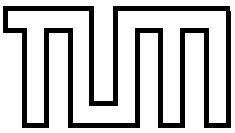
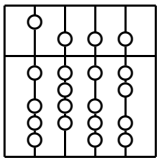


Technische Universität München
Fakultät für Informatik



Diplomarbeit

Modell-basierte Transformationen von Informationsmodellen zum Management von Anwendungslandschaften

Sabine Buckl

Aufgabensteller:	Prof. Dr. Florian Matthes
Betreuer:	André Wittenburg
Abgabedatum:	15.12.2005

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Dezember 2005

Sabine Buckl

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die mir beim Erstellen dieser Arbeit durch ihre fachliche und persönliche Unterstützung zur Seite gestanden sind.

Allen voran danke ich Herrn Prof. Dr. Florian Matthes für die Ermöglichung und Förderung meiner Diplomarbeit.

André Wittenburg danke ich für sein hervorragende und immer präsente Betreuung. Sein großes Interesse und seine hilfreiche Unterstützung waren für die Verwirklichung dieser Arbeit sehr wichtig.

Allen Mitgliedern des Lehrstuhls danke ich für die freundliche und angenehme Atmosphäre. In zahlreichen Gesprächen und Diskussionen konnten viele Probleme gelöst und Fragen geklärt werden.

Mein ganz besonderer Dank gilt jedoch meinen Eltern und meinem Partner, die mich all die Jahre hinweg tatkräftig unterstützt haben und meinen Plänen und Ideen gegenüber immer offen waren.

Zusammenfassung

Die hohe Lebensdauer und steigende Anzahl von IT-Systemen in Unternehmen führt zu immer komplexeren Anwendungslandschaften. Gleichzeitig entwickelt sich die Informationstechnologie von einem Werkzeug zur Umsetzung der Unternehmensstrategien hin zu einem integralen Bestandteil dieser Strategie selbst. Aufgabe des Enterprise Architecture Managements ist es diesen Wandel zu unterstützen, die gegenseitige Ausrichtung von Business und IT zu verbessern und ebenso die Steuerung von komplexen Anwendungslandschaften zu optimieren.

Der Prozess zum Enterprise Architecture Management basiert auf einen Informationsmodell, welches die Informationsobjekte der Enterprise Architecture geeignet in Beziehung setzt. Da weder in Forschung noch in der Wirtschaft derzeit ein standardisiertes Informationsmodell existiert, wird in dieser Arbeit untersucht, welche Konzepte zur Modellierung eines Informationsmodells notwenig sind. Ein gemeinsames Metamodell ermöglicht einen Rückschluss auf die Anforderungen hinsichtlich eines Repositories, welches die Informationsobjekte konform zum Informationsmodell speichern kann.

Des Weiteren liegen die Informationen für das Enterprise Architecture Management bereits in verschiedenen Datenbasen in einem Unternehmen vor, so dass verschiedene Informationsmodelle zusammengeführt werden müssen. Hierzu untersucht die Arbeit die Eignung von Modelltransformationen unter Berücksichtigung der Anforderungen an Metamodellierungkonzepte, die sich aus der explorativen Analyse existierender Modelle ergibt. Die Modelltransformation von Informationsmodellen ermöglicht es zusätzlich auf Erkenntnisse des Forschungsprojektes Softwarekartographie zurückzugreifen, so dass existierende Visualisierungssmethodiken für die Enterprise Architecture ohne eine Redefinition der Visualisierung wieder verwendet werden können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

»Nichts ist beständiger als der Wandel«
Heraklit von Ephesos (540-480 v. Chr.)

Die Gesellschaft befindet sich gegenwärtig im Wandel von einer Industriegesellschaft in eine Informationsgesellschaft, in der die Informations- und Telekommunikationstechnologien, als entscheidende Erfolgs- und Wettbewerbsfaktoren für Unternehmen im Mittelpunkt stehen. Die Informationstechnologie wird zunehmend nicht nur als Werkzeug, das die Umsetzung der Unternehmensziele unterstützt angesehen, sondern ist ein integraler Bestandteil der Unternehmensstrategie geworden [PRW03].

Sowohl der Zugriff auf Informationen als auch die generelle Informiertheit sind nach einer Studie des Fraunhofer Instituts[1] entscheidende Faktoren für die Produktions- und Wettbewerbsfähigkeit eines Unternehmens. Durch die verkürzten Innovationszyklen von Produkten erweist sich die Anpassungsfähigkeit der strategischen Planung, organisatorischer Abläufe und Strukturen, um auf veränderte Markt- und Umweltbedingungen reagieren zu können, zunehmend als existenzielle Eigenschaft von Unternehmen [Win04].

Die Forderung nach einer flexiblen Unternehmensarchitektur (engl. Enterprise Architecure), um auf strategische Neuausrichtungen des Unternehmens reagieren zu können einerseits und der Anforderung neue IT-Lösungen zeitoptimiert umzusetzten andererseits, führt zu punktuellen Optimierungen bezogen auf die Gesamt-IT-Architektur. Eine heterogene, komplexe Anwendungslandschaft mit einer Vielzahl von Insellösungen und extrem hohem Wartungsaufwand ist die Folge. Das Management dieser komplexen Anwendungslandschaften stellt in vielen Unternehmen gegenwärtig eine wesentliche Aufgabe dar [Der03].

Die Einführung einer Enterprise Architecture ermöglicht eine gesamtheitliche Sicht auf die Anwendungslandschaft des Unternehmens und trägt zu einer geeigneten Steuerung der Evolution der Anwendungslandschaft bei. Basis der Enterprise Architecture bildet die Konzeption eines geeigneten Informationsmodells. Weder in der Forschung noch in der Wirtschaft (sowohl bei Anwendern als auch bei Werkzeugherstellern) existiert ein gemeinsames Informationsmodell und ein gemeinsames Metamodell

[1] Vgl. die Nutzerstudie *Office Performance* [Fra05].

für ein Informationsmodell zur Beschreibung von Anwendungslandschaften, welches sowohl Best Practices bündelt, als auch eine gemeinsame Modellierungsmethodik verfolgt [seb05a].

1.1 Aufgabenstellung und Zielsetzung

Die Arbeit entsteht im Rahmen des Forschungsprojektes *Softwarekartographie* des Lehrstuhls für *Software Engineering betrieblicher Informationssysteme* der TU München. Das Ziel des Forschungsprojektes ist die Entwicklung von Methoden und Modellen zur Beschreibung, Bewertung und Gestaltung von Anwendungslandschaften. Diese Arbeit fokussiert auf der Modellierung und Dokumentation von Informationsmodellen für die Softwarekartographie und der Enterprise Architecture im Allgemeinen.

Auf Basis der Erkenntnisse im Forschungsprojekt Softwarekartographie, sollen existierende Informationsmodelle auf gemeinsame Ansätze bei der Modellierung untersucht werden. Neben der Analyse relevanter Informationsmodelle ausgewählter Unternehmen und Werkzeughersteller, werden zusätzlich standardisierte Modelle, wie das Common Information Model (CIM) der DMTF [DMT05] und das IT Portfolio Management Facility (ITPMF) der OMG [OMG04a] betrachtet. Aufbauend auf der Analyse sollen Gemeinsamkeiten der Metamodelle der untersuchten Informationsmodelle identifiziert und konsolidiert werden, um eine Obermenge der verwendeten Konzepte (Klassen, Assoziationen, etc.) zu extrahieren.

Ausgehend von der Analyse existierender Informationsmodelle und der Konsolidierung der verwendeten Konzepte, soll ein Abgleich mit der Meta Object Facility (MOF) stattfinden, der den MOF-Ansatz zur Modellierung von Informationsmodellen verifizieren soll. Zusätzlich werden Transformationssprachen zur Modell-basierten Transformation zwischen verschiedenen Informationsmodellen auf ihre Unterstützung der Modellierungskonzepte analysiert.

Ziel bei der Transformation von Informationsmodellen ist zum einen mehrere Informationsmodelle in ein gemeinsames Modell zu überführen, um die Informationen aus verschiedenen Quellen für die Enterprise Architecture zu nutzen. Zum anderen können Mechanismen, z.B. zur Visualisierung der Enterprise Architecture, die auf einem gegebenen Informationsmodell beruhen, wieder verwendet werden, wenn das Quellmodell in das vorgegeben Zielmodell transformiert wird.

Die Verifikation des MOF-Ansatzes zur Modellierung von Informationsmodellen und die Analyse existierender Transformationssprachen zur Modell-basierten Transformation stellen die Zielsetzung dieser Arbeit dar. Eine exemplarische Transformation zwischen zwei verschiedenen Informationsmodellen soll die entwickelten Konzepte auf ihre Eignung hin untersuchen.

1.2 Aufbau der Arbeit

Die Diplomarbeit ist in sieben Hauptkapitel gegliedert. Die Kapitel 2 bis 3 haben einen einführenden Charakter, während die Kapitel 4 bis 6 den zentralen Teil der Arbeit ausmachen. Kapitel 7 schließt die Arbeit durch eine Zusammenfassung und einen Ausblick ab.

Im folgenden zweiten Kapitel "Thematisches Umfeld" erfolgt eine Einführung in den thematischen Kontext dieser Arbeit. Die Begriffe Enterprise Architecture und Softwarekartographie werden eingeführt und diskutiert, um die Motivation und Aufgabenstellung der Arbeit näher zu erläutern.

Eine Einführung, in die für das Verständnis der weiteren Arbeit notwendigen Teilbereiche rund um das Themengebiet der Modellierung bietet, das dritte Kapitel "Modelle und Modellierungssprachen". Begriffe und Konzepte der Softwarekartographie und der Model Driven Architecture werden ebenso vorgestellt, wie standardisierte Modellierungssprachen.

Die "Analyse von Informationsmodellen" in Kapitel vier stellt den zentralen Teil der Arbeit dar. Informationsmodelle von Unternehmen und Werkzeugherstellern, sowie standardisierte Ansätze werden auf Gemeinsamkeiten bei der Modellierung analysiert und mit den Konzepten standardisierter Modellierungssprachen abgeglichen.

Aufbauend auf den Analyseergebnissen der Informationsmodelle werden in Kapitel fünf "Analyse von Transformationssprachen" existierende Transformationssprachen vorgestellt und auf ihre Eignung zur Transformation zwischen Informationsmodellen untersucht.

Eine exemplarische Transformation von Informationsmodellen erfolgt in Kapitel sechs "Transformation von Informationsmodellen", um die in den vorangegangenen Kapiteln entwickelten Konzepte auf ihre Eignung hin zu untersuchen.

Abschließend werden im siebten Kapitel "Zusammenfassung und Ausblick" die Ergebnisse der Arbeit zusammengefasst, sowie ein Ausblick auf weiterführende Themen aufgezeigt.

2 Thematisches Umfeld

Dieses Kapitel soll die Arbeit in ihr thematisches Umfeld einbetten und einen Überblick über das Themengebiet schaffen. Außerdem sollen die theoretischen Grundlagen bereitgestellt werden, auf denen die weitere Arbeit basiert und wichtige Begriffe näher erläutert werden.

2.1 Enterprise Architecture

Ein wichtiger immer wiederkehrender Begriff in dieser Arbeit, der in der Informatik und Wirtschaftsinformatik eine zunehmend größere Rolle spielt[1], ist der Begriff der *Enterprise Architecture* (Unternehmensarchitektur). Da dieser weder in der Praxis noch in der Wissenschaft einheitlich verwendet wird, soll zunächst die Bedeutung des Begriffs Architektur im Kontext der Informatik und anschließend der Begriff der Enterprise Architecture näher bestimmt werden.

2.1.1 Definition

Nach der Definition des IEEE Standards 1471-2000 bedeutet Architektur: "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [IEE00, Seite 9]. Demnach besteht eine Architektur aus zwei wichtigen Bestandteilen, zum einem dem Aufbau des Systems, das die einzelnen Komponenten und ihre Beziehungen zueinander und zur Umgebung beschreibt, und zum anderen aus den Leitsätzen, auf denen das Design und die Entwicklung basieren.

Diese beiden Bestandteile finden sich auch in der Definition von Sinz wieder, der die Architektur im Kontext der Informatik analog zum Architekturverständnis im Bauwesen definiert: Eine Architektur besteht aus einem Bauplan im Sinne einer Spezifikation und Dokumentation, der alle Komponenten und ihre Beziehungen unter allen relevanten Blickwinkeln enthält, sowie den Konstruktionsregeln, für die Erstellung des Bauplans [Sin02].

[1] Die Anzahl der Publikationen, die sich mit Enterprise Architecture beschäftigen, ist in den letzten Jahren deutlich gestiegen [LW04].

Aus dieser Analogie zum Bauwesen ergibt sich auch der Unterschied der beiden Definitionen. Während im Bauwesen Architektur im statischen Sinne verstanden wird, geht die IEEE explizit auf das dynamische Verhalten der Architektur während ihrer Entwicklung (*Evolution*) ein.

Zachman, der sich in den 80er Jahren als einer der ersten mit Architekturen im Kontext der Informatik beschäftigte definiert Architektur als "that set of design artefacts, or descriptive representations, that are relevant for describing an object such that it can be produced to requirements (quality) as well as maintained over the period of its useful life (change)" [Zac97, Seite 5]. Zachman legt in seiner Definition einen Schwerpunkt auf den zeitlichen Aspekt und geht besonders auf die Veränderungen ein, der eine Architektur unterworfen ist und stellt entsprechende Anforderungen an die formalen Beschreibungsmöglichkeiten.

Aus den oben genannten Definitionen lässt sich ein kollektives Verständis von Architektur finden: Die Architektur eines Systems besteht aus einem oder mehreren Plänen, die alle Komponenten eines Systems, ihre Beziehungen sowie die Strategien, auf denen die Planung, das Design und die Entwicklung der Komponenten im Laufe der Zeit basiert, beschreiben.

Die Definition von Architektur eines Systems lässt sich auf die Architektur eines Unternehmens übertragen. Ein Unternehmen ist ein System und jedes System besitzt notwendigerweise eine Architektur [Sin04].

Im Kontext der Enterprise Architecture besagt Zachmans Definition, dass jede Enterprise Architecture aus Modellen (descriptive representations) besteht, die relevante Objekte des Unternehmens beschreiben, so dass Unternehmensstrategien und -ziele (requirements) erfüllt und während des Lebenszyklus der Objekte überwacht (change) werden können [Krc05].

Eine Enterprise Architecture besteht in der Regel aus mehreren zusammenhängenden Modellen, die die Struktur, Funktion, Ziele und Strategien eines Unternehmens beschreiben. Sie ist somit die Basis für die strategische Entwicklung eines Unternehmens und hat Auswirkungen auf die Planung, Durchführung und Kontrolle von Unternehmensaktivitäten [Buh04].

Um die Entwicklung eines Unternehmens verfolgen zu können, enthält die Enterprise Architecture nicht nur ein Modell der aktuellen Organisation (Ist-Zustand), sondern auch Modelle der visionären gestalteten Organisation in der Zukunft (Soll-Zustand) und Planmodelle, die die einzelnen Zwischenschritte auf dem Weg dorthin beschreiben [McG03].

Mithilfe der Modelle soll die Enterprise Architecture einen ganzheitlichen Überblick liefern. Die Enterprise Architecture fokussiert auf Aspekte von unternehmensweitem Interesse und bildet somit eine Makro-Architektur. Die Mikro-Architektur hingegen bezieht sich auf die Architektur einer konkreten Anwendung und muss sich in die Makro-Architektur eingliedern lassen und mit ihr kompatibel sein [Gae04].

2.1.2 Nutzen einer Enterprise Architecture

Aus der Notwendigkeit selbst in großen Unternehmen Wissen über die informationswirtschaftliche Nutzung und die technologischen Handlungsmöglichkeiten des eigenen Unternehems haben zu müssen, ergibt sich die Forderung nach einer systematischen Entwicklung und Modellierung der Enterprise Architecture [Krc05].

Die historisch gewachsenen, heterogenen Prozess- und IT-Landschaften führen zu einer hohen Komplexität und einer geringen Flexibilität, die zudem eine hohe Kostenbelastung z.B. durch mehrfach Lizenzen, Pflege und Wartung bedeuten. Eine Enterprise Architecture soll helfen die unkoordiniert entstandene IT-Landschaft z.B. durch Standardisierung von Prozessen oder Software zu reorganisieren [Buh04, Uhl04].

Traditionell wird die IT eher als Mittel zur Reduzierung des administrativen Aufwandes gesehen und nicht als strategisches Mittel um Unternehmensziele zu erreichen. Ein Leistungs- oder Wettbewerbsvorteil wird nicht durch die IT alleine erlangt. Die IT kann Unternehmen nur helfen ihr Leistungsziel zu errreichen, wenn sie effektiv an die Unternehmensstrategien, -prozesse und -anwendungen angepasst ist. Das optimale Alignment[2] von Business und IT, wie in Abbildung 2.1 visualisiert wird, ist die primäre Aufgabe einer Enterprise Architecture. Das Business bestimmt hierbei die Richtung und Priorisierung von IT-Investitionen, während die IT zur Realisierung der Geschäftsstrategien befähigt [MS03].

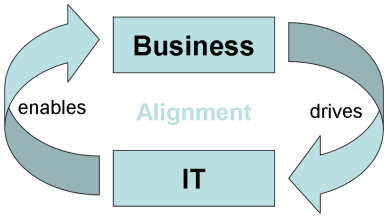


Abbildung 2.1: Alignment zwischen Business und IT

Die Einführung einer Enterprise Architecture hat neben der Optimierung des Alignments von Business und IT noch weiteren Nutzen:

- Simplifikation der Analyse und Gestaltung komplexer Systeme durch ganzheitlichen Überblick und Konzentration von Wissen [BNS03]
- Vermeidung von Fehlentscheidungen durch fundierte Entscheidungsfindung und vereinfachte Konsensbildung [Kle04]
- Senkung der Kosten durch Konsolidierung der Informationstechnologie und Geschäftsprozessoptimierung [Kle04]

[2] deutsch: Abgleich, Anpassung, Ausrichtung

- Förderung der Integration durch standardisierte Geschäftsprozesse, Schnittstellen und unternehmensweite Interoperabilität [Fed01]
- Steigerung der Flexibilität durch Standardisierung, verbesserte Dokumentation und erhöhte Transparenz [Zac99]
- Verbesserung der Wettbewerbsfähigkeit durch beschleunigte Entwicklungsprozesse und erhöhte Flexibilität [Fed01]
- Steigerung des Unternehmenswertes durch effizientes Management von Veränderungen und strategische Differenzierung [Fed01]

Demgegenüber stehen die Kosten für die Erstellung und konsequente Pflege der Enterprise Architecture:

Bei der Erstellung werden Informationen benötigt, die auf viele Personen verteilt meist nur in Form von Kopfwissen vorhanden sind. Diese Informationen sind in der Regel unvollständig oder sogar widersprüchlich und müssen unter hohem Zeit- und Kostenaufwand zusammengetragen und konsolidiert werden [Buh04].

Nach der Erstellung der Enterprise Architecture muss diese konsequent gepflegt und auf dem neusten Stand gehalten werden, da Modelle sehr schnell veralten und sonst zu Fehlannahmen oder -entscheidungen führen können. Dies verlangt von den verantwortlichen Mitarbeitern eines Unternehmens einen hohen Grad an Verständnis für die Notwendigkeit Zeit mit der Dokumentation von Modellen zu verbringen [McG03].

Aus dieser Notwendigkeit entsteht eine neue Rolle im Unternehmen: die Rolle des *Enterprise Architects*. Seine primäre Aufgabe nach der Einführung einer Enterprise Architecture ist die Etablierung eines Pflegeprozesses. Alle Personen, die die Enterprise Architecture verändern, müssen in diesen Pflegeprozess eingebunden werden. Dementsprechend verlang die Rolle eines *Enterprise Architect* einen hohen Grad an technischem Verständnis und sozialer Kompetenz [New02].

2.1.3 Enterprise Architecture Frameworks

Um Unternehmen die Einführung einer Enterprise Architecture zu erleichtern wurden verschiedene Frameworks zur Erstellung einer Enterprise Architecture entwickelt. In den folgenden Abschnitten soll der Aufbau unterschiedlicher Ansätze vorgestellt werden.

Das Zachman Framework

Das ”Framework for Informationsystem Architecture” wurde 1987 von John Zachman, dem Gründer des *Zachman Institute for Framework Advancement* (ZIFA), in Anlehnung an die Prinzipien der klassischen Architekturlehre entwickelt. Dieser Einfluss findet

sich in dem einheitlichen Wortschatz und einem gegebenen Satz an Sichten und Regeln wieder. Das Framework beschränkt sich dabei auf eine reine Architektur ohne eine strategische oder planende Methodik [Zac87].

Das Zachman Framework eignet sich neben der Beschreibung für Informationssystemarchitekturen auch zur Beschreibung von Enterprise Architectures. Die Einführung des Zachman Frameworks als universelle Sprache zur Unterstützung der Kommunikation, Untersuchung und Implementierung von Enterprise Architecture Konzepten bildet das Hauptziel der ZIFA [Zac05].

Nach der Definition von Zachman existiert nicht eine Systemarchitektur sondern ein Satz von sich ergänzenden Architekturen, die die Architektur des Systems aus unterschiedlichen Perspektiven beleuchten. Dieses Verständnis findet sich in der zweidimensionalen Struktur des von ihm vorgeschlagenen Frameworks wieder. In dem Framework werden unterschiedliche Perspektiven und Fragestellungen als Ausgangspunkte für die Betrachtung der Enterprise Architecture festlegt.

Es werden dabei sechs verschiedene Blickwinkel auf das umzusetzende System unterschieden [Zac87, SZ92]:

- **Scope:** Blickwinkel des strategischen Managements mit Schwerpunkt auf betriebwirtschaftlichen Aspekten (Zweck, Größe, Kosten, etc. des Systems)
- **Business Model:** Blickwinkel des operativen Managements mit Schwerpunkt auf den Geschätsprozessen, -einheiten und ihrer Interaktion
- **System Model:** Blickwinkel des Designers mit Schwerpunkt auf die graphischen Modelle, die für die Implementierung ausgerichtet sind.
- **Technological Model:** Blickwinkel des Entwicklers mit Schwerpunkt der Abwägung von Werkzeugen, Technologie und Material
- **Components Model:** Blickwinkel des Programmierers mit Schwerpunkt auf der detaillierten Darstellung der Lösung (Programmierspache, Datenbankspezifikation, etc.)
- **Descriptive Details:** Blickwinkel des RZ-Mitarbeiters[3] mit Schwerpunkt auf der Darstellung des betriebsfähigen Systems im Unternehmen

Die sechs Perspektiven bilden einen fallenden Abstraktionsgrad von einer stark betriebswirtschaftlich orientierten Sicht bis hin zu einer rein technischen Sicht.

[3] Rechenzentrum-Mitarbeiter (engl: Operator)

Die zweite Dimension im Zachman Framework bilden die sechs[4] für ein Produkt charakteristischen Fragestellungen:

- **What/Data:** Beschreibt die in der jeweiligen Perspektive betrachteten Entitäten und deren Beziehungen
- **How/Function:** Beschreibt die Funktionalität und die Geschäftsprozesse die ein der jeweiligen Perspektive Anwendung finden
- **Where/Network:** Beschreibt die Systemverteilung und die Verbindungen des Unternehmens
- **Who/People:** Beschreibt die Interaktion des Systems mit Personen
- **When/Time:** Beschreibt die zeitlichen Beziehungen in der jeweiligen Perspektive
- **Why/Motivaton:** Beschreibt die Motivation des Unternehmens (Unternehmensziele, -strategien)

Durch die Kombination dieser zwei Dimensionen zu einer Matrix entsteht ein Analyseframework für die Untersuchung von Entwicklungsprozessen von System- und Enterprise Architectures, vgl Abbildung 2.2. Durch die Unabhängigkeit der einzelnen Achsenelemente kann jede Zelle der Matrix unabhängig betrachtet und dokumentiert, und so die Komplexität der Gesamtarchitektur verringert werden [Sys05].

Das ARIS Framework

Die *Architektur Integrierter Informationssysteme* (ARIS) [Sch91, Sch98, Sch01] ist als universell einsetzbares Rahmenkonzept zur Beschreibung betrieblicher Informationssysteme entwickelt worden. Ihr Ziel ist eine ganzheitliche - aus allen Sichten und über alle Entwicklungsphasen - Beschreibung der Informationssysteme, die zur Unterstützung der Geschäftsprozesse eingesetzt werden.

