10(2):75–92, 1998.
- [11] DSTG: *Pattern By Example*. Delta Software Technology GmbH, Oktober 2003.
- [12] DSTG: *ANGIE Add-Ins – Benutzerhandbuch*. Delta Software Technology GmbH, Juni 2007.
- [13] DSTG: *SCORE Adaptive Bridges – Referenzhandbuch*. Delta Software Technology GmbH, März 2007.
- [14] ESSL, MANUEL und UWE OEHLER: *Praxisbuch SAP xApp Analytics*. SAP PRESS, 2006.
- [15] FIRUS, VIKTORIA, HEIKO KOZIOLEK, STEFFEN BECKER, RALF H. REUSSNER und WILHELM HASSELBRING: *Empirische Bewertung von Performanz-Vorhersageverfahren für Software-Architekturen*. In: LIGGESMEYER, PETER, KLAUS POHL und MICHAEL GOEDICKE (Herausgeber): *Software Engineering 2005 Proceedings - Fachtagung des GI-Fachbereichs Softwaretechnik*, Band 64 der Reihe *GI-Edition of Lecture Notes in Informatics*, Seiten 55–66, 2005.
- [16] FOWLER, MARTIN: *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
- [17] FOWLER, MARTIN: *Language Workbenches: The Killer-App for Domain Specific Languages?* <http://www.martinfowler.com/articles/languageWorkbench.html>, 2005. Letzter Zugriff: 15.12.2008.
- [18] FOX, MARIA und DEREK LONG: *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. J. Artif. Intell. Res. (JAIR), 20:61–124, 2003.
- [19] FREIMUT, BERND, TEADE PUNTER, STEFAN BIFFL und MARCUS CIOLKOWSKI: *State-of-the-Art in Empirical Studies*. Technischer Bericht ViSEK/007/E, ViSEK, University of Kaiserslautern, 2002.
- [20] GAMMA, ERICH, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [21] GANTER, BERNHARD und RUDOLF WILLE: *Formale Begriffsanalyse*. Springer, 1996.

- [22] GIESE, CORD: *Frame-basierte Generatoren in der Praxis*. ObjektSpektrum, 1:64–70, 2004.
- [23] GIESE, CORD und RÜDIGER SCHILLING: *Modellgetriebene Generatorentwicklung*. ObjektSpektrum, 3:77–83, 2005.
- [24] GIESE, CORD, ARND SCHNIEDERS und JENS WEILAND: *A Practical Approach for Process Family Engineering of Embedded Control Software*. In: *Proceedings of the 14th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*. IEEE Computer Society Press, March 2007.
- [25] GOLDSCHMIDT, T., J. WINZEN und R. REUSSNER: *Evaluation of Maintainability of Model-driven Persistency Techniques*. In: *IEEE CSMR 07 - Workshop on Model-Driven Software Evolution (MoDSE2007)*, Seiten 17–24, 2007.
- [26] GOLDSCHMIDT, THOMAS, RALF REUSSNER und JOCHEN WINZEN: *A Case Study Evaluation of Maintainability and Performance of Persistency Techniques*. In: *ICSE '08: Proceedings of the 30th international conference on Software engineering*, Seiten 401–410, New York, NY, USA, 2008. ACM.
- [27] HASSELBRING, WILHELM, RALF REUSSNER, HOLGER JAEKEL, JURGEN SCHLEGELMILCH, THORSTEN TESCHKE und STEFAN KRIEGHOFF: *The Dublo Architecture Pattern for Smooth Migration of Business Information Systems: An Experience Report*. In: *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, Seiten 117–126, Washington, DC, USA, 2004. IEEE Computer Society.
- [28] HEUVEL, WILLEM-JAN VAN DEN: *Matching and Adaptation: Core Techniques for MDA-(ADM)-driven Integration of new Business Applications with Wrapped Legacy Systems*. In: *Proceedings of MELS Workshop (EDOC)*. IEEE Press, October 2004.
- [29] HEUVEL, WILLEM-JAN VAN DEN: *Aligning Modern Business Processes and Legacy Systems - A Component-Based Perspective*. Cooperative Information Systems. MIT Press, 2007.
- [30] HEUVEL, WILLEM-JAN VAN DEN, WILHELM HASSELBRING und MIKE P. PAPA-ZOGLU: *Top-Down Enterprise Application Integration with Reference Models*. In: *EFIS*, Seiten 11–22, 2000.
- [31] HEUVEL, WILLEM-JAN VAN DEN, JOS VAN HILLEGERSBERG und MIKE PAPA-ZOGLU: *A Methodology to Support Web-Services Development Using Legacy Systems*. In: *Engineering Information Systems in the Internet Context*, Seiten 81–103, 2002.
- [32] HUDAK, PAUL: *Modular Domain Specific Languages and Tools*. In: *Proceedings of the 5th International Conference on Software Reuse (ICSR '98)*, Seiten 134–