Den Ausgangspunkt für den ARIS-Ansatz bildet dabei ein Modell für Unternehmensprozesse. ARIS verfolgt zwei Grundgedanken um die Komplexität des Modells zu verringern:

Sichtenprinzip: Durch die Zerlegung des Modells in unterschiedliche Sichten, wird die Komplexität der einzelnen Sichten verringert. ARIS zerlegt das Modell in fünf[5] unterschiedliche Sichten, um die Komplexität der Modellgestaltung und -nutzung beherrschen zu können. Die Zerlegung wird dabei von dem aus der Software-Entwicklung

[4] In der ursprünglichen Verson von 1987 definierte Zachman nur drei Fragestellungen. 1992 erweiterte er zusammen mit Sowa sein Konzept um die Fragestellungen Who, When und Why. Des Weiteren wurde ein Metamodell für Business, System und Technological Model eingeführt und sieben Regeln zur Formalisierung des Frameworks veröffentlicht. Für weitere Informationen siehe [SZ92].

[5] In der ursprünglichen ARIS Version waren nur 4 Sichten vorgesehen. 2001 führte Scheer noch eine zusätzlich fünfte Sicht, die sogenannte Leistungssicht ein [Sch01].

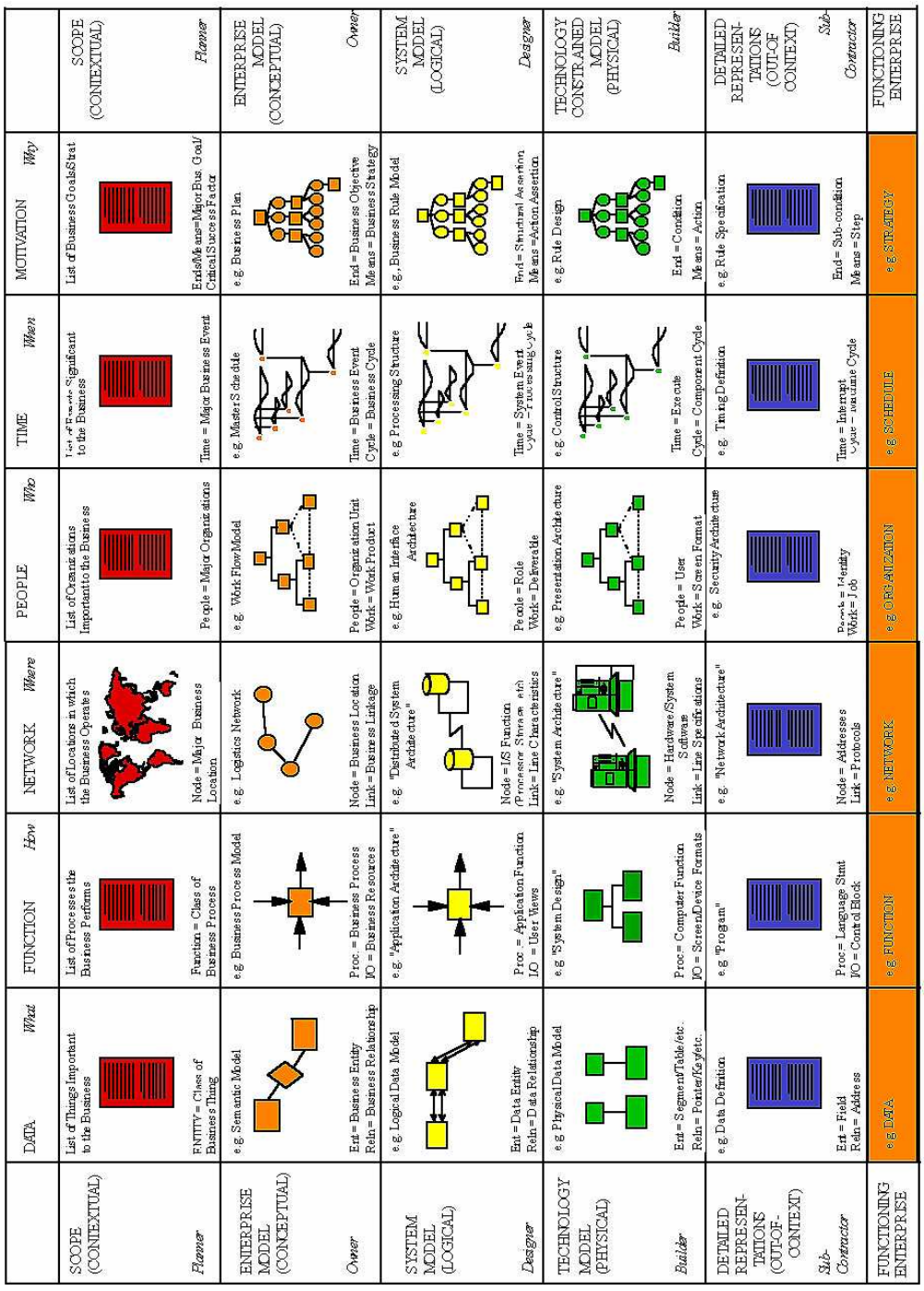


Abbildung 2.2: Das Zachman Framework

stammenden Prinzip der maximalen Kohäsion[6] und minimalen Kopplung[7] getrieben, um möglichst unabhängige Sichten zu erhalten:

- **Datensicht**: Zustände (Umfelddaten) und Ereignisse (die Daten generieren) z.B. Bestellung, Kundendaten, etc.
- **Funktionssicht:** Funktionale Elemente und ihre Zusammenhänge z.B. Auftragsannahme, Produktionsplanung, etc.
- **Organisationssicht:** Struktur von Organisationseinheiten z.B. Sachbearbeiter, Abteilung, etc.
- **Ressourcensicht:** Komponenten der Informationstechnik z.B. Rechner, Datenbanken, etc.
- **Leistungsicht:** Dienst-, Sach- und Finanzleistungen z.B. geprüfter Kundenauftrag, Kundenzahlung, etc.

Zusätzlich wird eine Steuerungssicht eingeführt, um die Beziehungen des Gesamtmodells trotz der getrennten Modellierung der einzelnen Sichten erhalten zu können. Für alle Sichten definiert Scheer Metamodelle, die zu einem integrierten Metamodell verknüpft werden können.

Ebenenprinzip: Neben dem Sichtenprinzip verfolgt der ARIS-Ansatz noch das Konzept unterschiedlicher Beschreibungsebenen. Dabei werden die Informationssysteme in unterschiedlicher Nähe zur Informationstechnik anhand eines Life-Cycle-Konzepts beschrieben. Dieser Life-Cycle umfasst den Lebenslauf eines Informationssystems von der betriebswirtschaftlichen Problemstellung bis zur technischen Implementierung, siehe Abbildung 2.3.

Den Ausgangspunkt der Beschreibungsebenen des ARIS-Ansatzes bildet die *betriebwirtschaftliche Problemstellung*. Sie besteht aus einer fachlichen Zielsetzung in einer halbformalen Beschreibung ohne Details. Die vier Sichten werden in jeweils drei Beschreibungsebenen unterteilt:

- **Fachkonzept:** Formales Anwendungskonzept in formalisierter Sprache mittels datenverarbeitungsfremden Modellen
- **DV-Konzept:** Übertragung des Fachkonzeptes in die DV, Schnittstellenspezifikation
- **Technische Implementierung:** Übertragung des DV-Konzeptes auf konkrete hardware- und softwaretechnische Komponenten

[6] Bindung in Modulen [Bal01].
[7] Kopplung zwischen Modulen [Bal01].

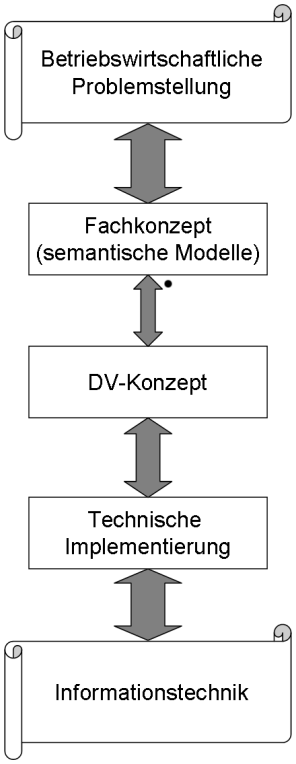


Abbildung 2.3: Beschreibungsebenen eines Informationssystems nach [Sch98]

Mithilfe des Life-Cycle-Modells[8] wird die Ressourcensicht als eigenständiger Beschreibungsgegenstand durch die drei Beschreibungsebenen - besonders dem DV-Konzept und der Technischen Implementierung - in der jeweiligen Sicht ersetzt. Der Ebene des Fachkonzepts kommt eine besondere Bedeutung zu, da sie die größte Lebensdauer innerhalb des Life-Cycle-Modells besitzt.

Durch die Zerlegung in Sichten und Ebenen besteht die Möglichkeit für jede Sicht und jede Ebene gesonderte Methoden zur Beschreibung zu verwenden, die eine Fokussierung auf den Schwerpunkt der jeweiligen Sicht oder Ebene ermöglichen, vgl Tabelle 2.1.

Diese 4-Sichten-Architektur ist auch unter dem Begriff des ARIS-Hauses bekannt, siehe Abbildung 2.4.

[8] Im Aris Kontext endet das Life-Cycle-Modell, im Gegensatz zum allgemeinen Verständnis mit der technischen Implementierung. Die Evolution eines Informationssystems im Betrieb ist im ARIS Life-Cycle-Modell nicht enthalten.

	Fachkonzept	DV-Konzept	Implementierung
Datensicht	erweitertes ER-Modell	Relationen-diagramm	Tabellendiagramm
Funktionssicht	Zieldiagramm, Funktionsbaum, Ablaufdiagramm	Module, Mini-spezifikationen	Programm, Compiler, Interpreter
Organisationssicht	Organigramm	Netztopologie	Netzdiagramm
Leistungssicht	Produktmodell	ARIS	ARIS
Steuerungssicht	Wertschöpfungs-diagramm, EPK, Vorgangskette	Zugriffsdiagramm	Zugriffsdiagramm

Tabelle 2.1: Beschreibungsmethoden und Diagramme des ARIS-Frameworks nach [Plü02]

2.1.4 Vergleich der beiden Frameworks

Das Ziel beider Ansätze ist die Integration von Informationssystemen und die Bestrebung die Lücke zwischen Geschäftsprozessen und Implementierung zu schließen. Beide Frameworks greifen dazu auf das Life-Cycle-Konzept zurück. Bei ARIS wird dieses explizit erwähnt und in die unterschiedlichen Phasen in Anlehnung an die Software-Entwicklung unterteilt. Zachman benutzt anstelle des Life-Cycle-Begriffs den Ausdruck der unterschiedlichen Blickwinkel oder Perspektiven. Dabei können den einzelnen Phasen (ARIS) die entsprechenden Blickwinkel (Zachman) zugeordnet werden, vgl. Tabelle 2.2 [BNS03].

ARIS Phasen	Zachman Blickwinkel
betriebswirtschaftl. Problemstellung	Scope
Fachkonzept	Business Model
DV-Konzept	System Model
Technische Implementierung	Technological Components Model
Informationstechnik	Descriptive Details

Tabelle 2.2: Zuordnung der ARIS-Phasen zu Zachman-Blickwinkeln

Abbildung 2.4: ARIS-Haus: Zerlegungsebenen und -sichten des ARIS Konzeptes [Sch98]

Beide Ansätze benutzen eine zweidimensionale Aufspaltung um die Komplexität des Gesamtmodells zu verringern. Dazu wird einerseits ein Abstraktionskonzept verwendet, um von den betriebswirtschaftlichen Aspekten zu der technischen Lösung zu gelangen und zweitens ein Konzept der unterschiedlichen Sichten bzw. Fragestellungen. Hierbei entsprechen die Fragestellungen von Zachman Daten und Funktion dem Datenkonzept bei ARIS. Ebenso stimmen die Fragestellungen Netzwerk und Menschen mit der Organisationssicht und Zeit und Motivation mit der Funktionssicht überein [Sch05].

Neben den Gemeinsamkeiten finden sich aber auch Unterschiede in den beiden Ansätze vor allem in ihrer Fokussierung wieder:

Das Zachman Framework legt seinen Schwerpunkt auf die Identifikation unterschiedlicher Perspektiven auf Informationssysteme, um ein Medium für die verbesserte Kommunikation zwischen diesen Sichten bereitzustellen. Das Bewusstsein für verschiedene Rollen führt zu einer vollständigen Dokumentation der Architektur aus allen Blickwinkeln. Eine Verbesserung der Kommunikation wird durch die Möglichkeit der einzelnen Person ihre Sicht in den größeren Kontext einordnen zu können erreicht. Im Gegensatz zum ARIS Ansatz ist die strategische Planung als eigener Aspekt im Framework enthalten. Nachteilig wirkt sich bei diesem Schwerpunkt die, durch die eindeutige Differenzierung aller Sichten entstehende Komplexität aus, die der Simplizität und Klarheit des Frameworks entgegenwirkt [Fra94].

Bei ARIS liegt der Fokus auf der betriebswirtschaftlichen Sicht des Unternehmens. Es findet sich kein mit Zachman vergleichbares Perspektivenkonzept, welches eine an-

schauliche Modellierung für die an der Entwicklung und Nutzung von Informationssystemen beteiligten Personen verspricht. Die Ausrichtung erfolgt trotz des betriebswirtschaftlichen Ansatzes - durch die Analyse von Vorgangsketten - weitestgehend an den Bedürfnissen von Software-Ingenieuren [Fra94].

Obwohl das Zachman Framework neutral gegenüber Werkzeugen, Methodiken, Prozessen und Modellierungstechniken entwickelt worden ist, basiert es doch auf Annahmen, die daten- und prozessbasierte Methoden implizieren. Die Bestrebung des Zachman Frameworks liegt in der Integration aller Teilsichten zu einem Gesamtmodell durch Abbildung aller Modelle auf eine gemeinsame Sprache[9]. Diese Einführung einer gemeinsamen Sprache würde die Verwaltung des Gesamtmodells in einem Repository und die Definition von Transformationsregeln ermöglichen [Fra94].

Im ARIS Ansatz wird ein Metamodell für jede Sicht-Phasen-Kombination spezifiziert, die zu einem integrierten Metamodell zusammengesetzt werden können [Sch05]. Das Zachman Framework definiert im Gegensatz dazu nur Teilmodelle für das System und Business Model. Eine Zusammenführung aller Modelle zu einem integrierten Metamodell ist bei Zachman aufgrund der fehlenden Metamodelle nicht möglich.

Das Design und Management einer Enterprise Architecture ist ohne Werkzeugunterstützung nicht realisierbar. Kommerzielle Werkzeuge basieren auf den hier vorgestellten Ansätzen. So bildet der ARIS Ansatz die Basis für das ARIS Toolset der *IDS Scheer* und das Zachman Framework ist beispielsweise im System Architect von *Telelogic AB* umgesetzt.

Erst die Existenz von Werkzeugen zur Erstellung und Pflege einer Enterprise Architecture ermöglicht deren erfolgreiche Einführung in ein Unternehmen. Der Nutzen einer Enterprise Architekture hängt von ihrer Datenqualität ab. Ein hoher Aktualisierungsgrad der zu einer verbesserten Datenqualität und aktuellen Repositories führt, ermöglicht eine automatische Generierung von Modellen der Enterprise Architecture. Einem verwandten Themengebiet widmet sich der Bereich der Softwarekartographie, auf den im folgenden Abschnitt näher eingegangen werden soll.

2.2 Softwarekartographie

Wie in dem vorangegangenen Kapitel erläutert wurde, besteht der Hauptnutzen der Einführung einer Enterprise Architecture aus dem Alignment von Business und IT. Die damit in Verbindung stehende Optimierung und Standardisierung (siehe Abschnitt 2.1.2) der Informationstechnologie führt zu einer Veränderung der Anwendungslandschaft[10] des Unternehmens. Die Softwarekartographie beschäftigt sich mit

[9] Sowa und Zachman empfehlen die Verwendung von "conceptual graphs".

[10] "Unter einer Anwendungslandschaft (engl. Application Landscape oder Software Application Landscape) versteht man die Gesamtheit aller betrieblichen Informationssysteme in einem Unternehmen." [seb05b]

Methoden und Modellen zur Beschreibung, Bewertung und Gestaltung von Anwendungslandschaften. In den folgenden Abschnitten soll eine kurze Einführung in den Bereich der Softwarekartographie - ihrer Ziele, Betrachtungsebenen, Softwarekarten und Anforderung an ein Werkzeug - gegeben werden.

2.2.1 Ziele der Softwarekartographie

Die Langlebigkeit und ständig wachsende Anzahl von Informationssystemen in einem Unternehmen führt zu immer komplexer werdenden Anwendungslandschaften. Das Ziel der Softwarekartographie liegt in der Bereitstellung von Methoden und Modellen zur Beschreibung, Bewertung und Gestaltung dieser Landschaften. Sie grenzt dabei an bereits existierende Wissenschaften und Domänen an, siehe Abbildung 2.5.

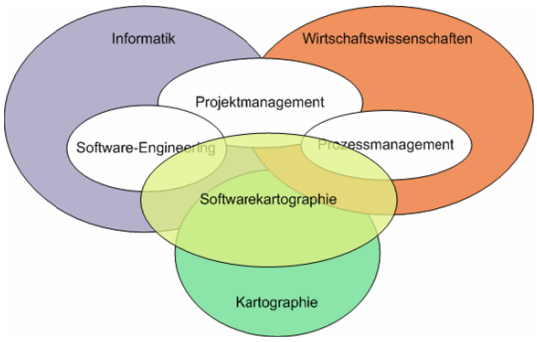


Abbildung 2.5: Domänen der Softwarekartographie [MW04a]

In der *Kartographie* finden sich Ansätze zur Erstellung von intuitiv verständlichen Karten und Leitlinien zur Verwendung von Farben und Formen. Die *Informatik* bietet mit dem Bereich des *Software-Engineering* in der Praxis erprobte Modelle und Methoden zur Dokumentation von Architekturen an, die eine Wiederverwendung im Bereich der Softwarekartographie nahelegen. Bei den *Wirtschaftswissenschaften* sind der Bereich des *Prozessmanagement* und die für die Softwarekartographie relevanten Kostenaspekte von Bedeutung. Das *Projektmanagement* als Querschnitts-Domäne zwischen *Informatik* und *Wirtschaftswissenschaften* steuert und plant die Evolution von Anwendungslandschaften und resultiert in zeitabhängigen Darstellungen, die mithilfe von Ist-, Plan- und Soll-Landschaften beschrieben werden können [MW04a].

Das Forschungsprojekt Softwarekartographie des Lehrstuhls *Software-Engineering für betriebliche Informationssysteme* am Institut für Informatik der TU München, in dessen Rahmen diese Arbeit entstanden ist, hat sich folgende Ziele gesetzt [LMW05a]:

- Bildung eines einheitlichen Begriffapparates zur Beschreibung von Anwendungslandschaften
- Entwurf eines geeigneten Informationsmodells (siehe Kapitel 3.1.2)

- Visualisierung von Anwendungslandschaften in Form von Softwarekarten (siehe Kapitel 2.2.3)
- Entwicklung eines Prototypen zur Generierung von Softwarekarten (siehe Kapitel 2.2.4)

Im Rahmen dieser Arbeit wird vor allem auf Vorraussetzungen für das vierte Ziel, die *Entwicklung eines Prototypen zur Generierung von Softwarekarten* eingegangen. Um ein geeignetes Werkzeug konzipieren zu können, muss eine Transformation der unterschiedlichen Informationsmodelle der Unternehmensarchitekturen in das Informationsmodell des Werkzeuges stattfinden. Die Analyse der Metamodellierungs-Konzepte (vgl Kapitel 4.1) und der möglichen Transformationssprachen in Kapitel 5.2 stellen den Kern dieser Arbeit dar.

2.2.2 Betrachtungsebenen der Softwarekartographie

Die Softwarekartographie liefert mit den Softwarekarten (siehe Kapitel 2.2.3) einen Beitrag zur Architekturbeschreibung im Rahmen des *Enterprise Architecture Managements*[11].. Die Softwarekarten sollen die Darstellung der gesamten Anwendungslandschaft ermöglichen, als Kommunikationsgrundlage dienen und gleichzeitig spezifische Fragestellungen beantworten, die sich aus den unterschiedlichen Perspektiven ergeben. Um auf die verschiedenen Anforderungen und Fragestellungen der Stakeholder[12] eingehen zu können, werden verschiedene Betrachtungsebenen (vgl. Sichtenkonzept des Zachman Frameworks 2.2) wie in Abbildung 2.6 dargestellt, benötigt [LMW05c].

Nach Matthes et al. [MW04a, MW04b, LMW05c] lassen sich drei Betrachtungsebenen in der Softwarekartographie unterscheiden:

Auf der obersten, strategischen Ebene (”Warum?”) finden sich die Unternehmensziele und -strategien eines Unternehmens. IT-Strategien oder neue gesetzliche Regelungen fordern Veränderungen von Geschäftsprozessen und wirken sich somit auf die Anwendungslandschaft des Unternehmens aus. Exemplarisch sei die IT-Strategie ”Forderung nach einer Reduzierung der Individualsoftware und Beseitigung von sogenannten Insellösungen” genannt, die einen direkten Einfluss auf die Anwendungslandschaft ausübt.

Geschäftsprozesse und Geschäftsobjekte sind auf der operativen, mittleren Ebene (”Was?”) anzusiedeln. Änderungen und Neuerungen die in diesem Bereich entstehen,

[11] ”*Enterprise architecture management* is a continuous and iterative process controlling and improving the existing and planned IT support for an organization. The process not only considers the information technology (IT) of the enterprise, also business processes, business goals, strategies etc. are considered in order to build a holistic and integrated view on the enterprise. Goal is a common vision regarding the status quo of business and IT as well as of opportunities and problems arising from these fields, used as a basis for a continually aligned steering of IT and business” [seb05a]

[12] ”Ein Stakeholder ist ein Individuum, ein Team oder eine Organisation, das/die Interessen (engl. Concern) an dem zu analysierendem System hat/haben” [IEE00, seb05b].

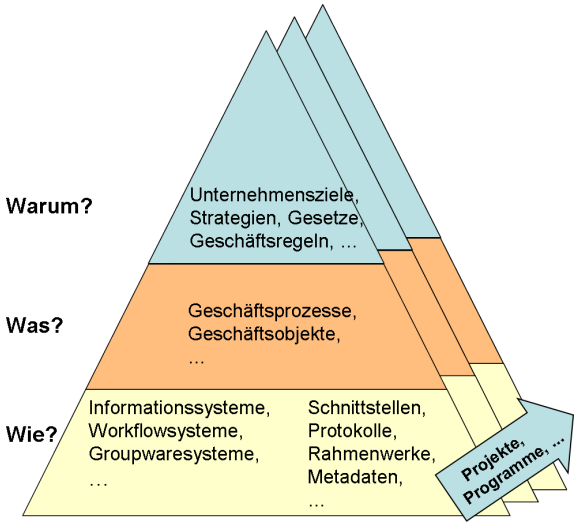


Abbildung 2.6: Betrachtungsebenen der Softwarekartographie [MW04a]

z.B. durch kürzere Produktionszyklen und sich wandelnde Kundenanforderungen, haben einen direkten Einfluss auf die Anwendungslandschaft. Um den neuen Anforderungen entsprechen zu können, müssen Informationssysteme adaptiert, abgelöst oder sogar neu entwickelt werden.

Der Implementierung von Geschäftsprozessen und Geschäftsobjekten und ihrer Unterstützung durch Informationssysteme widmet sich die unterste, technische Ebene (”Wie?”). Die Fokussierung dieser Ebene liegt auf den Technologien, Software-Architekturen, Schnittstellen, etc. der Informationssysteme.

Die Summe der vorgestellten drei Betrachtungsebenen ermöglichen eine statische Analyse der Anwendungslandschaft, die noch um die dynamischen Aspekte - ausgelöst durch Adaption, Ablösung oder Neuentwicklung von Informationssystemen durch geänderte Rahmenbedingungen, Geschäftsprozesse oder -ziele -, die die Evolution der Anwendungslandschaft beschreiben, erweitert werden muss [LMW05c].

Zusätzlich zu diesen drei Betrachtungsebenen lassen sich nach Matthes et al. [MW04a] fünf Aspekte für die Analyse von Anwendungslandschaften unterscheiden:

- **Planerische Aspekte:** Darstellung der Evolution einer Anwendungslandschaft
- **Wirtschaftliche Aspekte:** Darstellung der verschiedenen Kostenarten, die im Lebenszyklus eines Informationssystems entstehen
- **Fachliche Aspekte:** Darstellung von Prozessen, Organisationseinheiten, Geschäftsobjekten, Anzahl Nutzer, etc.

- **Technische Aspekte:** Darstellung von Implementierungssprachen, Schnittstellen, genutzer Middleware, etc.
- **Operative Aspekte:** Darstellung des Betriebs von Informationssystemen und damit verbundenen Ereignissen

Kennzahlen[13] die diese Aspekte verdeutlichen, lassen sich auf Softwarekarten darstellen, um den spezifischen Fragestellungen gerecht zu werden.

2.2.3 Softwarekarten

"Eine Softwarekarte ist eine graphische Repräsentation der Anwendungslandschaft oder von Ausschnitten selbiger" [seb05b]. Sie soll Informationssysteme, für den jeweiligen Betrachter relevante Aspekte und die Beziehungen zwischen ihnen visualisieren. Den Aufbau einer Softwarekarte aus einem Kartengrund, auf den mehrere Schichten aufgelegt werden können, verdeutlicht Abbildung 2.7.

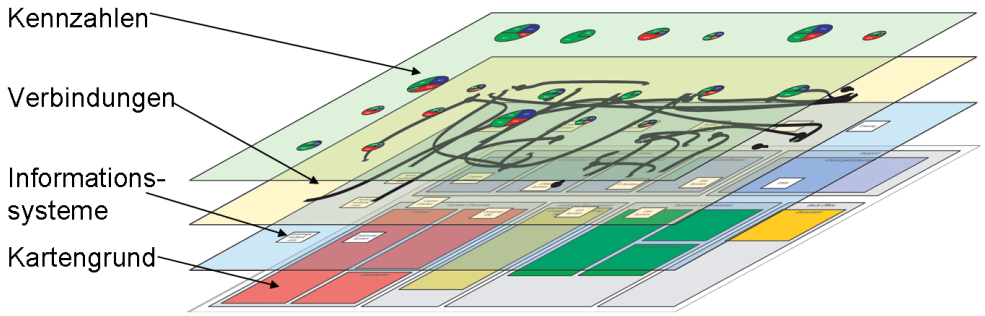


Abbildung 2.7: Schichtenaufbau einer Softwarekarte [LMW05b]

Durch das Schichtenprinzip wird eine Mehrfachverwendung des Kartengrundes ermöglicht, die zu einem erhöhten Wiedererkennungswert führt. Zu jeder Schicht, den Kartengrund ausgenommen, existiert genau eine Referenzschicht, die die Ausrichtung der einzelnen Elemente dieser Schicht bestimmt [LMW05a].

Die spezifischen Fragestellungen und entsprechenden Anforderungen, die die Softwarekarte beantworten soll, bestimmen die Auswahl und Platzierung der dargestellten

[13] "*Kennzahlen* (engl. Measures) erfassen Sachverhalte quantitativ und in konzentrierter Form. Merkmale einer Kennzahl sind Informationscharakter, Quantifizierbarkeit und Informationsform. Der Informationscharakter zeigt, dass eine Beurteilung von Sachverhalten und Zusammenhängen möglich ist. Quantifizierbarkeit bedeutet, dass Sachverhalte auf metrischen Skalen gemessen werden können und dadurch "genaue" Aussagen möglich sind. Die Informationsform führt dazu, dass komplexe Sachverhalte komprimiert und auf einfache Art dargestellt werden." [seb05b] Für weiterführende Informationen zu Kennzahlen in der Softwarekartographie sei auf die Arbeit von [Bey04] verwiesen.

Elemente. Der Aufbau der Softwarekarte erlaubt eine Klassifizierung derselbigen in unterschiedliche Typen[14]:

- Clusterkarte
- Proszessunterstützungskarte
- Intervallkarte
- Softwarekarte ohne Kartengrund zur Verortung

Die auf einer Softwarekarte angezeigten Informationen können durch das Ein- /Ausblenden und Zoom-In/Out gefiltert und somit die Informationsdichte variiert werden. Durch die unterschiedlichen Kartentypen und Darstellungsmöglichkeiten lassen sich für jeden Anwendungsfall geeignete Softwarekarten erzeugen [LMW05c].

2.2.4 Werkzeug zur Softwarekartographie

Eines der Ziele des Forschungsprojektes Softwarekartographie ist die Entwicklung eines Prototypen zur Generierung von Softwarekarten, der neben der Unterstützung der unterschiedlichen Kartentypen und dem mehrschichtigen Aufbau auf Basis des Kartengrundes noch weitere Anforderungen erfüllen muss [LMW05c]:

Import von Datenbeständen aus Repository-gestützten Anwendungen: Für die Erstellung von Softwarekarten sollen die bereits in einem Unternehmen existierenden Datenquellen genützt werden können. Die Hoheit über diese Daten soll bei den ursprünglichen Quellen verbleiben.

Abbildung der Importdaten auf ein generisches Informationsmodell für Anwendungslandschaften: Durch die fehlende Existenz eines einheitlichen Datenformates für den Export von Daten aus Repository-gestützen Anwendungen, wird eine Konvertierung der Importdaten in das Informationsmodell der Softwarekartographie erforderlich.

Abbildung der Objekte des Informationsmodells auf Gestaltungsmittel und -variablen: Das einheitliche Informationsmodell ermöglicht eine Abbildung seiner Objekte auf die Gestaltungsmittel und -variablen der Softwarekartographie.

Export der Softwarekarten in unterschiedliche Grafik- und Datenformate: Um den Nutzen der generierten Softwarekarte zu steigern, wäre eine Exportfunktion in unterschiedliche Grafik- und Datenformate wünschenswert, die es dem Nutzer der Karte erlauben sie z.B. in unterschiedliche Dokumente einbinden zu können.

Abbildung 2.8 stellt die Architektur eines Werkzeuges zur Softwarekartographie dar, das die oben genannten Anforderungen erfüllt und erläutert seine Beziehungen zu den Repository-gestützten Anwendungen und dem Intranet.

[14] Für detaillierte Beschreibungen der Softwarekartentypen sei auf [Bre05, LMW05c, MW04b, Sek05] verwiesen.

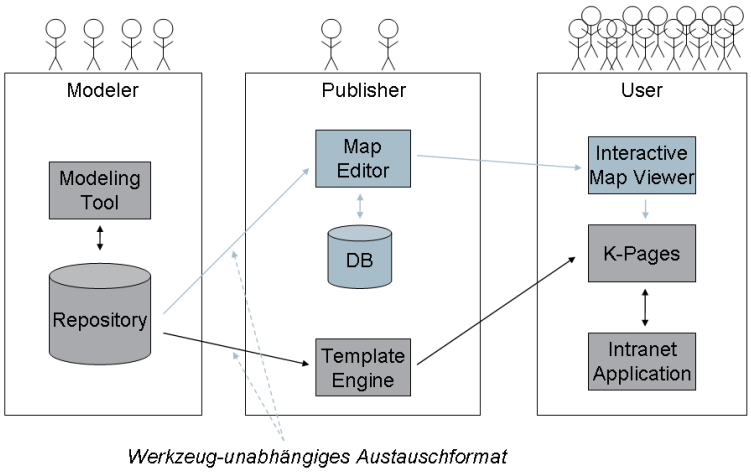


Abbildung 2.8: Architektur eines Werkzeuges zur Softwarekartographie [seb05d]

Die im allgemeinen in Unternehmen vorhandene Architektur sieht eine Erstellung von sogenannten Knowledge-Pages (K-Pages) mittels einer Template Engine vor, die die entsprechenden Daten aus den Repositorys gewinnt. Die erzeugten K-Pages, die manuell erstellte Softwarekarten, Informationen über Informationssysteme und die Anwendungslandschaft enthalten, werden über das Intranet möglichst vielen Nutzern zur Verfügung gestellt [LMW05c].

Dieser Workflow soll durch das Werkzeug zur Softwarekartographie dahingehend ergänzt werden, dass die Daten aus dem Repository einem Map Editor zur Verfügung gestellt werden, mit dessen Hilfe Softwarekarten erstellt werden und über die Export Funktion in die existierenden K-Pages eingebunden werden können [LMW05c].

Basis des Map Editors bilden ein geeignetes Informations- und Visualisierungsmodell, auf die im nächsten Kapitel näher eingegangen werden soll.

3 Modelle und Modellierungssprachen

Ziel dieses Kapitels ist eine Einführung in die theoretischen Grundlagen der Modellierung und Modelltransformationen. Zu diesem Zweck wird im ersten Abschnitt 3.1 einleitend eine Klassifizierung der unterschiedlichen Modelltypen, die in der Softwarekartographie eine Rolle spielen, vorgenommen. Der Vorstellung der Modellierungssprachen *UML*[1] und *MOF*[2] in Abschnitt 3.2 folgt abschließend eine Einführung in die *Model Driven Architecture* und das Themengebiet der Modelltransformationen in Abschnitt 3.3, die den Kern dieser Arbeit darstellen und die Grundlage für das Verständnis der folgenden Kapitel bilden.

3.1 Modelle in der Softwarekartographie

Ein in dieser Arbeit häufig verwendetes Wort ist der Begriff des Modells. Dieser soll in dem folgenden Abschnitt als Einleitung in das Themengebiet der Modellierung definiert werden.

3.1.1 Modellbegriff

Der Modellbegriff läßt sich nach [Sta73, Seite 131-133] durch drei Hauptmerkmale definieren:

- **Abbildungsmerkmal:** *Modelle sind stets Modelle von etwas*, d.h. Abbildungen oder Repräsentationen von natürlichen oder künstlichen Orginalen, die selbst wieder Modelle sein können.
- **Verkürzungsmerkmal:** *Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals* sondern beschränken sich auf solche, die den jeweiligen Modellerschaffern und/oder Modellbenutzern relevant erscheinen.

[1] Unified Modeling Language [OMG04c, OMG04d]
[2] Meta Object Facility [OMG04b]

- **Pragmatisches Merkmal:** *Modelle sind ihren Orginalen nicht per se eindeutig zugeordnet* sie erfüllen ihre Ersetzungsfunktion für bestimmte - erkennende und/oder handelnde, modellbenutzende - Subjekte, innerhalb bestimmter Zeitintervalle und unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen.

Aus dieser allgemeinen Definition von Stachowiak entwickelten sich nach [Tho05] drei unterschiedliche Ausprägungen des Modellbegriffs:

Der *mathematische Modellbegriff* entwickelte sich aus der Modelltheorie der mathematischen Logik. Das Modell stellt eine mittels mathematischer Zeichen formalisierte empirische Gegebenheit dar. Die verwendeten Symbole vertreten Quantitäten und die Relationen zwischen den Elementen des Systems werden in Form von mathematisch Gleichungen und Ungleichungen ausgedrückt.

In der Betriebswirtschaftslehre ist der *abbildungsorientierte Modellbegriff* von Bedeutung. Die zentrale Rolle dieser Auffassung spielt die Abbildung zwischen Original und Modell. Im *abbildungsorientierten Modellbegriff* wird im Gegensatz zu Stachowiaks Abbildungsmerkmal-Verständnis immer ein reales Problem (Mangel an Struktur) unterstellt und der Versuch unternommen die Realität homomorph, d.h. struktur- und verhaltensgetreu wiederzugeben. Im *abbildungsorientieren Modellbegriff* ist Modellierung als Strukturgebung (Erklärungsmodelle) zu verstehen.

Im *konstruktionsorientierten Modellbegriff* wird eine subjektive Problemdefinition in den Vordergrund gerückt. Modelle werden in der *konstruktionsorientierten* Auffassung nicht als Rekonstruktion vorgegebener Strukturkomplexe gesehen, sondern als Lösungskonstruktionen (Gestaltungsmodelle) subjektiver Problemstellungen aufgefasst.

Für den weiteren Verlauf der Arbeit sind vor allem der *abbildungsorientierte* und *konstruktionsorientierte* Modellbegriff von Bedeutung.

3.1.2 Objektmodell, Informationsmodell, Metamodell

Um die Komplexität einer Anwendungslandschaft beherrschen zu können wird das Konzept der Modellierung angewandt. Eine Komplexitätsreduzierung wird dabei durch die unterschiedlichen Modelltypen erreicht. Die nachfolgenden Abschnitte sollen die, für die weitere Arbeit relevanten Modelltypen klassifizieren und eine kurze Einführung in ihren Aufbau und ihre Bedeutung für die Softwarekartographie geben.

Die Erarbeitung eines Modells erfolgt dabei nach Esser in folgenden Schritten [Ess02]:

Der zu modellierende Ausschnitt der Realität wird subjektiv interpretierend erhoben und das dabei entstehende Objektsystem in ein *Objektmodell* abgebildet. Bei dieser Abbildung der Realität erfolgt durch Vereinfachung eine Komplexitätsreduzierung des *Objektmodells* gegenüber dem Diskursbereich[3]. Das Objektsystem wird unter Beach-

[3] Als Diskursbereich bezeichnet man den zu modellierenden Ausschnitt bzw. Teil der Realität.

tung des Zwecks und der Abstraktion von Einzelmerkmalen in ein Modellsystem überführt, dass durch ein *Informationsmodell* abgebildet wird. Dieses betrachtet nun nicht mehr die einzelnen Informationssysteme, sondern fasst sie zu Klassen oder Objekttypen zusammen. Die Syntax zur Erstellung eines *Informationsmodell* wird im *Metamodell*, das modellhaft das *Informationsmodell* beschreibt und gegen das sich die unterschiedlichen *Informationsmodelle* auf Konsistenz und Verhaltenstreue überprüfen lassen, festgelegt [Ess02].

In Abbildung 3.1 wird dieser Zusammenhang zwischen Realität und den unterschiedlichen Systemen und Modellen visualisiert.

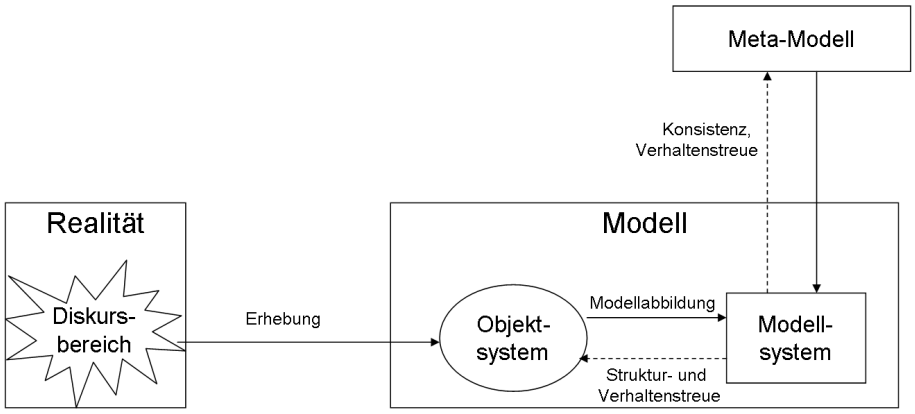


Abbildung 3.1: Beziehungen zwischen den Modellen und der Realität [Sin96]

Tabelle 3.1 fasst die unterschiedlichen Modelltypen zusammen und stellt exemplarisch einige Abbildungselemente jedes Modelltyps vor. Die in der Tabelle aufgeführten Modelltypen bilden von unten nach oben einen Abstraktionsvorgang, wobei jeweils die Begriffe einer untergeordneten Ebene Elemente der Begriffe der übergeordneten Ebene sind [Sch98].

Die für diese Arbeit bedeutsamen Modelle sind vor allem das *Informationsmodell* und das *Metamodell*. Auf die speziellen Eigenschaften dieser zwei Modelle und ihre Bedeutung für die Softwarekartographie soll in den folgenden Abschnitten näher eingegangen werden.

Informationsmodell

Der Begriff des Informationsmodells findet je nach Kontext unterschiedliche Verwendung. Teubner definiert Informationsmodelle wie folgt: ”Informationsmodelle bilden nicht nur einzelne Informationssysteme ab, sondern das Unternehmen bzw. die Unternehmensorganisation und stellen damit einen Bezugsrahmen für die Beschreibung und organisatorische und softwaretechnische Integration der Informations- bzw. An-

	Dargestellte Elemente	Beispiele
Metamodell	Meta-Objekttypen	Informationsobjekt, Funktion, Komponententyp
Informationsmodell	Anwendungsbezogene Objekttypen	Kunde, Auftragsbearbeitung, Workstation
Objektmodell	Einzelobjekte	Kunde Müller, Bearbeitungsauftrag 4711, Workstation WS75

Tabelle 3.1: Beispiele für die jeweiligen Modelltypen in Anlehnung an [Sch98]

wendungssysteme dar. (...)" [Teu99, Seite 53]. Nach Becker, Schütte [BS04] ist ein Informationssystem "die immaterielle Repräsentation des betrieblichen Objektsystems aus Sicht der in diesem verarbeiteten Informationen für Zwecke des Informationssystem- und Organisationsgestalters." [BS04, Seite 67]. Beiden Definitionen ist der Bezugaufbau zum betriebswirtschaftlichen System gemein.

Scheer definiert ein Informationsmodell aus der Sicht der praktischen Anwendung. Demnach enthält ein Informationsmodell auf der einen Seite die Konstrukte, die zur Beschreibung der Anwendung eingesetzt werden und liefert somit den Methodenrahmen. Auf der anderen Seite bildet das Informationsmodell das Datenmodell des Repositories und ermöglicht die Einstellung der Ergebnisse der Anwendungsmodellierung [Sch98].

Dieser Arbeit wird die folgende allgemeinere Definition zugrundegelegt: "Ein Informationsmodell (engl. Information model) ist ein Modell, welches die Beziehungen zwischen und Attribute von Informationsobjekten (Entitäten) darstellt. Ein Informationsobjekt (Entität) ist eine Abstraktion eines real existierenden Objektes (z.B. "Rechnung buchen:Geschäftsprozess", "BMW 318i:Produkt", "Fibu-System:Anwendungssystem" etc.), die für das Modell relevante Informationen über das Objekt aggregiert. Das Informationsmodell kategorisiert die Informationsobjekte - die Entitäten - mittels Entitätstypen und deren Beziehungen - den Relationen - mittels Relationstypen." [seb05b]

Im Gegensatz zu einem Informationsmodell für die Prozessmodellierung, das auf Funktionen, Ergebnisse, Daten etc. und deren Simulation fokussieren würde, liegt der Schwerpunkt des Informationsmodells für die Softwarekartographie auf Informationsobjekten von Anwendungslandschaften, deren Beziehungen und ihren relevanten Aspekten.

Das Informationsmodell für die Softwarekartographie hat in seiner Ausprägung in erster Linie abbildungsorientierten Charakter. Durch die Erstellung eines Informations-

modells wird versucht ein Erklärungsmodell für die Struktur der Anwendungslandschaft aufzubauen. Daneben hat ein Informationsmodell auch einen konstruktionsorientierten Aspekt, indem es Objekte, die in der visionären Anwendungslandschaft eine Rolle spielen einführt und benennt [Ess02].

Den Ausgangspunkt zur Erstellung eines Informationsmodell für die Softwarekartographie bilden die Stakeholder und ihre Interessen, die den Informationsbedarf charakterisieren. Stakeholder können Organisationseinheiten (Fachbereiche etc.), real existierende Personen (”Max Mustermann”) oder Rollen (Projektleiter etc.) sein. Die unterschiedlichen Interessen der Stakeholder führen zu Fragestellungen, wie z.B.: ”Welche Anwendung wird von einer bestimmten Organisationseinheit genutzt und welcher Geschäftsprozess wird durch diese Nutzung unterstützt?”. Diese Fragestellungen resultieren in Informationsobjekten, die geeignet in Beziehung gesetzt werden müssen [LMW05b]. Abbildung 3.2 visualisiert ein exemplarisches Informationsmodell für die vorangegangene Fragestellung.

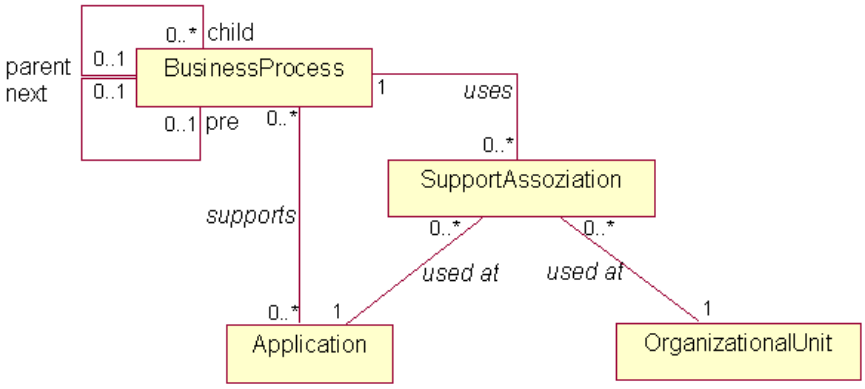


Abbildung 3.2: Exemplarisches Informationsmodell für eine Fragestellung

Aus diesen Fragestellungen und den daraus resultierenden Softwarekarten (vgl. Kapitel 2.2.3) ergibt sich die Forderung nach der Unterscheidung zwischen *optional* und *obligatorisch* zu pflegenden Entitäten, Assoziationen und Attributen von Entitäten.

Für die oben angeführte Fragestellung eignet sich eine Softwarekarte vom Typ *Prozessunterstützungskarte*[4], die Anwendungen entsprechend ihrer Unterstützung von Geschäftsprozessen und Organisationseinheiten anordnet. Daraus ergeben sich die Forderungen nach *obligatorischen* Assoziationen zwischen Anwendung und Organisationseinheit und zwischen Geschäftssystem und der Nutzungsbeziehung. Zusätzlich könnte der Status eines Informationsystems von Interesse sein und würde ein *optionales* Attribut darstellen, das auf einer zusätzlichen Schicht mittels Farbcodierung angezeigt werden könnte [LMW05a].

[4]Für weitere Informationen zu Kartentypen sei auf die Arbeiten von [Bre05, LMW05c, MW04b, Sek05] verwiesen.

Metamodell

Zur Konstruktion von Modellsystemen - und deren graphischer Darstellung als Informationssysteme - werden Kostrukte, Strukturen und syntaktische Beziehungsregeln zur Modellbildung benötigt, die in einem Metamodell beschrieben werden. Metamodelle stellen einerseits Baukästen zur Konstruktion einer Klasse von Modellen zur Verfügung und ermöglichen andererseits eine Validierung dieser Modelle. Sie enthalten Methodenwissen (Arbeitstechniken, Vorgehensweisen, Werkzeuge) zur Konstruktion von Modellsystemen [Ess02].

Ein Metamodell bezeichnet die modellhafte Beschreibung eines Modells unter einem bestimmten Aspekt, dem sogenannten Metaisierungsprinzip. Das Metaisierungsprinzip bezeichnet die gewählte Abstraktionsebene über der tatsächlichen Modellebene. Im Bereich der Softwarekartographie liegt der Fokus auf dem Metaisierungsprinzip der statischen Struktursemantik [Jec00].

In der Softwarekartographie ist das Metamodell, das die Sprache beschreibt mit der ein Modell definiert werden kann, vor allem für den Bereich der Enterprise Architecture-Tools von Bedeutung. Basierend auf dem Metamodell können Werkzeuge Unternehmensmodelle auf ihre Korrektheit und Vollständigkeit überprüfen und mit Hilfe von Modelltransformationen (siehe Kapitel 3.3) in ein geeignetes Informationsmodell zum Enterprise Architecture Management überführen [BNS03].

3.1.3 Semantisches und symbolisches Modell

Eine Reduzierung des Kosten- und Zeitaufwandes zur Erstellung und Pflege von Softwarekarten lässt sich durch die Konzeption eines Werkzeuges zur regelbasierten, automatischen Erstellung von Softwarekarten erreichen. Die Konzeption dieses Werkzeuges verlangt eine klare Trennung zwischen dem zu visualisierenden Inhalt und seiner Repräsentation, d.h. von semantischen und symbolischen Modell, wie sie Abbildung 3.3 zeigt.

Das semantische Modell befasst sich mit der Struktur und Bedeutung der Architektur. Der Fokus liegt auf den eigentlichen Informationen - den Informationsobjekten - die über die Anwendungslandschaft verfügbar sind, unabhängig von deren Darstellungsformen. Auf der Seite des symbolischen Modells findet sich eine Beschreibung der Repäsentation der Architektur. Hier erfolgt die Darstellung der Informationsobjekte des semantischen Modells über ein geeignetes Visualisierungsmodell. Die Erfahrbarkeit und Kommunizierbarkeit eines Informationsobjektes ist abhängig von der Existenz eines symbolischen Modells, dementsprechen ist ein semantisches Modell ohne mindestens ein dazugehöriges symbolisches Modell nicht sinnvoll [LMW05b, Tor04].