- 142, Washington, DC, USA, June 1998. IEEE Computer Society Washington, DC, USA.
- [33] KEMPA, MARTIN und ZOLTÁN ÁDÁM MANN: *Model Driven Architecture*. Informatik Spektrum, 28(4):298–302, 2005.
- [34] KRUSE, STEFFEN, MALTE ZILINSKI, HILKE GARBE und CLAUS MÖBUS: *MDA und KI: Domänenspezifische Modellierung und Umsetzung wissensintensiver Prozesse*. In: *Lecture Notes in Informatics: Software Engineering 2008*, Seiten 184–190, February 2008.
- [35] KÜBLER, JENS: *Wartbarkeit im Kontext modellgetriebener Softwareentwicklung*. Diplomarbeit, Universität Karlsruhe (TH), 2008.
- [36] LINK, JOHANNES: *Softwaretests mit JUnit*. Dpunkt Verlag, Second Auflage, 2005.
- [37] MCDERMOTT, D.: *PDDL — the planning domain definition language*, 1998.
- [38] MERTENS, PETER: *Individual- und Standardsoftware: tertium datur?* In: MAYR, H.C. (Herausgeber): *Beherrschung von Informationssystemen*, Seiten 55–81, Wien u. a., 1996.
- [39] MICROSOFT: *Overview of Domain-Specific Language Tools*. [http://msdn2.microsoft.com/en-us/library/bb126327\(v5.80\).aspx](http://msdn2.microsoft.com/en-us/library/bb126327(v5.80).aspx), 2007. Letzter Zugriff: 15.12.2008.
- [40] MILLER, JOAQUIN und JISHNU MUKERJI: *MDA Guide Version 1.0.1*. Object Management Group (OMG), June 2003.
- [41] MILNER, R.: *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, UK, 1999.
- [42] MÖBUS, CLAUS: *Towards an Epistemology on Intelligent Problem Solving Environments: The Hypothesis Testing Approach*. In: GREER, J. (Herausgeber): *Proceedings of AI-ED 95, World Conference on Artificial Intelligence and Education*, Seiten 138–145, Washington, DC, 1995. Association for the Advancement of Computing in Education (AACE).
- [43] MÖBUS, CLAUS, HEIKO SEEBOLD und HILKE GARBE: *A Greedy Knowledge Acquisition Method for the Rapid Prototyping of Knowledge Structures*. In: *K-CAP '05: Proceedings of the 3rd international conference on Knowledge capture*, Seiten 211–212. ACM, 2005.
- [44] OBJECT MANAGEMENT GROUP (OMG): *Business Process Modeling Notation Specification*. OMG, February 2006. Final Adopted Specification.
- [45] OBJECT MANAGEMENT GROUP (OMG): *Meta Object Facility (MOF) Core Specification Version 2.0*. Object Management Group (OMG), January 2006.



- [46] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS): *Web Services Business Process Execution Language Version 2.0*, April 2007.
- [47] PELTZ, CHRIS: *Web Services Orchestration and Choreography*. IEEE Computer, 36(10):46–52, 2003.
- [48] POERNOMO, IMAN H., RALF H. REUSSNER und HEINZ W. SCHMIDT: *Architectures of Enterprise Systems: Modelling Transactional Contexts*. In: *Proceedings of the First IFIP/ACM Working Conference on Component Deployment (CD 2002)*, Band 2370 der Reihe *Lecture Notes in Computer Science*, Seiten 233–243. Springer-Verlag, Berlin, Germany, June 2002.
- [49] POERNOMO, IMAN H., RALF H. REUSSNER und HEINZ W. SCHMIDT: *Architectural Configuration with EDOC and .NET Component Services*. In: CHROUST, GERHARD (Herausgeber): *Euromicro 2003, IEEE, Antalya - Turkey, September 3rd-5th, 2003*, 2003.
- [50] RAJPATHAK, DNYANESH: *A Generic Library of Problem-Solving Methods for Scheduling Application*. Doktorarbeit, The Open University, 2004.
- [51] RAMOS, C.S. und N. OLIVEIRA, K.M.AND ANQUETIL: *Legacy software evaluation model for outsourced maintainer*. In: *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*, Seiten 48–57, Mar 2004.
- [52] REUSSNER, RALF: *Automatic component protocol adaptation with the CoConut/J tool suite*. Future Generation Comp. Syst., 19(5):627–639, 2003.
- [53] REUSSNER, RALF, JENS HAPPE und ANNEGRETH HABEL: *Modelling Parametric Component Contracts and the State Space of Composite Components by Graph Grammars*. In: *Fundamental Approaches to Software Engineering (FASE)*, 2005.
- [54] REUSSNER, RALF und WILHELM HASSELBRING (Herausgeber): *Handbuch der Software-Architektur*. dpunkt.verlag, 2. Auflage, 2008.
- [55] REUSSNER, RALF, HEINZ W. SCHMIDT und IMAN PEOERNOMO: *Reliability prediction for component-based software architectures*. Journal of Systems and Software, 66(3):241–252, 2003.
- [56] REUSSNER, RALF H., IMAN H. POERNOMO und HEINZ W. SCHMIDT: *Reasoning on Software Architectures with Contractually Specified Components*. In: CECHICH, A., M. PIATTINI und A. VALLECILLO (Herausgeber): *Component-Based Software Quality: Methods and Techniques*, Nummer 2693 in *LNCS*, Seiten 287–325. Springer, 2003.
- [57] RUSSELL, NICK, WIL M. P. VAN DER AALST, ARTHUR H. M. TER HOFSTEDDE und PETIA WOHEDE: *On the suitability of UML 2.0 activity diagrams for business*