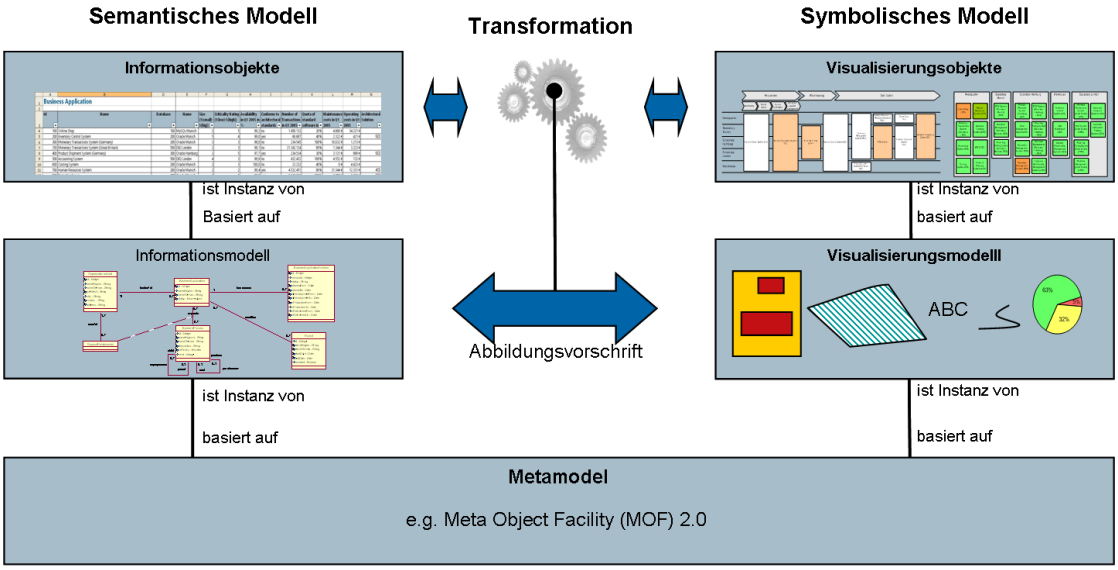


Abbildung 3.3: Zusammenhang zwischen semantischem und symbolischem Modell nach [LMW05b]

Semantisches Modell

Das semantische Modell kann in drei Ebenen untergliedert werden. Auf der obersten Ebene finden sich die instanziierten Modelle, die die Informationsobjekte, d.h. Abstraktionen real existierender Objekte enthalten. Diese Modelle werden durch ein geeignetes Informationsmodell auf der mittleren Ebene beschrieben. Der Zusammenhang zwischen einem Informationsobjekt und einem Entitätstyp des Informationsmodells entspricht dabei genau der Beziehung eines Objekts zu seiner Klasse in der objektorientierten Programmierung. Das Informationsmodell basiert auf einem Metamodell auf der untersten Ebene. Die Analyse verschiedener Informationsmodelle und die Verifikation des MOF-Ansatzes für die Modellierung von Informationsmodellen zur Softwarekartographie stellen eine zentrale Aufgabe dieser Arbeit (vgl. Kapitel 4) dar.

Symbolisches Modell

Um die Kommunizierbarkeit und Erfahrbarkeit des semantischen Modell zu gewährleisten, steht ihm das symbolische Modell ergänzend gegenüber. Auf oberster Ebene im semantischen Modell finden sich die instanziierten Softwarekarten, die die Visulaisierungsobjekte enthalten. Die Ausrichtung der Elemente der Softwarekarte erfolgt durch ein Visualisierungsmodell, das Vorgaben über die Organisation der unterschiedlichen Darstellungen enthält und auf der mittleren Ebene anzusiedeln ist. Die Informationsobjekte des semantischen Modells werden über ein geeignetes Visualisierungsmodell

dargestellt und ermöglichen so die Kommunizierbarkeit der Informationen nicht nur durch geeignete Karten, wie in Abbildung 3.3 dargestellt, sondern auch durch tabellarische Reports etc. Die Anforderungen an das Visualisierungsmodell ergeben sich aus den Anforderungen zur Erstellung von Softwarekarten (vgl Kapitel 2.2.3). Um eine Konformität und Kompatibilität verschiedener Visulaisierungsmodelle erzielen zu können, wird auf unterster Ebene analog zum semantischen Modell ein Metamodell formalisiert.

Transformation

Um eine weitgehend automatisierte Visualisierung der Anwendungslandschaft und eine Konsistenz in der Visualisierung erreichen zu können, bedarf es einer Verbindung des semantischen Modells mit dem symbolischen Modell. Diese Transformation der über eine Unternehmensarchitektur vorhandenen Informationen in eine geeignete Visualisierung verlangt eine genaue Spezifikation, welche Elemente auf Seite des semantischen Modells in welche Elemente auf der Seite des symbolischen Modells abgebildet werden. Dabei soll dem Modellierer eine gewisse Gestaltungsfreiheit erhalten bleiben, um eine - in Hinblick auf ästhetische Kriterien und Lesbarkeit - optimierte Darstellung zu erhalten. Der verfälschungsfreie Transport von Informationen kann hierbei nur gewährleistet werden, wenn händische Veränderungen unterbunden werden, oder zu entsprechenden Änderungen an den Informationsobjekten führen [LMW05b].

3.2 Modellierungssprachen

Zur Darstellung der in den vorangegangenen Abschnitten definierten Modelle, wird eine geeignete Modellierungssprache benötigt. Das Umfeld der modellbasierten Entwicklung auf Basis objektorientierten Modelle ist vor allem durch die Standardisierungsbemühungen der *Object Management Group* (OMG) geprägt. In den folgenden Abschnitten werden einige für die Arbeit relevante Standards und Initiativen der OMG vorgestellt.

Meta Object Facility (MOF)

Klassen von Modellen lassen sich durch Metamodelle spezifizieren, indem eine Menge gültiger Strukturen dieser Modelle festgelegt wird. Diese Syntaxdefinition wird ebenfalls für Metamodelle benötigt um eine gültige Klasse von Metamodellen festlegen zu können. Die Spezifikation der *Meta Object Facility* (MOF) [OMG04b] der *OMG* beschreibt eine abstrakte Sprache und ein Grundgerüst zur Erzeugung und Spezifikation von plattformunabhängigen Metamodellen.

Die MOF sieht vier[5] Ebenen von Modellen vor, in denen ein Modell der einen Ebene beschrieben wird, als Instanz des Metamodells der direkt darüber liegenden Ebene. Auf oberster Ebene definiert sich das MOF Modell rekursiv selbst, um eine endlose Zahl an Metaebenen zu verhindern. Abbildung 3.4 stellt diese vier Ebenen graphisch dar.

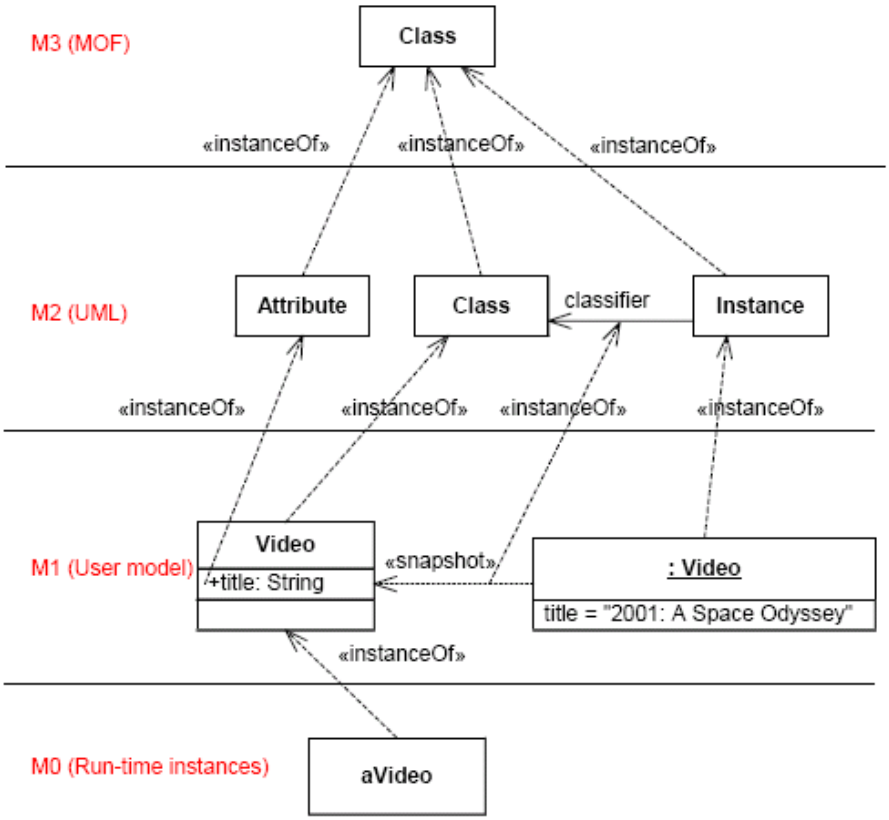


Abbildung 3.4: Die 4 Ebenen der MOF [OMG04c]

Auf der untersten Ebene M0 des MOF Frameworks finden sich Objekte der Realität, die durch Modelle der M1-Ebene beschrieben werden können. Ein Bespiel hierfür ist ein konkretes Objekt "aVideo" als Instanz der Klasse Video. Die Ebene M1, die auch als Modellebene bezeichnet wird beherbergt Benutzermodelle. So zum Beispiel UML-Modelle, durch die ein konkretes System spezifiziert wird. In dieser Ebene werden Daten der Informationsebene beschrieben, exemplarisch sei hier die Klasse Video genannt. Die Metamodelle bilden die Ebene M2, die die Definition und den Aufbau von Mo-

[5]Die MOF schreibt keine feste Anzahl von Ebenen vor. Sie legt fest, dass minimal zwei Ebenen vorhanden sein müssen, um die Navigation von einer Instanz zu ihrem Metaobjekt (Classifier) zu ermöglichen. Es gibt aber keine maximale Anzahl von Ebenen [OMG04b].

dellen beschreibt. Das bekannteste Beispiel eines M2 Modells ist das Metamodell der UML. Die oberste Ebene M3 enhält Meta-Metamodelle, die die Struktur und Semantik der M2 Modelle beschreiben. Auf dieser Ebene ist auch das MOF Meta-Metamodell zur Spezifikation von Metamodellen anzusiedeln [Mar04].

MOF 2.0 besteht aus zwei Hauptpaketen, dem *Essential MOF* (EMOF) und dem *Complete MOF* (CMOF). Dabei bildet EMOF eine Teilmenge des MOF, dessen primäres Ziel die Definition simpler Metamodelle mit einfachen Konzepten ist. Eine genaue Beschreibung der in EMOF enthaltenen Konzepte erfolgt in Kapitel 4.1.

CMOF erweitert das EMOF unter anderem durch explizite Assoziationskonzepte, Sichtbarkeiten und Constraints. Constraints ermöglichen die zusätzliche Definition von Einschränkungen mit Hilfe der *Object Constraint Language* (OCL) [OMG05a], die einen Teil der CMOF bildet. Die Visualisierung der MOF erfolgt durch die *Unified Modelling Language* (UML).

Unified Modeling Language (UML)

Die *Unified Modeling Language* (UML) [OMG04c, OMG04d] ist eine graphische Modellierungs- und Spezifikationssprache mit Schwerpunkt auf dem Entwurf von Software-Systemen. Sie definiert verschiedene Diagrammarten, denen eine abstrakte Syntax zugrunde liegt, die sich in einem gemeinsamen Modell, dem UML-Metamodell niederschlägt. Die UML ist eine offene Methode zur Spezifikation, Visualisierung, Konstruktion und Dokumentation von Modellen für Software-Systemen [Oes04].

Die UML 2.0 ist formal in folgende Teilbereiche gegliedert:

- **Infrastructure:** Kern der Architektur, Profile und Stereotypen
- **Superstructure:** Statische und dynamische Modellelemente

Im Bereich betrieblicher Informationssysteme und objektorientierter Anwendungen hat sich die UML mittlerweile zu einem de-facto Standard entwickelt. In diesem Bereich und auch in der weiteren Arbeit sind vor allem die Strukturdiagramme von Bedeutung [Mar04].

Zwischen UML und MOF existieren unterschiedliche Abhängigkeiten: Die Definition der MOF 2.0 basiert auf dem Paket *Core* der UML 2.0 *Infrastructure*. Neben dem Zusammenhang von MOF und UML als Meta-Metasprache der Metasprache, wird außerdem die UML zur Visualisierung der MOF verwendet.

3.3 Model Driven Architecture (MDA)

Die *Model Driven Architecture* (MDA) [MM03, KWB03] ist eine Initiative der *OMG* zur modellbasierten Entwicklung von Software-Systemen. Ihr Ziel ist die Wiederverwendung von Modellen, die im Verlauf der Entwicklung von Software-Systemen entstehen. Die *OMG* verfolgt dabei den Ansatz, dass ein einziges Modell eines Software-Systems eine zu ungenaue Abbildung darstellt, um daraus ein Software-System konzipieren zu können. Sie verwendet unterschiedliche Modelle zur Spezifikation von Systemen auf verschiedenen Abstraktionsebenen:

- **Computation Independent Model** (CIM): beschreibt ein Software-System auf fachlicher Ebene. Es dient zur Verständigung zwischen Softwarearchitekten und Anwendern über Leistungsumfang und Anforderungen.
- **Platform Independent Model** (PIM): modelliert die Funktionalität und Struktur eines Systems unabhängig von der gegebenen Plattform[6].
- **Platform Specific Model** (PSM): bezeichnet ein Modell eines Software-Systems unter Berücksichtigung von plattformspezifischen Details.

Die Verwendung von UML als Modellierungssprache, wie von der *OMG* empfohlen, ermöglicht die Definition von Abbildungsvorschriften auf Basis der Metamodelle der Modelle verschiedener Abstraktionsebenen. Die *OMG* unterscheidet dabei zwei Transformationstypen:

- **Modelltransformationen** von einem Modell in ein anderes Modell (mode-to-model)
- **Codetransformationen** von einem Modell in den Code (model-to-code)

Die Transformationen zwischen PIMs stellen hierbei Verfeinerungsschritte dar und die Transformation eines PIMs in ein PSM eine Abbildung eines plattformunabhängigen Modells auf ein Modell einer bestimmten Infrastruktur.

Der Vorteil des MDA Ansatzes liegt in der Möglichkeit komplexe Systeme durch den hohen Abstraktionsgrad in kompakten Modellen darstellen zu können. Daneben verspricht der MDA Ansatz eine Wiederverwendbarkeit der abstrakten Modelle, z.B. für die Implementierung der gleichen Anwendung auf unterschiedlichen Plattformen. Die Transformation von Modellen unterschiedlichen Abstraktionsgrades erspart zusätzlichen Entwicklungsaufwand und sichert die Konsistenz der Modelle untereinander.

Der MDA-Ansatz sieht sogenannte "Mappings" zwischen Metamodellen verschiedenen Abstraktionsgrades vor, durch die die Abbildung von Instanzen von Metamodellen aufeinander festgelegt wird. Eine standardisierte Spezifikationstechnik für die Definition von Transformationen zwischen MDA-Modellen existiert derzeit noch nicht und

[6] Die technologische Infrastruktur, auf der eine Anwendung basiert.

ist Gegenstand des *Request for Proposal: Query / Views / Transformations* [OMG05b] (QVT) der *OMG*. Die Empfehlung der OMG für die Verwendung von UML als Modellierungssprache, lässt auf die Spezifikation von UML-Modelltransformationen schließen, die durch MOF Metamodelle beschrieben werden können. Das Ziel des RFPs ist die Definition einer einheitlichen Sprache zur Überführung von Modellen, deren Metamodelle durch MOF spezifiziert wurden.

Die Konzeption eines Werkzeuges zur Softwarekartographie verlangt ein geeignetes Informationsmodell im semantischen Modell, um eine Abbildung auf Softwarekarten mit Hilfe des symbolischen Modells verwirklichen zu können. Die Informationsmodelle verschiedener Unternehmen, müssen sich in dieses Informationsmodell abbilden lassen, um eine Nutzung des Werkzeuges zu ermöglichen. Die Analyse verschiedener Informationsmodelle von Unternehmen und Werkzeugherstellern zur Entwicklung eines integrierten Metamodells für Informationsmodelle und anschließende Transformation unter Verwendung von Modell-basierten Transformationssprachen bilden den folgenden Hauptteil dieser Arbeit.

4 Analyse von Informationsmodellen

Im Rahmen des Forschungsprojekts Softwarekartographie wurden in Zusammenarbeit mit mehreren großen und mittelständischen Unternehmen verschiedene Softwarekarten untersucht und eine erste Klassifizierung von Softwarekarten und Views definiert [LMW05b]. In einem weiteren Schritt wurden unternehmensspezifische Informationsmodelle analysiert und neue Informationsmodelle für das Enterprise Architecture Management entwickelt [Bre05, Hal04, Lau04]. Die Konzeption eines Werkzeuges zur Softwarekartographie basiert auf einem geeigneten Informationsmodell, das den Ausgangspunkt für die Transformation der Informationsobjekte in Visualisierungsobjekte bildet (siehe Abschnitt 2.2.4).

In einem ersten Schritt sollen in dem folgenden Kapitel 4.1 die unterschiedlichen Metamodellierungskonzepte zu den in Kapitel 3.2 vorgestellten Modellierungssprachen zugeordnet werden. Darauf aufbauend werden in einem zweiten Schritt die erstellten Informationsmodelle von Unternehmen auf die verwendeten Metamodellierungskonzepte in Kapitel 4.2 untersucht. Der Untersuchung der Informationsmodelle von Werkzeugherstellern in Kapitel 4.3 folgt eine Analyse der verwendeten Metamodellierungskonzepte standardisierter Informationsmodelle der Modelle *CIM*[1]. [DMT05] und *ITPMF*[2]. [OMG04a] in Kapitel 4.4. Ein Resümee, das die in der Analyse gewonnenen Erfahrungen über die Verwendung von Metamodellierungskonzepten und eine Empfehlung für die zu verwendende Metamodellierungssprache als Basis für die Transformation reflektiert, bildet in Kapitel 4.5 den Abschluss des vierten Kapitels.

4.1 Metamodellierungskonzepte und ihre Einordnung in Modellierungssprachen

Dieses Kapitel soll einen Überblick über die unterschiedlichen Metamodellierungskonzepte geben und sie den in Kapitel 3.2 vorgestellten Modellierungssprachen zuordnen. Zu diesem Zweck soll zunächst der Zusammenhang zwischen UML und MOF näher erläutert und anschließend die Metamodellierungskonzepte von EMOF, CMOF und UML vorgestellt werden.

[1] Common Information Model der *Distributed Management Task Force* (DMTF)
[2] IT Portfolio Management Facility der *OMG*

4.1.1 Zusammenhang zwischen UML und MOF

Beide Modellierungssprachen sind parallel entwickelte Standardisierungen der *Object Management Group*. Bereits bei der Entwicklung der beiden Standards wurde der Kern der Sprachversion UML 1.x als Grundlage für die MOF verwendet. Beide Spezifikationen enthielten viele ähnliche Modellierungselemente, die sich aber im Detail unterschieden. Mit Aufnahme des Normierungsprozesses für die Version 2.0 wurde das Ziel verfolgt, den Kern von UML und MOF - auf der Grundlage einer einheitlichen Menge von Basiskonzepten - vollständig zu vereinen. Diese Menge an einheitlichen Basiskonzepten bildet genau das Paket der *UML 2.0 Infrastructure* [OMG04c], die die gemeinsame Grundlage für die UML Sprachdefinition und die MOF der Version 2.0 bildet [KHB05]. Abbildung 4.1 stellt die verschiedenen Subpakete des Core-Pakets dar.

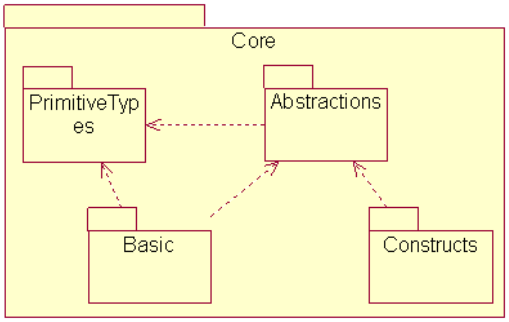


Abbildung 4.1: Das Paket Core der UML und seine Subpakete [OMG04c]

Das Subpaket *Primitive Types* enthält vordefinierte Datentypen, wie Integer, Boolean, String und Unlimited natural, die für die Metamodellierung benötigt werden. Das Paket *Basic* definiert die Grundbegriffe Klasse, Attribut, Operation, Paket. Weiterführende Konzepte wie z.B. Generalisierung, Einschränkung, Multiplizitäten[3] und Sichtbarkeiten, werden durch das Paket *Abstractions* eingeführt. Das Assoziationskonzept ist in dem Paket *Constructs* enthalten. Die Benutzung der in den einzelnen Pakten definierten Konzepte beruht auf Importbeziehungen zwischen diesen Paketen [OMG04c].

Die MOF 2.0 besteht aus zwei Hauptpaketen, dem Essential MOF (EMOF) und dem Complete MOF (CMOF). Die Abbildung 4.2 visualisiert die Beziehungen zwischen den Paketen der UML 2.0 Infrastructure und der MOF.

Die Grundlage von EMOF bildet das *Basic* Paket der UML Infrastructure, das elementare Metamodellierungskonzepte zur Verfügung stellt. Durch die Zusammenführung mit den Paketen *Identity*, *Reflection* und *Extension* werden außerdem Konzepte zur Er-

[3] Die Begriffe Multiplizität und Kardinalität werden fälschlicherweise oft synonym verwendet, eine Multiplizität gibt den Bereich erlaubter Objekte auf Klassenebene an, während eine Kardinalität die tatsächliche Anzahl an Objekten auf Instanzebene beschreibt [OMG04c].

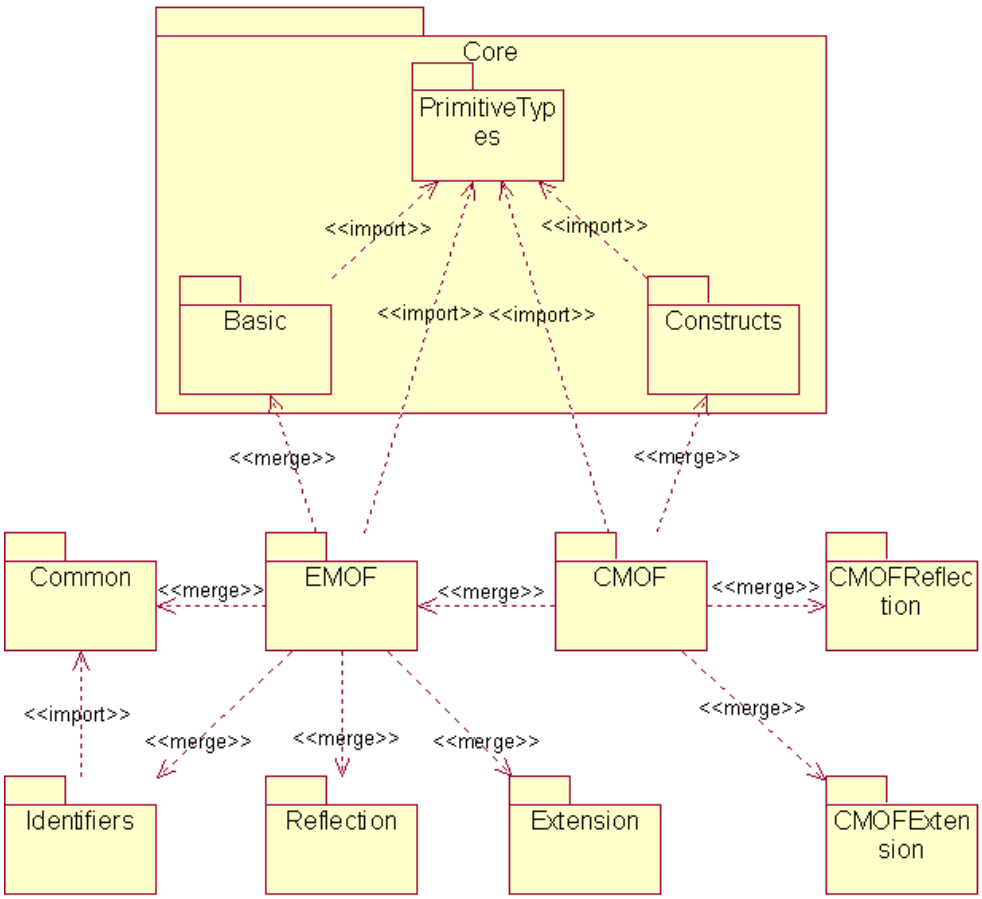


Abbildung 4.2: Beziehung zwischen UML Core und MOF [OMG04b]

forschung, Manipulierung, Identifizierung und Erweiterung von Metaobjekten und -daten bereitgestellt. Das Reflection-Paket führt den Begriff der Metaobjektinstanz (Objekt) ein. Metaobjekte haben den Vorteil, dass sie ohne vorheriges Wissen über die objektspezifischen Eigenschaften verwendet werden können. Die Erforschung und Änderung von Metaobjekten und Metadaten wird durch das Reflection-Paket ermöglicht. Mit Hilfe des Identity-Pakets wird jedem Objekt ein eindeutiger Identifikator zugeordnet, der es von anderen Objekten unterscheidbar macht. Das Extension-Paket bietet einen Erweiterungsmechanismus, der die Annotation einer Sammlung von Name-Werte-Paaren - sogenannten Tags - an Modellelementen ermöglicht.

Die CMOF definiert durch die Vereinigung des Pakets *Constructs* der UML Infrastructure mit den Konzepten von EMOF eine erweiterte Spezifikationssprache. Ein primäres Ziel der EMOF ist "to allow simple metamodels to be defined using simple concepts while supporting extensions (by the usual class extension mechanism in MOF) for more sophisticated metamodeling using CMOF." [OMG04b]

4.1.2 Einordnung der Metamodellierungskonzepte

Die folgenden Abschnitte sollen eine Übersicht der unterstützen Metamodellierungskonzepte der Modellierungssprachen EMOF, CMOF und UML liefern, um darauf aufbauend eine Analyse der verwendeten Konzepte in existierenden und relevanten Informationsmodellen zu ermöglichen.

Metamodellierungskonzepte des EMOF

Abbildung 4.3 zeigt ein vereinfachtes[4] Klassendiagramm, der in EMOF enthaltenen Metamodellierungskonzepte, zur Konzeption simpler Metamodelle.

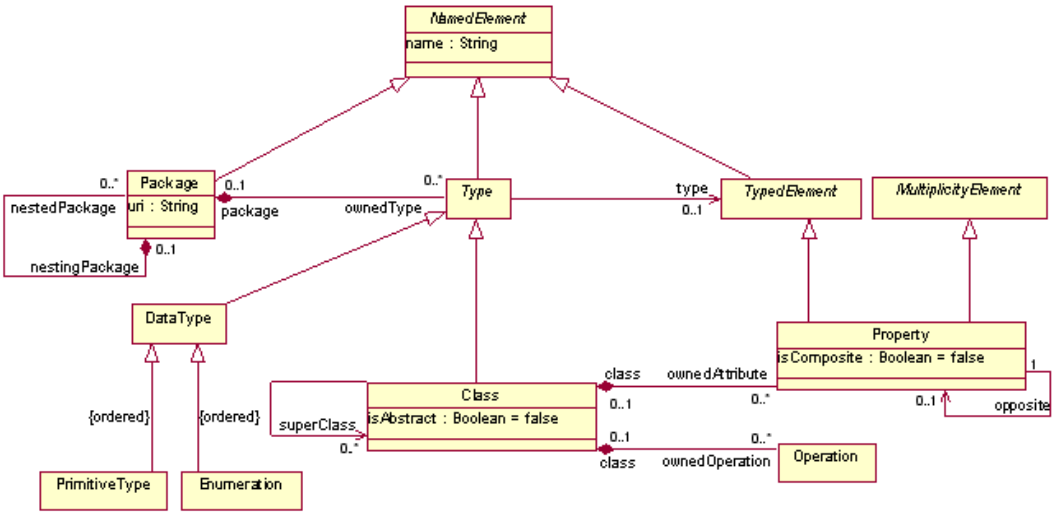


Abbildung 4.3: Klassendiagramm der EMOF nach [OMG04b, OMG04c]

Die einzelnen in Abbildung 4.3 dargestellten Metamodellierungskonzepte sollen im folgenden näher erläutert werden [OMG04b, OMG04c, Hit05, Bic04]:

NamedElement (Benennbares Element): NamedElement ist eine abstrakte Superklasse, die Modellelemente repräsentiert, die einen Namen besitzen können. Der Name dient zur eindeutigen Identifikation eines Elements im Modell.

Type (Typ): Ein Type bildet eine abstrakte Klasse, die eine Menge von Werten darstellt. Die Werte legen den Wertebereich eines typisierten Elements fest.

TypedElement (Typisiertes Element): Ein TypedElement beschreibt die Menge von Modellelementen, die einen Typ besitzen.

[4] Diagrammelemente, die für die weitere Arbeit nicht von Bedeutung sind, wurden aus Gründen der Übersichtlichkeit weggelassen.

MultiplicityElement (Muliplizitäten): Ein MultiplicityElement repräsentiert ein Intervall von ganzzahligen positiven Werten (inklusive der Null), welches Grenzen für die minimale und maximale Anzahl von Instanziierungen eines Elementes beschreibt.

Class (Klasse): Klassen gehören zu den elementarsten Modellierungskonstrukten. Instanzen von Klassen (Objekte) haben eine Identität, einen Zustand und ein Verhalten. Der Zustand eines Objektes wird durch die Attribute der Klasse vorgegeben, analog dazu wird das Verhalten eines Objektes durch die Operationen der Klasse beschrieben. Klassen können als *abstrakt* gekennzeichnet werden. Eine abstrakte Klasse kann nicht instanziiert werden. Sie spezifiziert im Allgemeinen Teile der Struktur und/oder des Verhaltens von Objekten. Das Konzept der abstrakten Klasse wird meist im Zusammenhang mit der Vererbung verwendet. Unter Vererbung versteht man die Übernahme der Attribute und Operationen der Superklasse (vererbende Klasse) in die Subklasse (erbende Klasse). MOF bietet hierbei die Möglichkeit eine Subklasse von mehreren Superklassen abzuleiten, dieses Konzept wird als Mehrfachvererbung bezeichnet. Instanzen der Subklasse können überall anstatt der Instanzen der Superklasse verwendet werden, weil sie deren Merkmale erben.

Property (Attribut): Attribute repräsentieren typisierte Eigenschaften einer Klasse, sie werden durch die Angabe eines Namens, eines Typs und einer Multiplizität spezifiziert. Der Typ eines Attributs kann eine MOF-Klasse oder ein MOF-Datentyp sein. Obwohl in EMOF das Assoziationskonzept nicht explizit enthalten ist, können bidirektionale Beziehungen zwischen Klassen durch ”verschränkte” Attribute nachgebildet werden, indem je ein Attribut des Typs als ”opposite” angegeben wird. Durch die Verwendung ”verschränkter” Attribute lassen sich binäre Assoziationen, Rollennamen, Aggregation und Komposition nachbilden. Die Multiplizitätsangaben kennzeichnen, ob Attribute optional, ein- oder mehrwertig sind. Sie legen obere und untere Schranken für die Anzahl der Werte fest, die Instanzen eines Attributs speichern können. Für jedes Attribut kann angegeben werden, ob es eine Kompositionsbeziehung repräsentiert, d.h. das Attribut wird erst nach der Erzeugung eines Objektes instanziiert und vor dem Löschen des Objektes entfernt.

Operation (Operation): Operationen spezifizieren das Verhalten von Objekten. Obwohl MOF zur Definition von Metamodellen spezifiziert wurde und in Metamodellen der Schwerpunkt auf der Modellierung von Strukturen liegt, enthält MOF das Metamodellierungskonzept Operation zur Spezifikation des Objektverhaltens. Da Operationen für die Modellierung von Informationsmodellen und bei der Transformation von Informationsmodellen keine Rolle spielen, wird das Metamodellierungskonzept der Operation an dieser Stelle nur aus Gründen der Vollständigkeit erwähnt und findet in dieser Arbeit keine weitere Beachtung.

DataType (Datentyp): Datentypen bilden die Grundbausteine der Modellierung und repräsentieren eine Menge von Werten. EMOF definiert im Paket PrimitiveType die vier Grunddatentypen *Boolean, Integer, String* und *UnlimitedNatural*. Dabei repräsentiert Boolean einen Aufzählungstyp mit den Werten true und false, Integer den Wertebereich der ganzen Zahlen, String die Menge aller Zeichenketten und UnlimitedNatural

die natürlichen Zahlen einschließlich des Zeichens *, zur Kennzeichnung des Wertes $+\infty$. Neben diesen Grunddatentypen können in EMOF Aufzählungen (engl. Enumerations), als strukturierte Datentypen verwendet werden. Datentypen sind in MOF über ihre Werte eindeutig zu identifizieren, die Definition von Datentypen ist nur in CMOF möglich. Das definieren eigener Typen, nicht Datentypen, mittels Klassenbildung ist auch in EMOF möglich.

Package (Paket): Pakete dienen der Organisation von Modellen durch die Bildung von logischen Einheiten. Pakete ermöglichen die Gruppierung von Typen und Paketen zum besseren Verständnis und zur Verwaltung von Modellen. Pakete können selbst Bestandteile anderer Pakete sein, wodurch eine Verschachtelung von Paketen ermöglicht wird. Dabei werden Pakete durch eine Kompositionsbeziehung miteinander verbunden, wobei das äußere Paket den Container darstellt, der das innere Paket enthält. Ein Paket darf sich selbst nicht enthalten, um eine rekursive Verschachtelung von Paketen zu verhindern.

Metamodellierungskonzepte des CMOF

CMOF baut auf Elementen des Pakets *Constructs* der UML Infrastructure auf (vgl. Abbildung 4.2) und definiert Metamodellierungskonzepte zur Spezifikation komplexer Metamodelle. Zu diesem Zweck erweitert CMOF die Konzepte von EMOF. Abbildung 4.4 zeigt ein vereinfachtes[5] Klassendiagramm von CMOF.

CMOF erweitert EMOF um die folgenden Metamodellierungskonzepte [OMG04b, OMG04c, Hit05, Bic04]:

Association (Assoziation): Assoziationen beschreiben Beziehungen zwischen typisierten Instanzen. Jede Assoziation besitzt mindestens zwei Assoziationsenden, die verschiedene Rollen innerhalb der Assoziation beschreiben. Jedes Assoziationsende besitzt einen Typ, der die Klasse angibt, die die Rolle ausfüllt. Dabei können mehrere Assoziationsenden denselben Typ besitzen. CMOF unterstützt lediglich die Modellierung binärer Assoziationen. Assoziationen werden durch die Klassen *Association* und *Property* repräsentiert. Die *Association*-Instanz modelliert dabei die Assoziation und die *Property*-Instanz die Assoziationsenden. Mit Hilfe der Modellierung von Assoziationsenden durch die *Property*-Klasse, kann für jedes Assoziationsende eine Multiplizität angegeben werden, die eine untere und obere Grenze für die Anzahl der Objekte an jedem Assoziationsende spezifiziert. Zusätzlich kann für jedes Ende angegeben werden, ob es ein navigierbares Assoziationsende darstellt oder nicht. Eine Assoziation kann eine einfache Beziehung zwischen Klassen beschreiben oder eine Kompositionsbeziehung, die einen Verbund kennzeichnet. Nur ein Assoziationsende einer Assoziation kann als Verbund - durch Setzen des Attributs *isComposite* - gekennzeichnet werden und stellt ein nicht-navigierbares Assoziationsende dar. Eine Kompositionsbeziehung

[5] Diagrammelemente, die für die weitere Arbeit nicht von Bedeutung sind, wurden aus Gründen der Übersichtlichkeit weggelassen.

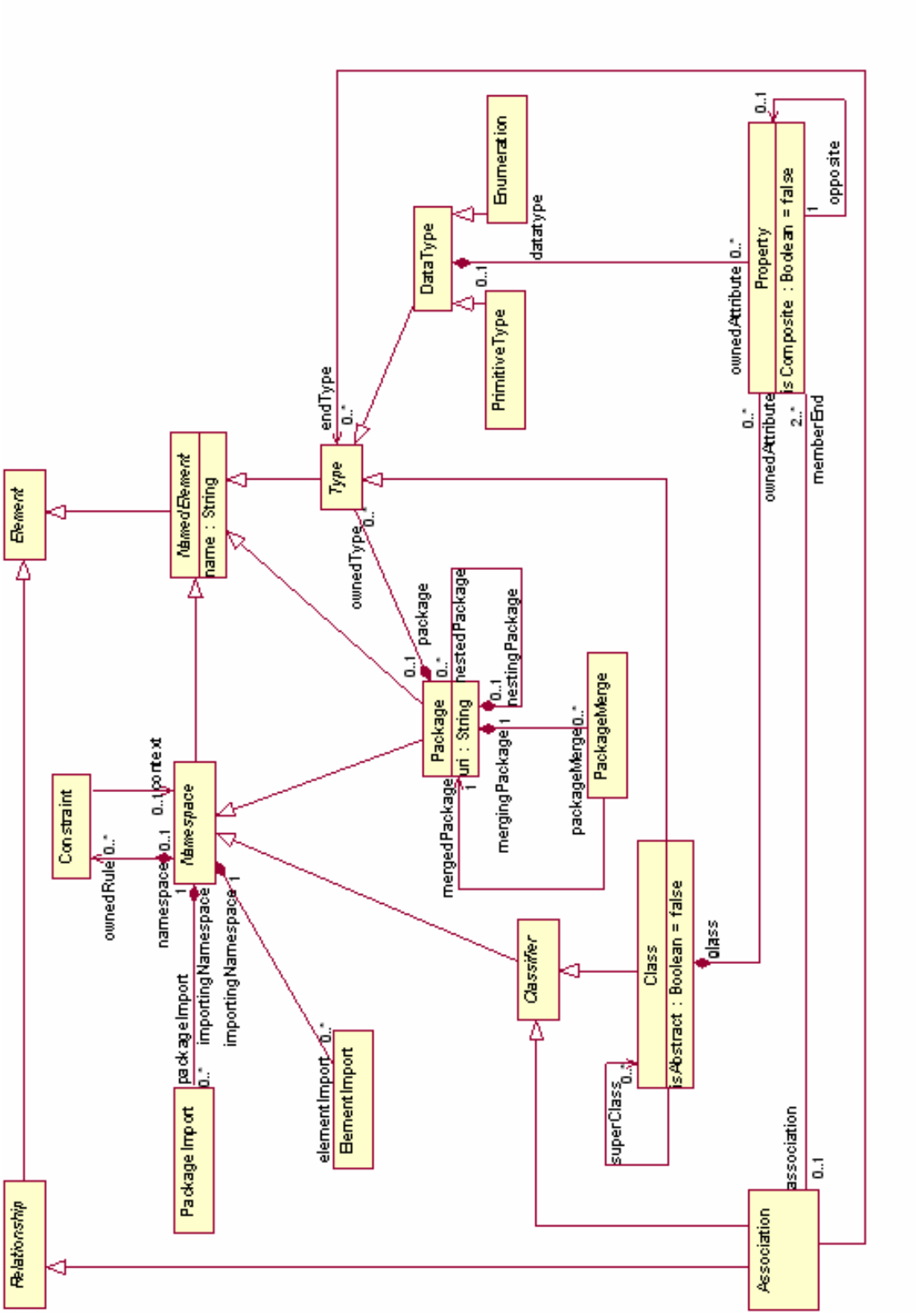


Abbildung 4.4: Klassendiagramm der CMOF nach [OMG04b, OMG04c]

koppelt den Lebenszyklus des enhaltenen Elements an den des Verbundes. Eine abgeschwächte Form der Kompositionsbeziehung stellt die Aggregation dar. Eine Aggregation ist eine Teil-von-Beziehung in der die Lebenszyklen der beteiligten Objekte unabhängig voneinander sind.

DataType (Datentyp): CMOF erweitert das Datentypen-Konzept aus EMOF. Durch die in CMOF konkrete Klasse DataType, ist eine Definition eigener Datentypen möglich. Zusätzlich können Datentypen in CMOF über Eigenschaften und Operationen verfügen.

Package Extension (Paket Erweiterung): CMOF erweitert das Paket-Konzept von EMOF. In CMOF definiert ein Paket einen Namensraum, der die im Paket enthaltenen Modellelemente umfasst. Zusätzlich zu der bereits in EMOF erklärten Schachtelung von Paketen definiert CMOF eine Importbeziehung zwischen Paketen und Modellelementen und eine Vereinigungsbeziehung (engl. PakageMerge) zwischen Paketen. Eine Importbeziehung fügt eine Referenz auf das zu importierende Modellelement oder Paket in den Namensraum des importierenden Pakets ein. Durch das Hinzufügen einer Referenz auf ein Element wird seine Benutzung ermöglicht und gleichzeitig eine Veränderung verhindert. Neben der Importbeziehung umfasst CMOF auch eine Vereinigungsbeziehung zwischen Paketen, in der definiert wird, wie ein ein Paket ein anderes durch Spezialisierung und Neudefinition erweitert. Das Konzept der Vereinigungsbeziehung sollte nur dann angewendet werden, wenn Elemente gleichen Namens in verschiedenen Paketen dasselbe Konzept beschreiben. Das vereinigende Paket nimmt die gleichnamigen Modellelemente der zu vereinigenden Pakete und verbindet sie durch Spezialisierung und Neudefinition zu einem Paket.

Constraints (Bedingungen): Bedingungen dienen der Beschreibung von Konsistenzbedingungen eines Modells. Durch die in EMOF vorhandenen Modellierungskonzepte kann ein statisches Modell beschrieben werden. CMOF erweitert dieses Modell um Konzepte zur Überprüfung von Konsistenzbedingungen. Bedingungen können in CMOF allen Instanzen vom Typ *Element* zugeordnet werden. CMOF definiert dabei keine feste Sprache mit der Bedingungen repräsentiert werden, empfiehlt aber die Verwendung der *Object Constraint Language* (OCL) [OMG05a] für die Spezifikation von Bedingungen in MOF- und UML-Modellen.

Modellierungskonzepte der UML

Die Unified Modeling Language dient nicht nur als Basis für die Definition der EMOF und CMOF, sondern definiert noch eine standardisierte, graphische Sprache zur Beschreibung von objektorientierten Modellen. Abbildung 4.5 stellt die Modellelemente eines Klassendiagramms, wie sie in dieser Arbeit verwendet werden, graphisch dar.

Die UML Superstructure definiert mehrere Diagramme mit verschiedenen graphischen Elementen für die unterschiedlichen Aspekte der Sofwareentwicklung. In dieser Arbeit werden nur die Modellierungselemente der Strukturdiagramme betrachtet, die für die

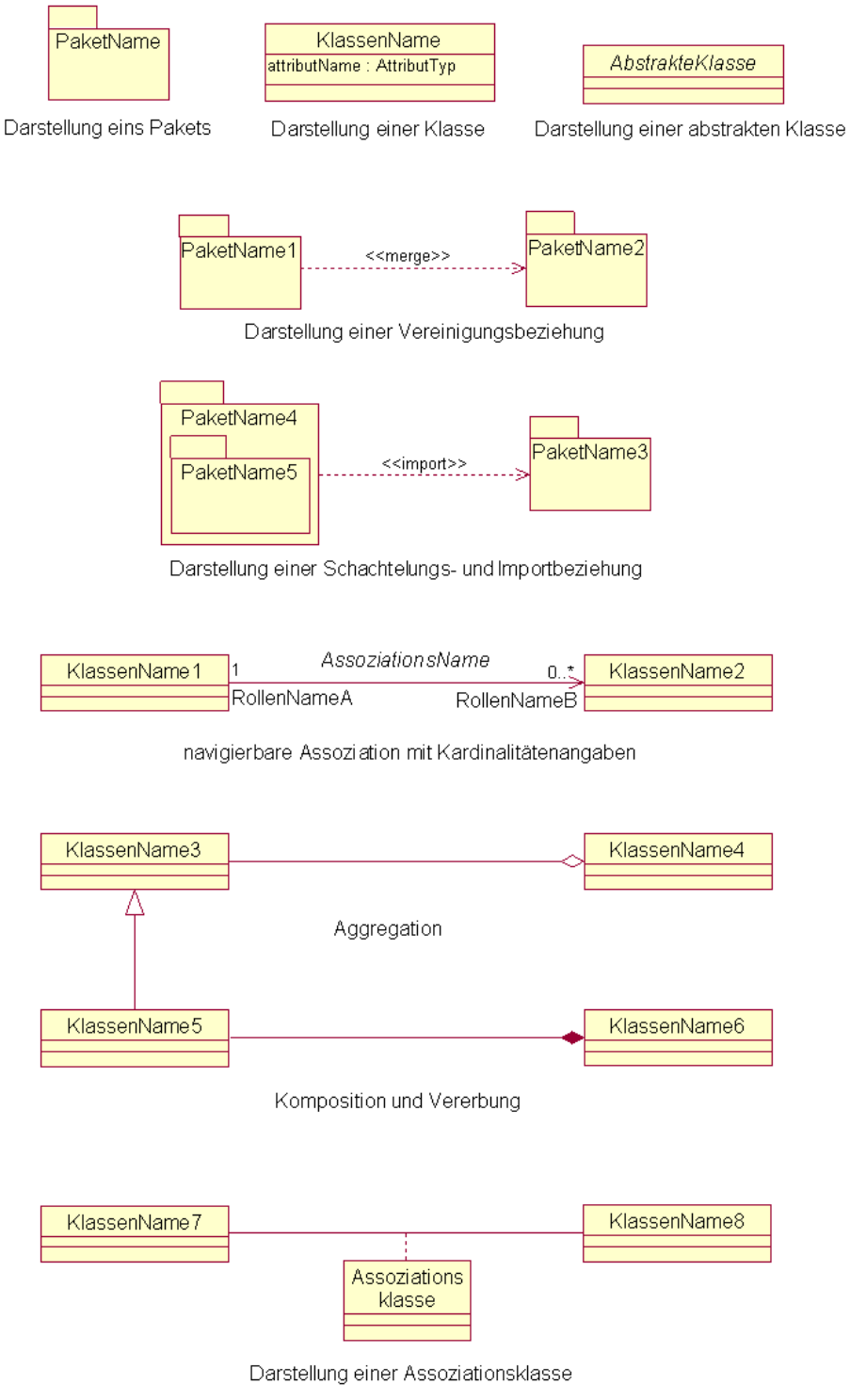


Abbildung 4.5: Modellelemente eines Klassendiagramms

Modellierung eines Informationsmodelles für das Enterprise Architecture Management eine Rolle spielen. Die UML erweitert die Konzepte der EMOF und CMOF noch um weitere Konzepte [OMG04c, OMG04d, Oes04, RHQ05, Dum05]:

Association class (Assoziationsklasse): Eine Assoziationsklasse ist ein Modellierungselement, das sowohl Klassen- als auch Assoziationseigenschaften besitzt. Eine Assoziationsklasse kann sowohl als Klasse mit Assoziationseigenschaften gesehen werden, als auch als Assoziation mit Klasseneigenschaften. Assoziationsklassen werden verwendet um komplexe Beziehungen zwischen Klassen, die eigene Attribute und Operationen besizten können, zu modellieren. Eine Assoziationsklasse kann alternativ als eigenständige Klasse mit zwei Assoziationen modelliert werden. Abbildung 4.6 zeigt ein Beispiel für eine Assoziationsklasse und ihre alternative Modellierung als eigenständige Klasse.

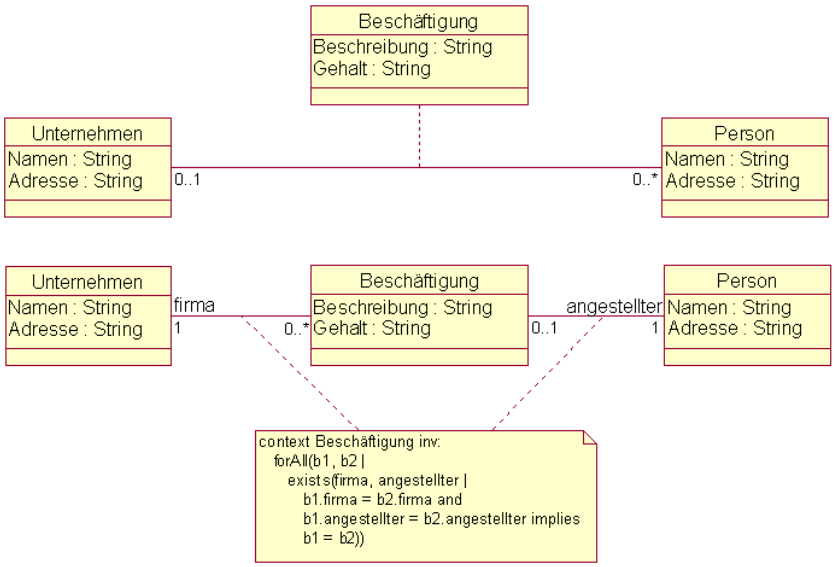


Abbildung 4.6: Assoziationsklassen und ihre alternative Modellierung

Stereotype (Stereotyp)[6]**:** Stereotypen sind Erweiterungsmechanismen, die es erlauben, Modellelemente in UML zu klassifizieren. Sie erweitern die vorhandenen Modellelemente des UML-Metamodells um neue Arten von Bausteinen zu kreieren, die in der jeweiligen Domäne benötigt werden. Durch Stereotypen lassen sich die möglichen Verwendungen eines Modellelements unterscheiden, dabei werden mehreren Klassen bestimmte Eigenschaften zugeschrieben. Ein Modellierungselement kann mit beliebig vielen Stereotypen klassifiziert werden. Stereotypen besitzen keine Typsemantik, sie ermöglichen die Spezialisierung sämtlicher Metaklassen des UML-Metamodells (mit Ausnahme der Metaklasse Stereotype selbst) hinsichtlich Funktionalität und Notation.

[6] In der aktuellen Version von MOF 2.0 [OMG04b] ist das *Profile Package*, welches Stereotypen beinhaltet, nicht Teil von MOF

Exemplarisch sei der Stereotyp Interface[7] genannt, der in Klassendiagrammen häufig Verwendung findet.