- process modelling*. In: STUMPTNER, MARKUS, SVEN HARTMANN und YASUSHI KIYOKI (Herausgeber): *Conceptual Modelling 2006, Third Asia-Pacific Conference on Conceptual Modelling (APCCM 2005), Hobart, Tasmania, Australia, January 16-19 2006*, Band 53 der Reihe *CRPIT*, Seiten 95–104. Australian Computer Society, 2006.
- [58] SAP AG: *Erstellen von Composite Applications*. [http://help.sap.com/saphelp\\_nw70ehp1/helpdata/de/47/fd6c25d7914aa6e10000000a421937/content.htm](http://help.sap.com/saphelp_nw70ehp1/helpdata/de/47/fd6c25d7914aa6e10000000a421937/content.htm), 2008. Letzter Zugriff: 15.12.2008.
- [59] SCHEER, AUGUST-WILHELM und MARKUS NÜTTGENS: *ARIS Architecture and Reference Models for Business Process Management*. In: AALST, WIL M. P. VAN DER, JÖRG DESEL und ANDREAS OBERWEIS (Herausgeber): *Business Process Management, Models, Techniques, and Empirical Studies*, Band 1806 der Reihe *Lecture Notes in Computer Science*, Seiten 376–389. Springer, 2000.
- [60] SCHEER, JÖRN W. und ANA CATINA (Herausgeber): *Einführung in die Repertory Grid-Technik. Band 1: Grundlagen und Methoden*. Hans Huber, 1. Auflage, 1993.
- [61] SCHMIDT, HEINZ W., BERND J. KRÄMER, IMAN H. POERNOMO und RALF H. REUSSNER: *Predictable Component Architectures Using Dependent Finite State Machines*. In: WIRSING, MARTIN, ALEXANDER KNAPP und SIMONETTA BALSAMO (Herausgeber): *Radical Innovations of Software and Systems Engineering in the Future, 9th International Workshop, RISSEF 2002*, Seiten 310–324. Springer, 2002.
- [62] SCHMIDT, HEINZ W. und RALF H. REUSSNER: *Generating Adapters for Concurrent Component Protocol Synchronisation*. In: *Proceedings of the Fifth IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*, 2002.
- [63] SCHMIDT, HEINZ W. und RALF H. REUSSNER: *Parameterised Contracts and Adaptor Synthesis*. In: *Proceedings of the ICSE Workshop of Component Oriented Software Engineering (CBSE5)*. IEEE, 2002.
- [64] SCHREIBER, G., H. AKKERMANS, A. ANJEWIERDEN, R. DE HOOG, N. SHADBOLT, W. VAN DE VELDE und B. WIELINGA: *Knowledge Engineering and Management*. The MIT Press, 2002.
- [65] SIMOS, MARK: *Organization domain modelling (ODM) guidebook version 2.0*. Technischer Bericht STARS-VC-A025/001/00, Synquiry Technologies, Inc., 1996.
- [66] STAHL, THOMAS, MARKUS VÖLTER, SVEN EFFTINGE und ARNO HAASE: *Modellgetriebene Softwareentwicklung*. dpunkt.verlag, Heidelberg, 2. Auflage, 2007.
- [67] STARKE, RICO: *Vergleich der Performance von Datenbank-Anbindungen unter.NET*. Diplomarbeit, Universität Oldenburg, 2004.

- [68] TESCHKE, THORSTEN, HOLGER JAEKEL, STEFAN KRIEGHOFF, MARC LANGNICKEL, WILHELM HASSELBRING und RALF REUSSNER: *Funktionsgetriebene Integration von Legacy-Systemen mit Web Services*. In: WILHELM HASSELBRING, MANFRED REICHERT (Herausgeber): *Tagungsband zum Workshop "EAI 2004 - Enterprise Architecture Integration"*, Seiten 19–28. GITO Verlag, 2004.

ISBN: 978-3-86644-416-4

[www.uvka.de](http://www.uvka.de)

ISBN 978-3-86644-416-4



9 783866 444164 >