Trigger/Event (Auslöser/Ereignis): Trigger und Events sind Konzepte die ein dynamisches Verhalten beschreiben und in der UML Superstructure im *Common Behaviors*-Paket definiert sind. Events können durch ihr Auftreten die Ausführung von assoziiertem Verhalten verursachen. Ein Trigger spezifiziert den Event, der die Ausführung auslösen kann, sowie mögliche Bedingungen die zur Filterung eingesetzt werden können.

In den vorangegangenen Abschnitten wurden die Metamodellierungselemente der Sprachen *Essential MOF* (EMOF), *Complete MOF* (CMOF) und *UML* vorgestellt. Gemeinsame Basis dieser Sprachen bildet die *UML 2.0 Infrastructure* [OMG04c]. EMOF definiert dabei die elementarste Menge an Modellierungskonzepten zur Konzeption einfacher Modelle und baut auf dem Paket *Basics* der UML Infrastructure auf. Es definiert eine Modellierungssprache, die eng an die Eigenschaften von Programmiersprachen angelehnt ist und eine unkomplizierte Implementierung ermöglicht. CMOF erweitert die Metamodellierungskonzepte von EMOF unter Verwendung des Pakets *Constructs* der UML Infrastruktur, um komplexe Metamodelle konzipieren und anschaulich darstellen zu können. Eine der wesentlichen Erweiterungen der CMOF gegenüber der EMOF stellt das Konzept der Bedingungen und der Assoziationen dar, letzteres ist in EMOF nicht explizit vorhanden und kann nur nachgebildet werden.

Die UML teilt sich in die *UML 2.0 Infrastructure* [OMG04c], die die Basis für EMOF und CMOF bildet und in die *UML 2.0 Superstructure* [OMG04d], die verschiedene Diagrammtypen für die Modellierung vorstellt und weitere Modellelemente definiert. Eine Möglichkeit zur domänenspezifischen Erweiterung der Modellelemente stellt die UML mit der Definition von Stereotypen bereit. Eine tabellarische Übersicht der unterstützten Modellierungskonzepte von EMOF, CMOF und UML bietet Tabelle 4.1.

	EMOF	**CMOF**	**UML**
Klassen	√	√	√
Abstrakte Klassen	√	√	√
Attribute	√	√	√
vordefinierte Datentypen	√	√	√
erweiterbare Datentypen		√	√
Pakete	√	√	√
Paketschachtelung	√	√	√

[7] Ein Interface ist eine Menge von Operationen, die das Verhalten von Klassen klassifizieren [OMG04c].

	EMOF	CMOF	UML
Paketimport		√	√
Paketvereinigung		√	√
Binäre Assoziationen	√[8]	√	√
n-äre Assoziationen			√
Assoziationsnamen		√	√
Rollennamen	√[8]	√	√
Multiplizitäten	√	√	√
Navigierbarkeit		√	√
Vererbung	√	√	√
Aggregation	√[8]	√	√
Komposition	√[8]	√	√
Assoziationsklassen			√
Bedingungen		√	√
Stereotypen			√
Trigger/Event			√

Tabelle 4.1: Tabellarischer Überblick der von Modellierungssprachen unterstützen Konzepte

4.2 Analyse von Informationsmodellen in Unternehmen

Anhand der im vorangegangenen Kapitel definierten Metamodellierungskonzepte sollen in diesem Kapitel vier Informationsmodelle von mittelständischen und großen Unternehmen auf die in ihnen verwendeten Konzepte analysiert werden. Ziel der Analyse ist die Bestimmung einer geeigneten Metamodellierungssprache als Basis für die Modelltransformationen von Informationsmodellen unterschiedlicher Unternehmen in ein geeignetes Informationsmodell für das Enterprise Architecture Management.

Im Rahmen des Forschungsprojektes Softwarekartographie wurden in Zusammenarbeit mit der BMW Group, HVB Systems, Siemens und der Krones AG IT-Landschaften

[8] Konzept kann durch "verschränkte" Attribute nachgebildet werden

mittelständischer und großer Unternehmen analysiert [Bre05, Hal04, Lau04]. Alle entwickelten Informationsmodelle wurden objektorientiert beschrieben und drei mit Hilfe des Modellierungswerkzeuges Rational Rose dokumentiert. Ein Informationsmodell wurde mit Hilfe einer selbstdefinierten Notation unter Verwendung objektorientierter Konzepte beschrieben. Die folgenden Abschnitte fassen die Analyseergebnisse der Metamodellierungskonzepte der Informationsmodelle zusammen.

Klassen, Attribute, Datentypen und Operationen

Klassen und Attribute stellen ein elementares Modellierungselement dar. Jedes der analysierten Informationsmodelle besteht aus einer Vielzahl von Klassen, die Attribute enthalten. In allen Modellen kommen primitive Datentypen, sowie Klassen als Attributtypen vor. Aufgrund der speziellen Verwendung von Informationsmodellen zur Modellierung struktureller Informationen über Anwendungslandschaften wird in keinem der analysierten Modelle das Modellierungskonzept der Operation verwendet. Abbildung 4.7 zeigt einen Ausschnitt aus einem exemplarischen Informationsmodell, der die Verwendung der Konzepte Klasse, Attribut und Datentyp visualisiert.

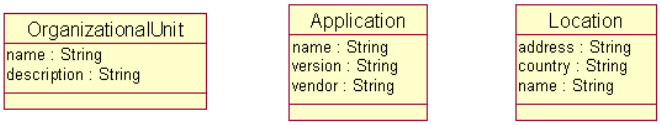


Abbildung 4.7: Exemplarische Verwendung der Konzepte Klasse, Attribut, Datentyp und Operation

Pakete, Paketschachtelung, Paketimport und Paketvereinigung

Drei der vier analysierten Informationsmodelle verwenden Pakete zur Gruppierung von Modellelementen. Da Informationsmodelle komplexe Systeme beschreiben, bietet sich die Verwendung von Paketen zur Komplexitätsreduzierung und Erhöhung der Übersichtlichkeit an. Die Aufteilung der Modellelemente auf Pakete führt gleichzeitig zu einer Verwendung des Import-Konzeptes in drei der vier analysierten Modelle, um eine Integration der in den unterschiedlichen Paketen enthaltenen Modellelemente zu ermöglichen. Zusätzlich verwenden zwei von vier analysierten Modellen das Konzept der Paketschachtelung. Das Konzept der Paketvereinigung findet hingegen bei keinem der analysierten Informationsmodelle Verwendung. Abbildung 4.8 visualisiert die Verwendung von Paketen bei der Modellierung von Informationsmodellen.

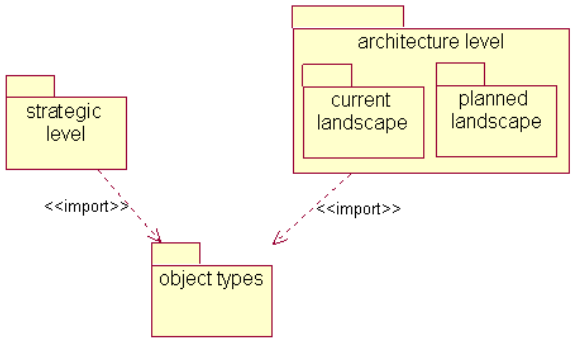


Abbildung 4.8: Exemplarische Verwendung von Paketen

Vererbung und Abstrakte Klassen

Das Konzept der Vererbung findet in allen vier analysierten Informationsmodellen Verwendung, zumeist in engem Zusammenhang mit dem Konzept der abstrakten Klasse, das ebenfalls in allen analysierten Informationmodellen verwendet wird. Dabei dient die Superklasse ausschließlich dem Zweck der Modellierung gemeinsamer Attribute und kann nicht selbst instanziiert werden. So können obligatorische Attribute, die für alle Informationsobjekte gepflegt werden sollen in einer abstrakten Klasse *AllObjects* definiert werden, zu der alle anderen Klassen in einer Vererbungsbeziehung stehen. Diese Verwendung der Konzepte Vererbung und abstrakte Klasse visualisiert Abbildung 4.9.

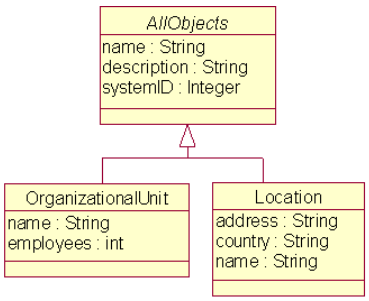


Abbildung 4.9: Exemplarische Verwendung der Konzepte Vererbung und abstrakte Klasse

Assoziation, Aggregation, Komposition, Multiplizitäten, Navigierbarkeit und Assoziationsklassen

Die Assoziation gehört zu den elementaren Metamodellierungskonzepten und findet sich in allen analysierten Informationsmodellen wieder, dabei beschränkt sich die Verwendung des Assoziationskonzeptes auf binäre Assoziationen. Folgende weiterführende Assoziationskonzepte fanden in den analysierten Informationsmodellen Verwendung:

- **Assoziationsnamen:** Alle Informationsmodelle verwenden Assoziationsnamen zur genaueren Beschreibung von Beziehungen.
- **Rollennnamen:** Alternativ werden Beziehungen in drei von vier analysierten Informationsmodellen durch Rollennamen näher spezifiziert.
- **Navigierbarkeit:** Das Konzept der Navigierbarkeit wird nur in zwei der analysierten Informationsmodelle als Modellierungskonzept eingesetzt.
- **Aggregation:** Drei von vier Informationsmodellen verwenden Aggregationen zur Beschreibung von Teile-von-Beziehungen.
- **Komposition:** Eine explizite Unterscheidung zwischen Aggregation und Komposition und die Verwendung beider Konzepte wird nur in einem der analysierten Informationsmodelle getroffen.
- **Multiplizitäten:** Die Verwendung von Multiplizitäten, um die Anzahl von Verbindungen zu einem Objekt festzulegen, wird in drei der analysierten Informationsmodelle unterstützt. Lediglich ein Informationsmodell verzichtet auf die Angabe von Multiplizitäten[9].
- **Assoziationsklassen:** Assoziationsklassen stellen in Informationsmodellen ein beliebtes Mittel dar, um Assoziationen genauer zu spezifizieren. Zwei der vier analysierten Informationsmodelle verwenden Assoziationsklassen. Im Kontext der Informationsmodelle werden dabei Attribute und keine Operationen an die Assoziation gehängt.

Abbildung 4.10 visualisiert exemplarische Verwendungen des Konzepts der Assoziation im Kontext der Informationsmodelle.

Stereotypen und Bedingungen

Stereotypen und Bedingungen ermöglichen eine genauere Spezifikation von Informationsobjekten. Eine mögliche Verwendung von Stereotypen bildet die Modellierung

[9] Diese Restriktion ist auf das verwendete Werkzeug zur Pflege der Informationsobjekte zurückzuführen, das keine Angabe von Multiplizitäten unterstützt (vgl. Kapitel 4.3)

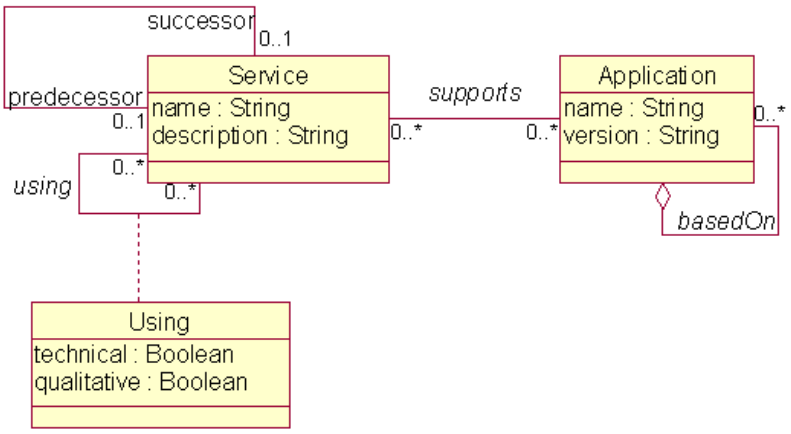


Abbildung 4.10: Exemplarische Verwendung des Assoziationskonzepts

von Auswahllisten mit fest vorgegebenen Werten, wie sie Abbildung 4.11 exemplarisch darstellt. Trotz dieser Möglichkeiten verwendet keines der analysierten Informationsmodelle Stereotypen. Bedingungen könnten in Informationsmodellen genutzt werden, um Abhängigkeiten zwischen mehreren Informationsobjekten genauer zu spezifizieren. Das Konzept der Bedingungen wurde in einem der analysierten Informationsmodelle verwendet. Abbildung 4.11 visualisiert exemplarisch eine mögliche Verwendung einer Bedingung mit Hilfe der *OCL*.

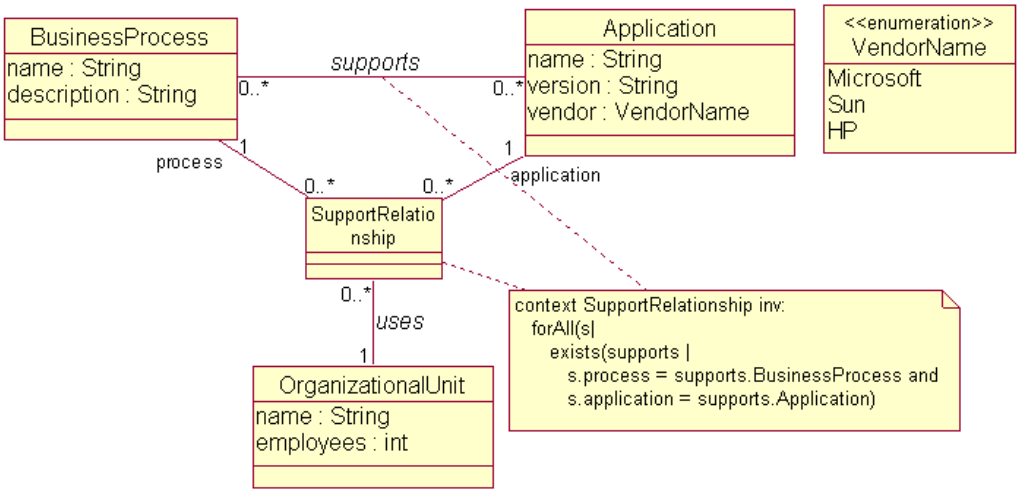


Abbildung 4.11: Exemplarische Verwendung der Konzepte Stereotyp und Bedingung

In den Arbeiten [Bre05, Hal04, Lau04] finden sich Beschreibungen, der analysierten Informationsmodelle mittelständischer und großer Unternehmen, auf denen die oben genannten Analyseergebnisse beruhen.

Eine tabellarische Übersicht der oben aufgelisteten Ergebnisse und eine Gegenüberstellung mit den in EMOF, CMOF und UML vorhandenen Metamodellierungskonzepten findet sich in Tabelle 4.2.

	EMOF	**CMOF**	**UML**	U_1	U_2	U_3	U_4
Klassen	√	√	√	√	√	√	√
Abstrakte Klassen	√	√	√	√	√	√	√
Attribute	√	√	√	√	√	√	√
vordefinierte Datentypen	√	√	√	√	√	√	√
erweiterbare Datentypen		√	√	√	√	√	√
Pakete	√	√	√	√	√	√	
Paketschachtelung	√	√	√		√	√	
Paketimport		√	√	√	√	√	
Paketvereinigung		√	√				
Binäre Assoziationen	√[10]	√	√	√	√	√	√
n-äre Assoziationen			√				
Assoziationsnamen		√	√	√	√	√	√
Rollennamen	√[10]	√	√	√	√	√	
Multiplizitäten	√	√	√	√	√	√	
Navigierbarkeit		√	√	√	√		
Vererbung	√	√	√	√	√	√	√
Aggregation	√[10]	√	√	√	√	√	
Komposition	√[10]	√	√			√	
Assoziationsklassen			√	√	√		

[10] Konzept kann durch "verschränkte" Attribute nachgebildet werden

	EMOF	CMOF	UML	U_1	U_2	U_3	U_4
Bedingungen		√	√		√		
Stereotypen			√				
Trigger/Event			√				

Tabelle 4.2: Tabellarischer Überlick der in Informationsmodellen von Unternehmen verwendeten Modellierungskonzepte

4.3 Analyse von Informationsmodellen von Werkzeugen

Anhand des im vorangegangenen Abschnitt definierten Bedarfs an Metamodellierungskonzepte für die Modellierung von Informationsmodellen sollen im folgenden Abschnitt Werkzeuge für das Enterprise Architecture Management untersucht werden. Die exemplarisch untersuchten Werkzeuge sind der *Corporate Modeler* von *Casewise*, das Tool *MEGA* der *MEGA International* und der *System Architect* der Firma *Telelogic*.

Casewise: Corporate Modeler

Die Firma *Casewise* bietet mit dem *Corporate Modeler* ein Werkzeug für das Enterprise Architecture Management an. Die Modellierung des Informationsmodells des Corporate Modelers erfolgt durch die Erzeugung von Entitäten und Beziehungen zwischen diesen Entitäten. Die Entitäten des Corporate Modeler entsprechen dem Klassenkonzept der UML und MOF. Entitäten besitzen typisierte Attribute. Attributtypen können dabei sowohl primitive Datentypen (Integer, String, etc.) sein als auch vordefinierte Objekttypen (Date, URL, etc.). Neben den Entitäten spielen die Beziehungen die zentrale Rolle im Metamodell des Corporate Modeler. Es werden binäre Beziehungen unterstützt, die in einem eigenen Assoziationsobjekt gespeichert werden. Dieses Assoziationsobjekt enthält den Namen der Assoziation, sowie die Namen der Assoziationsenden (Rollennamen). Durch die Speicherung einer Assoziation in einem eigenen Objekt, können diesem Objekt eigene Attribute hinzugefügt werden. Dieses Beziehungsobjekt ist jedoch kein *First-Level*-Objekt[11], sondern entspricht lediglich einer attributierten Assoziation. Das Konzept der abstrakten Klasse wird im Corporate Modeler durch die Markierung eines Objektes als *Template* umgesetzt. Weiterführende Konzepte der MOF

[11] Die Attribute der Assoziation werden zwar in einem eigenen Objekt gespeichert, doch können andere Objekte keinen Link zu dem Beziehungsobjekt haben.

und UML, wie Aggregation, Komposition oder Multiplizitäten werden vom Corporate Modeler nicht unterstützt. Abbildung 4.12 visualisiert ein Repository des Corporate Modeler mit den definierten Entitäten und Beziehungen [seb05a].

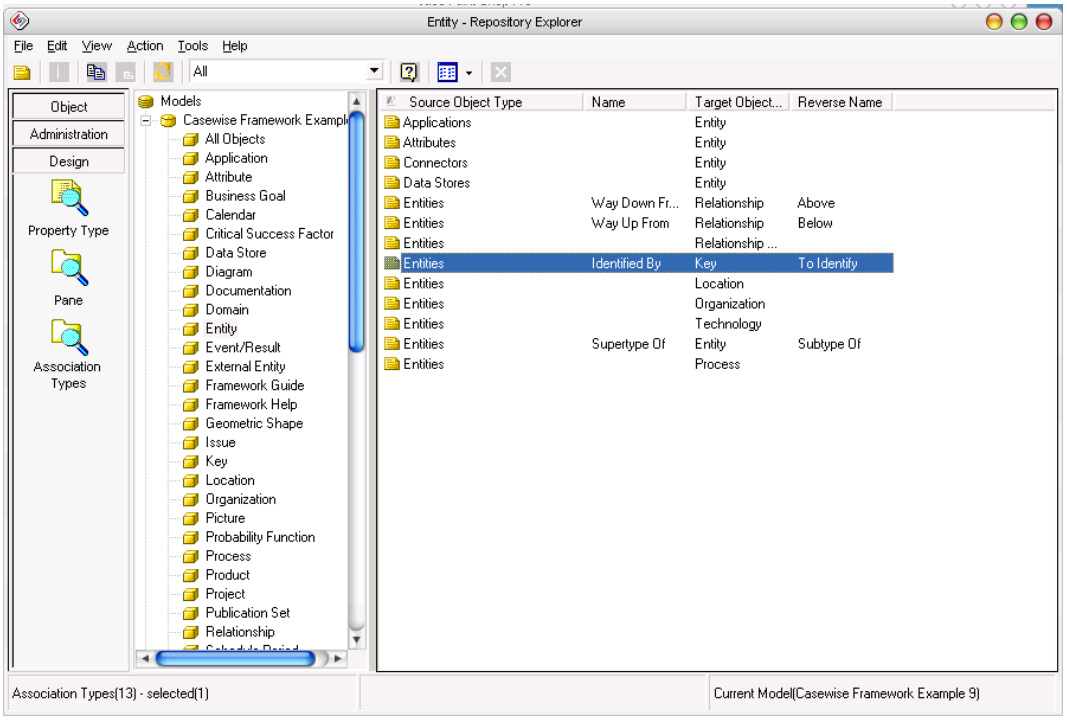


Abbildung 4.12: Entity-Diagramm des Corporate Modeler

MEGA International MEGA

MEGA International stellt mit *MEGA* ein Enterprise Architecture Management Werkzeug bereit, das laut Hersteller Angaben auf einem MOF kompatiblen Repository aufbaut. Die Metamodellierung in MEGA erfolgt über Objekte. Alles innerhalb des Metamodells wird als Objekt modelliert, sogar Attribute, Beziehungen und Diagramme. Dabei entspricht ein Metaobjekt in MEGA einem Entitätstyp, ein Metaattribut einem Attributstyp und eine Metaassoziation einem Assoziationstyp. Die Zuordnung eines Attributs zu einer Entität erfolgt in MEGA über eine Verknüpfung des Attributs mit dem entsprechenden Objekt. Aufgrund dieser Modellierung wird eine Wiederverwendung von Attributen ermöglicht.

Eine Beziehung zwischen Entitäten besteht aus drei Objekten: zwei Metaassoziationsenden und einer Metaassoziation. Diese Modellierung entspricht dem binären Assoziationsprinzip der UML und MOF. In MEGA können sowohl die Assoziationen als auch die Assoziationsenden benannt werden. Ebenso wird die Angabe von Multiplizi-

täten und Assoziationstypen (Aggregation, Komposition) unterstützt. Das Konzept der Vererbung kann in MEGA durch das *Sub-typing* nachgebildet werden. Eine vollwertige Unterstützung des Paketkonzept ist in MEGA nicht enthalten, lediglich die Definition von Namensräumen wird unterstützt. Weiterführende Konzepte, wie Assoziationsklassen, Stereotypen oder die Definition von Bedingungen werden von MEGA nicht unterstützt. Abbildung 4.13 zeigt ein mit MEGA erstelltes Klassendiagramm [seb05a].

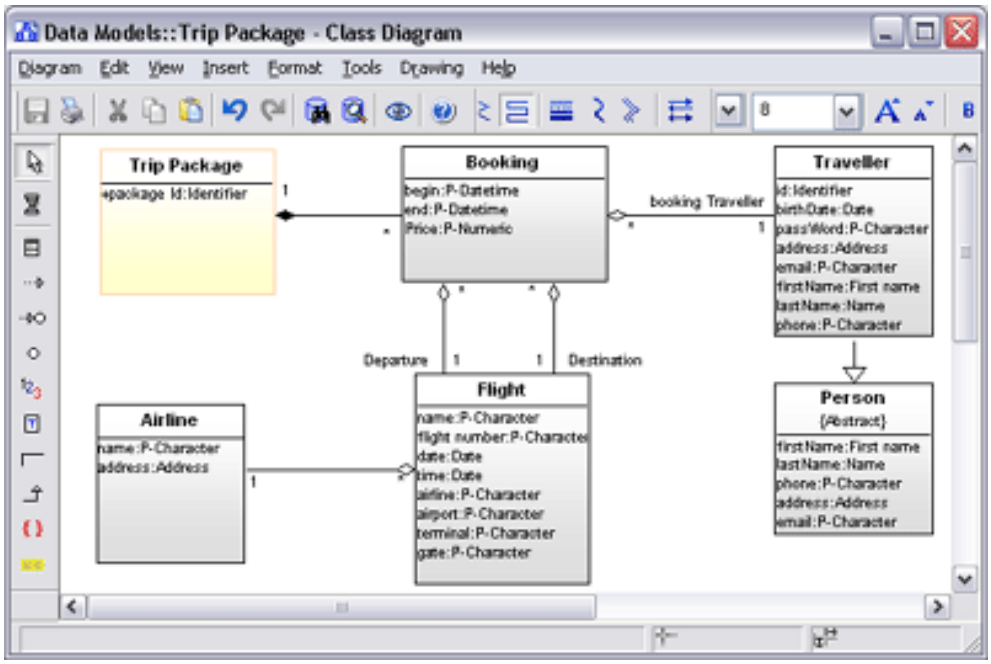


Abbildung 4.13: Klassendiagramm in MEGA [MEGA05]

Telelogic System Architect

Das Werkzeug *System Architect* der Firma *Telelogic* basiert auf einem vordefinierten Metamodell, das angepasst und erweitert werden kann. Die Anpassung erfolgt über eine SQL-ähnliche Datendefinitionssprache. Bei der Adaption können neue Entitäten, Attribute, Beziehungen, Symbole und Diagramme hinzugefügt und vordefinierte Elemente angepasst oder versteckt werden.

Die Entitäten des System Architect entsprechen dem Klassenkonzept der MOF und UML und können Attribute beinhalten. Die unterstützten Datentypen des System Architect umfassen primitive und vordefinierte Datentypen. Um Beziehungen genauer spezifizieren zu können unterstützt der System Architect Assoziationsnamen, Rollennamen, Multiplizitäten und das Konzept der Vererbung. Zusätzlich können im System Architect Attribute an Beziehungen angehängt und somit ein Teil des

Assoziationsklassen-Konzept modelliert werden. Die Abbidlung 4.14 visualisiert ein Beispieldiagramm im System Architect [seb05a].

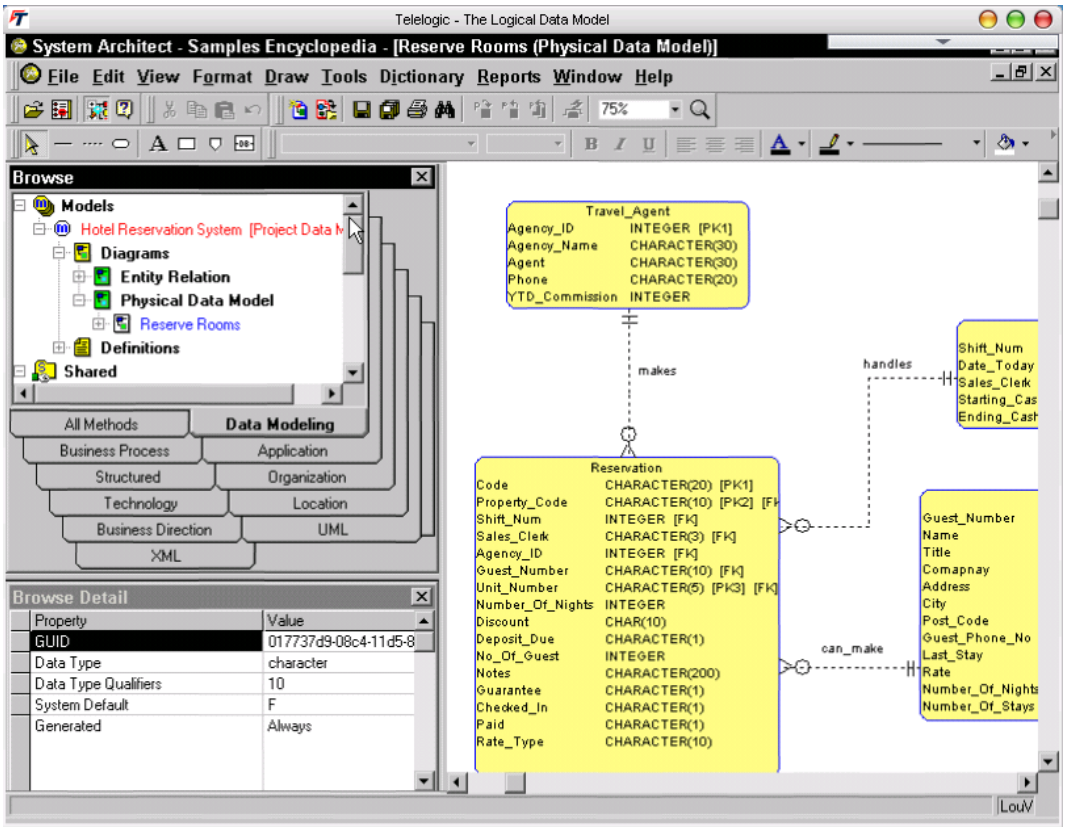


Abbildung 4.14: Screenshot des Tools System Architect von Telelogic

Die Analyse der unterschiedlichen Werkzeuge für das Enterprise Architecture Management zeigt eine große Übereinstimmung in der Unterstützung der Kernmodellierungskozepte, wie Klassen, Attribute, Assoziationen.

Bei der Unterstützung weiterführender Konzepte zeigen sich die unterschiedlichen Zielsetzungen der Werkzeuge. Während der Corporate Modeler nur die elementaren Metamodellierungskonzepte unterstützt und so einen simplen Modellierungsansatz bietet, verwendet *MEGA* viele weiterführende Konzepte, wie Vererbung, Aggregation und Komposition, um die Konzeption komplexer Modelle zu ermöglichen.

Einen tabellarischen Überlick über die, von dem Corporate Modeler (CM), MEGA und dem System Architect (SA) unterstützten Metamodellierungskonzepte und eine Gegenüberstellung zu EMOF, CMOF und UML enthält Tabelle 4.3.

	EMOF	CMOF	UML	CM	MEGA	SA
Klassen	√	√	√	√	√	√
Abstrakte Klassen	√	√	√	√[12]		
Attribute	√	√	√	√	√	√
vordefinierte Datentypen	√	√	√	√	√	√
erweiterbare Datentypen		√	√			
Pakete	√	√	√		√[13]	
Paketschachtelung	√	√	√			
Paketimport		√	√			
Paketvereinigung		√	√			
Binäre Assoziationen	√[14]	√	√	√	√	√
n-äre Assoziationen			√			
Assoziationsnamen		√	√	√	√	√
Rollennamen	√[14]	√	√	√	√	√
Multiplizitäten	√	√	√		√	√
Navigierbarkeit		√	√			
Vererbung	√	√	√		√	
Aggregation	√[14]	√	√		√	
Komposition	√[14]	√	√		√	
Assoziationsklassen			√	√[15]		√[15]
Bedingungen		√	√			
Stereotypen			√			
Trigger/Event			√			

Tabelle 4.3: Tabellarischer Überlick der von Werkzeugherstellern unterstützten Modellierungskonzepte

[12] Klassifizierung eines Objekt Typs als *Template* ist möglich.
[13] Die Definition von Namensräumen wird unterstützt.
[14] Konzept kann durch "verschränkte" Attribute nachgebildet werden.
[15] Es werden nur attributierte Assoziationen und kein vollständiges Assoziationsklassenkonzept unterstützt.

4.4 Analyse standardisierter Informationsmodelle

Nach der Analyse existierender Informationsmodelle von Unternehmen und Werkzeugen zum Enterprise Architecture Management in den vorangegangenen Kapiteln 4.2 und 4.3 sollen in den nächsten Abschnitten zwei standardisierte Informationsmodelle, das *Common Information Model* (CIM) der DMTF und die *IT Management Portfolio Management Facility* (ITPMF) der OMG, auf die von ihnen verwendeten Metamodellierungskonzepte untersucht werden.

Derzeit existiert kein standardisiertes Informationsmodell für das Enterprise Architecture Management, so dass in dieser Arbeit zwei existierende Modelle aus dem Bereich IT-Infrastruktur- und IT-Portfolio-Management herangezogen werden.

4.4.1 Common Information Model (CIM)

Das *Common Information Model* (CIM) ist ein von der *Distributed Management Task Force* (DMTF) entwickelter Standard zum Management von IT-Systemen. Sein Ziel ist die Bereitstellung einer einheitlichen, implementierungs- und plattformunabhängigen Managementschnittstelle verteilter Informationssysteme. Daneben stellt die Integration älterer Standards, wie *SNMP* [16] und *DMI*[17] ein weiteres Ziel der CIM dar, die eine verlustfreie Abbildung auf CIM ermöglichen soll.

Für die Strukturierung der zu verwaltenden Informationssysteme verwendet das CIM einen objektorientierten Ansatz. Es stellt ein konzeptionelles Framework zur Organisation der zu verwaltenden Information bereit und bietet ein mächtiges Datenmodell, das in drei Ebenen unterteilt ist:

- **Core Model:** Das Core Model besteht aus einigen wenigen Klassen, Beziehungen und Attributen, die ein Grundvokabular für die Analyse und Beschreibung der zu verwaltenden Systeme bilden. Das Core Model bildet die Basis für die Verfeinerungen und Erweiterungen im *Common Model*.
- **Common Model:** Das Common Model besteht aus einer Menge von Klassen, die jeweils für einen bestimmten Bereich des Management zugeschnitten sind und bereits als Basis für eine Reihe von Managementanwendungen dienen können. Diese Anwendungen umfassen Bereiche wie Systeme, Applikationen, Netzwerke, Endgeräte und Benutzerverwaltung, wobei mit fortschreitender Entwicklung des Modells neue Anwendungsfelder hinzukommen können. Die im Common Model beschriebenen Bereiche sind unabhängig von technischen Implementierungen.

[16] Simple Network Management Protocol [Cas89]
[17] Desktop Management Interface [DMT03b]

- **Extension Schema:** Die Extension Schemas sind technologiespezifische Erweiterungen des Common Models und werden nicht durch die DMTF festgeschrieben sondern durch Herstellerfirmen, die das Modell um implementierungs- und architekturspezifische Verfeinerungen ergänzen.

Das Core und Common Model gemeinsam werden als *CIM Schema* bezeichnet und von der DMTF ständig weiterentwickelt und in den verschiedenen Entwicklungsstufen auf der Webseite der DMTF[18] veröffentlicht.

Neben den drei Ebenen liefert CIM mit dem *Meta Schema* eine formale Definition des Modells. Es legt die in CIM verwendeten Begriffe, ihre Verwendung und ihre Semantik fest. Die Struktur des Meta Schema wird mit Hilfe der *Unified Modeling Language* festgelegt. Abbildung 4.15 visualisiert die Struktur des Meta Schemas.

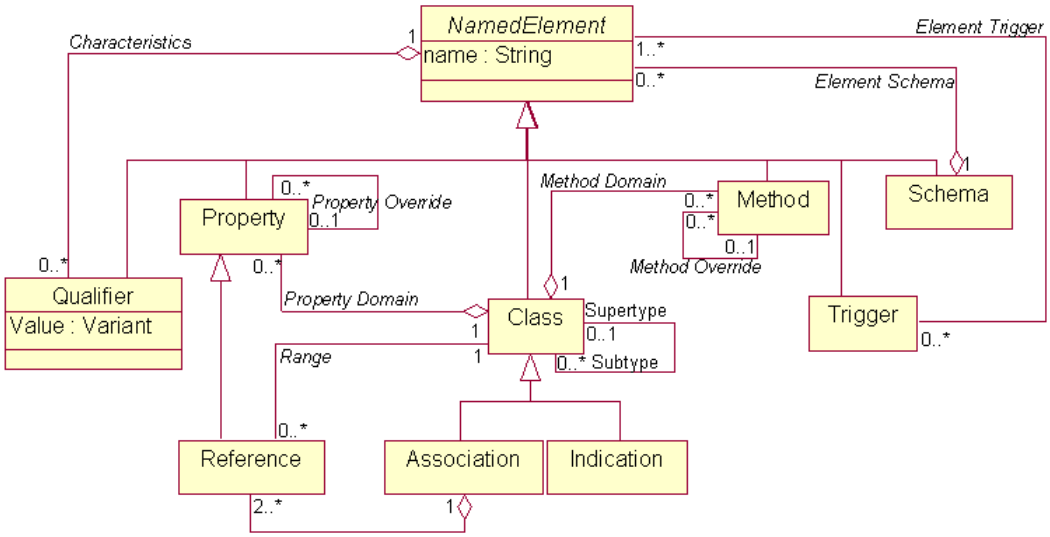


Abbildung 4.15: Klassendiagramm des CIM Meta Schemas

Das CIM Schema ist eine Untermenge der UML mit wenigen Erweiterungen. Das Kernkonzept der CIM bildet wie bei der UML das Klassen und Assoziationskonzept. Klassen können im CIM genau wie in der UML Attribute und Methoden beinhalten. Datentypen werden im CIM durch die *Qualifier* Klasse modelliert. Ein Qualifier kann eine Klasse auch als abstrakt kennzeichnen. Ein *Schema* im CIM fasst eine Menge von Klassen zusammen und bildet einen Namensraum, der dem Paketkonzept der UML entspricht. Die *Association* Klasse modelliert eine UML Assoziation, die mindestens zwei

[18] http://www.dmtf.org

References auf die Assoziationenden enthält. Durch die Association und Reference Klassen können Assoziations- und Rollennamen modelliert werden. Zusätzlich unterstütz das CIM die Definition von Multiplizitäten durch die Reference Klassen. Auch die Klassifizierungen von Assoziationen, wie Aggregation, Komposition, Vererbung und attributierte Assoziationen (Assoziationsklassen) werden von CIM unterstützt.

Das CIM erweitert die von UML verwendeten Konzepte noch um die Konzepte des *Triggers* und der *Indication*. Beide Klassen werden verwendet um eine Ereignisbehandlung zu ermöglichen. Der Trigger löst dabei die Ereignisbehandlung aus und die Indication repräsentiert einen Typ von Ereignismeldungen. Die Verwendung von Bedingungen ist in der aktuellen CIM Version nicht enthalten, soll aber in zukünftigen Versionen unterstützt werden [DMT05, DMT03a].

4.4.2 IT Portfolio Management Facility (ITPMF)

Die *IT Portfolio Management Facility* (ITPMF) [OMG04a] ist eine Spezifikation der *Object Management Group* (OMG), die eine geeignete Methodik bzw. ein geeignetes Metamodell zur Abbildung und Steuerung der Informationssysteme eines Unternehmens definiert. Ihr Fokus liegt dabei auf dem Umgang mit den installierten Software- und Plattformelementen eines Unternehmens.

Die ITPMF definiert einen strukturierten Managementansatz, zur effektiven und effizienten Ausrichtung der Informationssysteme an den Unternehmenszielen, den Anforderungen der Geschäftsprozesse und den gesetzlichen Anforderungen. Zu diesem Zweck definiert die ITPMF eine Menge von UML-Diagrammen, die jeweils Teilaspekte des Unternehmens modellieren:

Das *Control Diagram* definiert die zu verwaltenden Elemente, ihre Lebenszyklen und die Möglichkeiten des Erweiterungsmechanismus. Das *Context Diagram* beschreibt die Vereinbarungen (engl. Agreement), die zwischen Parteien in Bezug auf ein Element geschlossen werden können. Handelt es sich bei einem Element um ein "deployable"[19] Element, so spezifiziert das *Deployable Element Diagramm* die möglichen Arten des "Deployments", während das *Deployment Diagram* die Umsetzung des "Deployments" auf die Hardware an einem bestimmten Standort behandelt.

Die ITPMF spezifiziert zu den oben genannten Diagrammen noch weitere Diagramme: Neben dem *Kinds Diagram* und dem *Inheritance Diagram*, die Strukturaspekte des Informationsmodells umfassen, beschreibt das ITPMF das *Essential Diagram*. Das Essential Diagram enthält die Schlüsselklassen der einzelnen Diagramme, um den Zusammenhang zwischen den einzelnen Diagrammmen zu erläutern und einen Überblick zu ermöglichen.

[19] dt. einsetzbar, installierbar

Für die Konzeption der Modelle verwendet die ITPMF folgende Metamodellierungskonzepte:

- (abstrakte) Klassen
- Attribute
- primitive Datentypen
- Pakete, Paketschachtelung, Paketimport
- binäre Assoziationen, Assoziationsnamen, Rollennamen, Multiplizitäten
- Navigierbarkeit
- Aggregation, Komposition, Vererbung
- Bedingungen
- Stereotypen («enumeration»)

Im Gegensatz zu dem CIM-Modell, das die für die Modellierung von Informationmodellen benötigten Metamodellierungskonzepte beschreibt, definiert der ITPMF-Standard verschiedene Diagramme und die in ihnen enthaltenen Elemente.

In Tabelle 4.4 findet sich eine tabellarische Übersicht über die von CIM und ITPMF unterstützen Metamodellierungskonzepte.

	EMOF	CMOF	UML	CIM	ITPMF
Klassen	√	√	√	√	√
Abstrakte Klassen	√	√	√	√	√
Attribute	√	√	√	√	√
vordefinierte Datentypen	√	√	√	√	√
erweiterbare Datentypen		√	√	√	√
Pakete	√	√	√	√	√
Paketschachtelung	√	√	√	√	√
Paketimport		√	√	√	√
Paketvereinigung		√	√		
Binäre Assoziationen	√[20]	√	√	√	√

[20] Konzept kann durch ”verschränkte” Attribute nachgebildet werden

	EMOF	CMOF	UML	CIM	ITPMF
n-äre Assoziationen			√		
Assoziationsnamen		√	√	√	√
Rollennamen	√[20]	√	√	√	√
Multiplizitäten	√[20]	√	√	√	√
Navigierbarkeit		√	√		
Vererbung	√	√	√	√	√
Aggregation	√[20]	√	√	√	√
Komposition	√[20]	√	√	√	√
Assoziationsklassen			√		
Bedingungen		√	√		√
Stereotypen			√		√
Trigger/Event			√	√	

Tabelle 4.4: Tabellarischer Überlick der in standardisierten Informationsmodellen verwendeten Metamodellierungskonzepte

4.5 Zusammenfassung der Analyseergebnisse

Die in diesem Kapitel durchgeführte Analyse lässt sich in zwei Schritte unterteilen. Den ersten Schritt bildet die Analyse der Modellierungssprachen EMOF, CMOF und UML in Kapitel 4.1, die die von der jeweiligen Modellierungssprache unterstützten und definierten Konzepte untersuchte. Den zweiten Schritt bildet die Analyse der Informationsmodelle von Unternehmen und Werkzeugherstellern in Kapitel 4.2 und 4.3, die durch die Analyse von standardisierten Informationsmodellen in Kapitel 4.4 abgerundet wurde.

Die Analyse der Modellierungssprachen ergab eine unterschiedliche Zielsetzung der Sprachen. So besitzt die MOF im Gegensatz zur UML den Anspruch der Implementierbarkeit. Die Basis für die Spezifikation der Modellierungssprachen MOF und UML ist das Core-Paket der UML 2.0 Infrastructure. Dabei bildet die Essential MOF durch Importierung des Basic-Pakets die elementarste Menge an Modellierungskonzepten. Auf den Konzepten der Essential MOF aufbauend ist die Complete MOF definiert, die

durch Importierung des Constructs-Paktes noch weitere Modellierungskonzepte, z.B. im Bereich der Assoziationen hinzufügt. Die größte und umfassenste Menge an Modellierungselementen stellt die Unified Modeling Language bereit, die zusätzlich mit der Superstructure noch verschiedene Diagrammtypen für die Dokumentation bereitstellt. Abbildung 4.16 visualisiert die Verwendung des Core-Pakets als gemeinsame Basis der Modellierungssprachen.

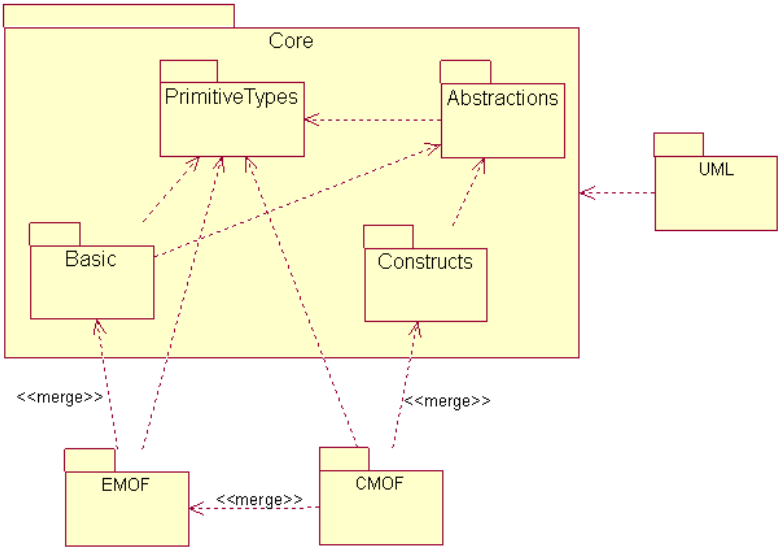


Abbildung 4.16: Das Core-Paket als Basis der Modellierungssprachen

In einem zweiten Schritt wurden die, von Unternehmen und Werkzeugherstellern aufgebauten Informationsmodelle auf die, in ihnen verwendeten Modellierungskonzepte untersucht. Tabelle A in Anhang A enthält eine Gesamtübersicht, der in Informationsmodellen von Unternehmen, Werkzeugherstellern und Standards verwendeten Konzepte und stellt sie den, in den Modellierungssprachen EMOF, CMOF und UML definierten Modellierungselementen gegenüber.

Bei der Analyse der verwendeten Konzepte, muss der unterschiedliche Schwerpunkt der Informationsmodelle von Unternehmen und Werkzeugherstellern berücksichtigt werden. Während Unternehmen den Schwerpunkt bei der Erstellung ihrer Informationskonzepte auf die Modellierung legen und damit auf die Informationsmodellebene, fokussieren die Werkzeughersteller auf die Verwendung des Informationsmodells zur Generierung von unterschiedlichen Softwarekarten, Views und Reports und befinden sich damit auf der Informationsobjektebene. Durch diese unterschiedlichen Schwerpunkte lassen sich Unterschiede bei den verwendeten Konzepten erklären.

Als exemplarisches Beispiel soll die Verwendung des Paket-Konzepts in den Informationsmodellen der Unternehmen und seine Unterstützung in den Modellen der Werk-

zeughersteller näher betrachtet werden: Das Paket-Konzept wird in allen untersuchten Informationsmodellen von Unternehmen verwendet, um eine Reduzierung der Komplexität und übersichtliche Modellierung der einzelnen Modelle zu gewährleisten und gleichzeitig den Zusammenhang zwischen den Modellen zu veranschaulichen. Bei den Werkzeugherstellern unterstützt lediglich ein Tool das Paket-Konzept, da dieses auf Objektebene zur Erzeugung von Karten, Views und Reports keine Rolle spielt.

Als Ergebnisse der Analyse lassen sich die in EMOF, CMOF und UML definierten Metamodellierungskonzepte in drei Kategorien bezüglich ihres Wichtigkeit als Metamodellierungskonzepte des Informationsmodells zum Enterprise Architecture Management einteilen:

- **obligatorisch:** Zu den Modellierungskonzepten, die ein Informationsmodell zum Enterprise Architecture Management unterstützen muss, gehören die elementaren Konzeptes aus denen jedes Informationsmodell aufgebaut ist. Zu diesen Konzepten zählen: Klassen, Attribute, primitive Datentypen und binäre Assoziationen. Jedes der analysierten Informationsmodelle von Unternehmen, Herstellern und Standardisierungen unterstützt diese Konzepte.

- **optional:** Neben den elementaren Modellierungskonzepten gibt es weitere Konzepte die für die Modellierung eines Informationsmodells von Nutzen, aber nicht zwingend notwendig sind. Zu diesen Konzepten zählen: Rollennamen, Assoziationsnamen, Multiplizitäten, Aggregationen, Kompositionen und Bedingungen. Diese Konzepte erlauben eine genauere Spezifikation der möglichen Ausprägungen von Informationsobjekten und können zu einer verbesserten Datenqualität beitragen. Ebenso in diese Katetorie einordnen lassen sich Konzepte, wie abstrakte Klassen, Vererbung, Pakete, Paketschachtelungen, Paketimport. Diese Modellierungskonzepte tragen viel zur übersichtlichen Modellierung von Informationsmodellen bei und werden aus diesem Grund in fast allen analysierten Modellen verwendet, verlieren aber auf der Ebene von Informationsobjekten ihre Bedeutung.

- **nicht benötigt:** In die dritte Kategorie lassen sich Modellierungskonzepte einordnen, die zur Modellierung eines Informationsmodells zum Enterprise Architecture Management nur einen vernachlässigbaren Beitrag leisten können. Zu diesen Konzepten zählen: Stereotypen, n-äre Assoziationen, Trigger und Events. Diese Modellierungskonzepte werden in den analysierten Informationsmodellen gar nicht oder nur vereinzelt verwendet. Das Konzept der Navigierbarkeit hat in der Programmierung einen großen Einfluss auf die Umsetzung in Code, während es bei den Informationsmodellen nur eine untergeordnete Rolle spielt[21]. Ebenso in diese Gruppe einordnen lässt sich das Modellierungskonzept der Assoziationsklasse. Für jede Assoziationsklasse gibt es, wie in Kapitel 4.1.2 gezeigt, eine äquivalente Umformung in Informationsmodellen.

[21] Lediglich zwei von acht analysierten Informationsmodellen unterstützen und verwenden das Konzept.

Betrachtet man lediglich die obligatorischen Metamodellierungskonzepte, so ist die EMOF ausreichen, wobei binäre Assoziationen über das Konstrukt der verschränkten Attribute abgebildet werden müssen. Eine visuelle Modellierung müsste dann beispielsweise Linien mit ausschließlich Assoziationsnamen zwischen zwei Klassen auf zwei verschränkte Attribute mit gleichem Namen transformieren.

Nimmt man als Basis jedoch insbesondere die analysierten Modelle der Unternehmen, so verwenden drei von vier Modellen sowohl Rollen- und Assoziationsnamen als auch Pakete zur Strukturierung. Soll zusätzlich eine Modularisierung des Informationsmodells möglich sein (vgl. Lankes et al. [LMW05b]), so sind die Konzepte *Import* und/oder *Merge* unumgänglich. Des Weiteren sind Bedingungen, wie sie beispielsweise in Abbildung 4.11 dargestellt sind, ein Werkzeug zur Erhöhung der Datenqualität, und können genutzt werden, um Assoziationsklassen in Informationsmodellen aufzulösen.

Aus diesen Forderungen lässt sich ableiten, das EMOF nicht ausreichend ist, um die Anforderungen an komplexe und wohl strukturierte Informationsmodelle zu implementieren und die Konzepte von CMOF benötigt werden.

5 Analyse von Transformationssprachen

Das Werkzeug zur Softwarekartographie, das im Rahmen des Forschungsprojektes entstehen, nutzt zwei Arten von Transformationen. Zum einen die Transformation zwischen dem semantischen und dem symbolischen Modell (siehe Abschnitt 3.1.3) und zum anderen die Transformation zwischen zwei semantischen Modellen. Die Motivation für die Transformation zweier semantischer Modelle rührt entweder von der Wiederverwendung existierender Viewpointdefinitionen her, oder basiert auf der Problemstellung mehrere semantische Modelle in ein gemeinsames Informationsmodell zusammenzuführen. In dieser Arbeit wird im Weiteren ausschließlich auf die Transformation zwischen semantischen Modellen eingegangen. Aufbauend auf der Analyse existierender und relevanter Informationsmodelle in Kapitel 4 sollen im folgenden Kapitel verschiedene existierender Transformationssprachen auf ihre Eignung zur Transformation von Informationsmodellen untersucht werden.

Einleitend in das Themengebiet der Modelltransformationen werden in diesem Kapitel verschiedene, existierende Transformationssprachen analysiert und klassifiziert. Der folgende Abschnitt 5.1 beschreibt eine Reihe von Eigenschaften von Spezifikationssprachen, mit deren Hilfe sich eine Klassifikation der Sprachen erreichen lässt. Diese Klassifikationseigenschaften bilden den Ausgangspunkt für die, in den folgenden Abschnitten vorgestellte Analyse der Spezifikationssprachen zur Modelltransformation (Abschnitt 5.2). Untersucht werden die Spezifikationssprachen *Eclipse Modeling Framework* (EMF) siehe Abschnitt 5.2.1, *Atlas Transformation Lanuage* (ATL) siehe Abschnitt 5.2.2, *Model Transformation Language* (MTL) siehe Abschnitt 5.2.3 und die *Bidirectional Object-Oriented Transformation Language* (BOTL) siehe Abschnitt 5.2.4. Die vorgestellten Transformationssprachen stellen eine repräsentative Auswahl existierender Ansätze dar, und eignen sich aufgrund der Verwendung des MOF-Ansatzes als Basis der Spezifikation für die Transformation von semantischen Modellen. Abschnitt 5.3 fasst die Analyseergebnisse der Transformationssprachen bezüglich ihrer Eignung zur Transformation von Informationsmodellen zusammen.

5.1 Klassifikation von Modelltransformationssprachen

Dieser Abschnitt beschreibt Eigenschaften von Spezifikationssprachen zur Modelltransformation, die eine Klassifizierung der Sprachen ermöglichen und einen Überblick, über die unterschiedlichen Sprachansätze bieten. Basis bilden die Klassifika-

tionsansätze von Modelltransformationssprachen in den Arbeiten von Czarnecki et al. [CH03], Gardner et al. [Gar03], Marschall [Mar04] und Prakash et al. [PSS06]. Die hier getroffene Auswahl an Klassifikationsmöglichkeiten wurde anhand der Anwendungsdomäne, der Transformation von semantischen Informationsmodellen getroffen. Eine weiterführende Taxonomie, wie sie zum Beispiel Czarnecki et al. für Codetransformationen trifft, ist für die Transformation von Informationsmodellen nicht von Bedeutung.

Transformationskategorie: Modelltransformationen lassen sich in zwei Hauptkategorien einteilen: *Modelltransformationen* (model-to-model) und *Codetransformationen* (model-to-code). Dabei können die Codetransformationen als ein Speziallfall der Modelltransformationen angesehen werden, da für jede Programmiersprache ein Metamodell, das die Sprache spezifiziert, angegeben werden kann. Einige Spezifikationssprachen unterstützen sowohl Modelltransformationen als auch Codetransformationen.

Sprachparadigma: Die verwendeten Spezifikationsprachen lassen sich in *deklarative, imperative* oder *hybride* Typen unterteilen. Eine deklarative Sprache definiert das Ergebnis, das "was", mittels des Konzeptes der Belegung, während eine imperative Sprache befehlsorientierten aufgebaut ist und das "wie" beschreibt. Hybride Sprachen verwenden eine Kombination von deklarativen und imperativen Konstrukten zur Definition von Transformationen. Die Auswahl der verwendeten Sprache zur Spezifikation von Transformationen hat großen Einfluss auf die Mächtigkeit und Benutzerfreundlichkeit des Ansatzes. So sind imperative Sprachansätze im allgemeinen mächtiger während deklarative Ansätze benutzerfreundlicher sind. Die OMG empfiehlt in ihrem RFP für die QVT [OMG05b] die Verwendung einer deklarativen Sprache.

Bidirektionalität: Transformationen können *unidirektional* oder *bidirektional* sein. Unidirektionale Transformationen können nur in eine Richtung ausgeführt werden, dabei wird aus einem Quellmodell ein Zielmodell berechnet oder aktualisiert. Bei einer bidirektionalen Transformation ist eine Ausführung der Transformation in beide Richtungen möglich, um eine Synchronisation zwischen den Modellen zu erreichen. Diese Bidirektionalität kann durch unterschiedliche Ansätze erreicht werden:

- **Bidirektionale Regeln:** Jede Transformationsregel ist bijektiv und kann in beide Richtungen ausgeführt werden.
- **Inverse Unidirektionale Regeln:** Zu jeder Transformationsregel kann eine inverse Regel angegeben werden, die die Transformation umkehrt.

Die Invertierbarkeit einer Transformation ist nicht nur von der Invertierbarkeit der Transformationsregeln abhängig, sondern auch von der Invertierbarkeit des Regelablaufs.

Kopplung der Modelle: Die Kopplung zwischen Modellen lässt sich in *ungekoppelt, unidirektional gekoppelt* und *bidirektional gekoppelt* unterscheiden. Die Möglichkeiten der Kopplung von Quell- und Zielmodellen ergeben sich aus der Direktionalität einer

Transformation. Modelle können voneinander unabhänig oder abhängig sein. Eine Unidirektionalität der Transformation ermöglicht eine unidirektionale Kopplung der Modelle, bei der Änderungen im Quellmodell an das Zielmodell weitergeleitet werden. Bei einer Bidirektionalität der Transformation besteht die Möglichkeit einer bidirektionale Kopplung der Modelle, bei der auch Änderungen in dem Zielmodell an das Quellmodell progagiert werden.

Werkzeugunterstützung: Um eine Nutzung der Spezifikationssprache in der Praxis zu ermöglichen, muss eine Werkzeugunterstützung vorhanden sein. Die Art der Werkzeugunterstützung hat wesentliche Auswirkungen auf die Anwenderfreundlichkeit. Das Werkzeug sollte eine Wiederverwendung bereits vorhandener Modelldokumentationen (z.B. Rational Rose Modelle, XMI) durch Importe ermöglichen.

Anwenderfreundlichkeit: Die Anwenderfreundlichkeit ist ein wesentliches Kriterium für die Benutzerakzeptanz einer Spezifikationsprache. Für die Anwenderfreundlichkeit sind vor allem die Lesbarkeit und Verständlichkeit der Notation von Bedeutung. Eine graphische Notation ist dabei einer textuellen vorzuziehen, ebenso die Verwendung bekannter und akzeptierter Konzepte (wie z.B. UML-Notationen). Des Weiteren spielt die Abstraktionsmöglichkeit, Komponierbarkeit und die damit verbundene Wiederverwendung von Teilspezifikationen eine Rolle für die Anwenderfreundlichkeit. Die Anwenderfreundlichkeit stellt ein subjektives Klassifikationskriterium da.

Umgang mit Fehlern: Im Verlauf einer Modelltransformation können Fehler auftreten, deren Vermeidung nicht durch die Syntax der verwendeten Sprache sichergestellt ist, wie z.B. die Erzeugung eines Zielmodells, das nicht dem Zielmetamodell entspricht. Mögliche Reaktionen auf auftretende Fehler sind der Abbruch der Transformation, das Ignorieren der Fehler und transaktionsgeschützte Transformationen. Im letzten Fall verlaufen speziell gekennzeichnete Teile der Transformation entweder fehlerfrei oder gar nicht[1]. Zusätzlich können Informationen, über die aufgetretenen Fehler, durch ein sogenanntes *Logging* gesammelt werden, um dem Benutzer bei der Fehlerfindung zu helfen.

5.1.1 Anforderungen an die Spezifikationssprache zur Informationsmodell-Transformation

Bei der Transformation von Informationsmodellen von Unternehmen in ein geeignetes Informationsmodell zum Enterprise Architecture Mangement handelt es sich um eine Modelltransformation (model-to-model). Für diese Art von Transformation ist ein deklarativer Sprachansatz aus Gründen der Anwenderfreundlichkeit, einem rein imperativen Sprachansatz vorzuziehen. Eine Bidirektionalität der Transformation ist ebenso wie eine bidirektionale Kopplung der Modelle wünschenswert, um Änderungen an den Informationsobjekten automatisch sowohl in das Quellmodell, als auch in das Zielmodell propagieren zu können.

[1] Vergleiche Transaktionen bei Datenbanken [KE04]

Für die praktische Nutzung der Transformation der Informationsmodelle von Unternehmen in ein geeignetes Informationsmodell zum Enterprise Architecture Management ist eine Werkzeugunterstützung der Spezifikationssprache unerlässlich. Ebenso wäre eine Importmöglichkeit existierender Modelldokumentationen - wie z.B. Rational Rose UML-Modelle - wünschenswert.

Als Vorraussetzung für die Definition von Transformationsregeln kann ein Grundverständnis der Modellierung angenommen werden, da die Spezifikation der Transformationsregeln von Modellierungsexperten vorgenommen werden sollte.

Die Spezifikationssprache sollte Möglichkeiten zur Verifikation der Transformationsregeln beinhalten, um das Erzeugen fehlerhafter Modelle zu verhindern. Ebenso sollte sie einen Validierungsmechanismus bereitstellen, mit dem überprüft werden kann, ob ein generiertes Zielmodell auch tatsächlich eine Instanz des Zielmetamodells darstellt.

In den folgenden Abschnitten sollen unterschiedliche Spezifikationssprachen zur Modelltransformation und ihre Werkzeugunterstützungen nach den oben genannten Eigenschaften klassifiziert und auf ihre Verwendbarkeit zur Transformation von semantischen Modellen in ein geeignetes Informationsmodell zum Enterprise Architecture Management überprüft werden.

5.2 Analyse existierender Transformationssprachen

Aufbauend auf der Klassifikation von Transformationssprachen und den Anforderungen an die Spezifikationssprache zur Informationsmodell-Transformation werden in den folgenden Abschnitten existierende Transformationssprachen vorgestellt und klassifiziert.

5.2.1 Eclipse Modeling Framework (EMF)

Das *Eclipse Modeling Framework* (EMF) [Bud03] ist ein Java-Framework und eine Code-Generierungs-Engine, die von der *Eclipse Foundation* als Plug-In für ihre Open-Source-Plattform *Eclipse* entwickelt wird. Es besteht aus drei Komponenten: Dem *EMF*, das das Metamodell *ecore* enhält, dem *EMF.Edit*, das wiederverwendbare Klassen anbietet, um Editoren für EMF-Modelle zu konstruieren, und dem *EMF.Codegen* zur automatischen Generierung von Java-Code. EMF stellt Mechanismen für die Transformationen von UML-Klassendiagrammen in Java-Code bereit. Es stellt ein Framework zur Verfügung, das es ermöglicht Modelle in einem der drei Formate XML, UML oder Java zu definieren und die jeweils anderen Darstellungen daraus zu generieren. Die interne Speicherung und Modelldefinition des EMF-Modells erfolgt im ecore-XMI-Format (*XML Metadata Interchange*). Abbildung 5.1 veranschaulicht die Beziehung zwischen XML, UML, Java und dem EMF-Modell.

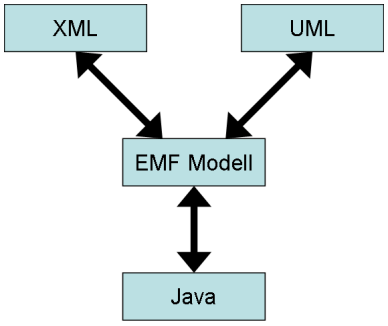


Abbildung 5.1: Zusammenhang zwischen XML, UML, EMF und Java nach [Bud03]

Das Metamodell des EMF-Modells wird ecore genannt und ist sehr nah an die Spezifikation der MOF angelehnt. Ebenso wie die MOF definiert sich das ecore rekursiv selbst, um eine endlose Anzahl an Metaebenen zu verhindern. EMF kann als effiziente Java Implementierung eines Teils der MOF-Programmierschnittstelle angesehen werden. Abbildung 5.2 zeigt einen vereinfachten Ausschnitt des *Ecore*-Modells.

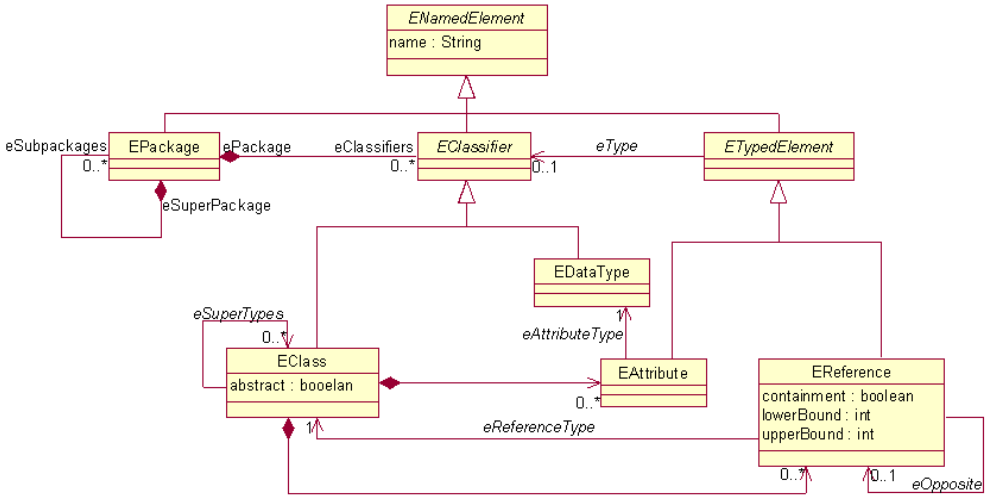


Abbildung 5.2: Vereinfachtes Ecore-Metamodell nach [Bud03]

Die von EMF unterstützten Metamodellierungskonzepte stimmen annähernd mit den in CMOF definierten Konzepten überein. Die Basis der Modellierung bilden Klassen, Attribute und Assoziationen. Das Paketkonzept mit dem Prinzip der Schachtelung wird ebenso unterstützt, wie der Paketimport. Im Gegensatz zu CMOF findet keine Unterstützung von Bedingungen statt. Eine tabellarische Übersicht über die, in EMF verfügbaren Metamodellierungskonzepte, bietet Tabelle 5.5.

Ziel von EMF ist es, Modelle auch aus anderen Formaten, wie z.B. einem relationalen Datenbank-Schema importieren zu können. Aus dem Modell kann mit Hilfe des EMF effizienter und stabiler Java-Code erzeugt werden, der leicht anpassbar ist.

Analyseergebnisse:

EMF stellt ein Werkzeug für die modellgetriebene Entwicklung bereit. Es lässt sich in die Kategorie der *Codetransformationen* einordnen und bietet automatische Codegenerierung von UML-Klassendiagrammen. Die Spezifikation von Transformationsregeln entfällt durch die automatische Generierung von Java-Code. Als Ausgangspunkt für die Erstellung eines Modells in EMF bieten sich vier Möglichkeiten an: Die Definition eins *XML Schemas*, annotierte Java-Interfaces, der Import eines UML-Modells aus Rational Rose oder die direkte Definition eines *Ecore* Modells. Eine exemplarische Anwendung des EMF mit allen vier Darstellungsmöglichkeiten findet sich im Anhang B.1. Die Aktualisierung der einzelnen Modelle erfolgt im Eclipse automatisch, spätestens mit der Speicherung einer Änderung.

Die Anwenderfreundlichkeit ist durch die automatische Generierung ohne Spezifikation von Transformationsregeln und die verschiedenen Importmöglichkeiten sehr hoch. Das Entstehen von Fehlern wird durch die automatische Transformation größtenteils verhindert, treten während der Transformation dennoch Fehler auf, so führen diese zu einem Abbruch der Transformation und einer automatischen Protokollierung der aufgetretenen Fehler.

Das EMF bietet noch eine Reihe weiterer Fähigkeiten, wie z.B. die *Reflection API* mit deren Hilfe schreibend und lesend auf die Modelle zugegriffen werden kann und die Möglichkeit der Registrierung von *EventListeners* an EMF-Objekten, die bei einer Änderung an Attributen oder Referenzen benachrichtigt werden.

Tabelle 5.1 fasst die Analyseergebnisse für das *Eclipse Modeling Framework* zusammen.

	Eclipse Modeling Framework
Transformationskategorie	Codetransformation
Sprachparadigma	—
Bidirektionalität	bidirektional
Kopplung der Modelle	bidirektional
Werkzeugunterstützung	eclipse
Importmöglichkeiten	Rose-Modelle, ecore, XML Schema, Java-Code

	Eclipse Modeling Framework
Anwenderfreundlichkeit	sehr hoch
Umgang mit Fehlern	Abbruch und Logging

Tabelle 5.1: Tabellarischer Überblick der Analyseergebnisse des Eclipse Modeling Framework

5.2.2 Atlas Transformation Language (ATL)

Die *Atlas Transformation Language* (ATL) [Uni05b] entstand als Teil der *Atlas Model Management Architecture* (AMMA) [Uni05a] am *Institut National de Recherche en Informatique et en Automatique* (INRIA) [INR05b]. ATL wurde entwickelt um Modelltransformationen und Codetransformationen, wie sie im *QVT RFP* [OMG05b] der OMG definiert sind, zu beschreiben. Es ist ein Modelltransformationssprache, die sowohl ein Metamodell, als auch eine konkrete Syntax definiert. In ATL werden Transformationen selbst als Modelle angesehen siehe Abbildung 5.3.

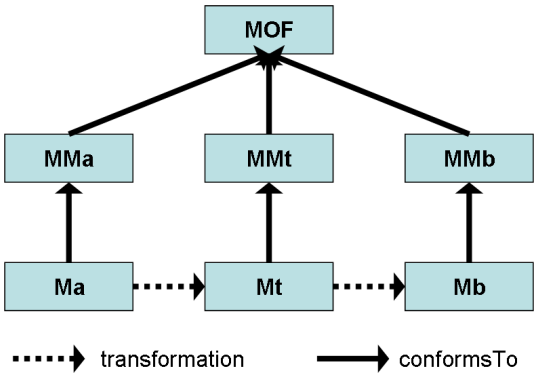


Abbildung 5.3: Transformationen in ATL nach [Uni05b]

An einer Transformation sind demnach drei MOF-konforme Metamodelle (Quellmetamodell MMa, Zielmetamodell MMb und Transformationsmetamodell MMt) beteiligt. Die Transformation des Quellmodells (Ma) in das Zielmodell (Mb) wird durch die Transformationsdefinition (Mt), die in der ATL Spezifikationssprache definiert ist, ausgeführt.

ATL ist eine hybride Spezifikationssprache, die sowohl deklarative als auch imperative Sprachkonstrukte unterstützt. Obwohl der bevorzugte Stil für die Definition von Trans-

formationsregeln der deklarative ist, unterstützt ATL die Definition von imperativen Konstrukten, um die Definition komplexer Transformationsregeln zu ermöglichen[2].

ATL Transformationen sind unidirektionale Transformationen, die auf *read-only* Quellmodellen arbeiten und *write-only* Zielmodelle erzeugen, d.h. in Quellmodellen kann während der Ausführung der Transformation nur navigiert, aber nichts geändert werden, während in Zielmodellen nicht navigiert werden kann, aber Änderungen vorgenommen werden können. Dies hat zur Folge, dass die Zuweisung einer Referenz einer Instanz des Zielmodells keine Referenz auf eine andere Instanz des Zielmodells enthält, sondern die Referenz auf die Instanz des Quellmodells, aus dem die Instanz des Zielmodells generiert wurde oder wird. Bidirektionale Transformationen können als Paare unidirektionaler Transformationsregeln - jeweils eine Transformationsspezifikation für jede Richtung - definiert werden.

ATL Transformationsdefinitionen werden *Modules* genannt und bestehen aus:

- **Header**-Abschnitt, der den Namen des Moduls und des Quell- und Zielmodells definiert.
- **Import**-Abschnitt, der die Bibliotheken angibt, die für das Spezifikation benötigt werden.
- **Helpers**-Abschnitt, der verwendet wird um globale Variablen und Funktionen zu definieren.
- **Rules**-Abschnitt, der die Transformationsregeln festlegt.

Transformationsregeln in ATL bestehen aus einem *Source Pattern* und einem *Target Pattern*. Das Source Pattern enthält eine Menge von *Source Types* (Elemente des Quellmetamodells oder von OCL) und einem *Guard* (boole'scher Ausdruck) und wird zu einer Menge von *Matches* im Quellmodell ausgewertet.

Das Target Pattern besteht aus einer Variablendeklaration, die einem Typ des Zielmetamodells entspricht und einer Menge von *Bindings*. Für einen gegebenen Match werden die Elemente des entsprechenden Typs im Zielmodell erzeugt und mit Hilfe der *Bindings* initialisiert.

Zusätzlich werden bei einem Match Links erzeugt, die eine Navigation des Zielmodells durch eine Navigation des Quellmodells und anschließende Verfolgung des Links, ersetzen. Ein Link besteht aus der Verbindung dreier Komponenten: Der Regel, dem *Match* (z.B. Quellelemente) und den neu generierten Zielelementen.

Eine exemplarische Transformation eines Modells mit Hilfe der ATL enthält Anhang B.2 [JK05].

[2] Der imperative Sprachanteil ist in der aktuellen Version des ATL-Tools noch nicht enthalten. Einen Überblick über aktuell unterstützte Eigenschaften und zukünftige Erweiterungen bietet [JK05].

Für die praktische Anwendung stellt die ATL eine auf *Eclipse* basierende IDE bereit, die *ATL Developement Tools* (ATD). Die ATD besteht aus den folgenden Komponenten (wie in Abbildung 5.4) dargestellt: Einem *ATL Compiler* für die Transformation von ATL-Programmen in Bytecode, einer *ATL Virtual Maschine*, die den vom Compiler erzeugten Byte-Code ausführt, einem *Model Handler Abstraction Layer* der die Umsetzung der Befehle der VM auf die *Model Handler* übernimmt, den *Model Handlers* die die Programmierschnittstelle für die Modellmanipulation bereitstellen (z.B. EMF, MDR[3]) und einem *Model Repository* das eine persistente Speicherung der Modelle ermöglicht [JK05].

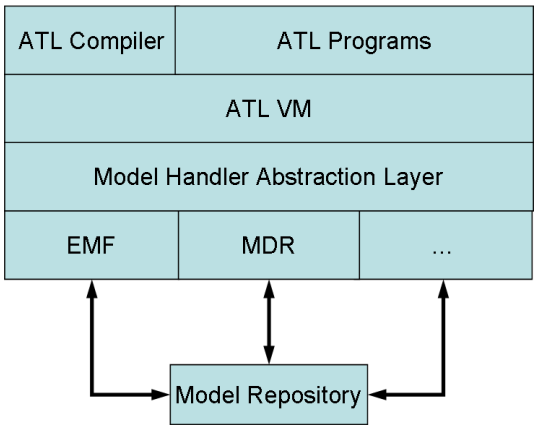


Abbildung 5.4: Architektur des ATL Werkzeuges nach [Uni05b]

Die ADT verwendet das *Eclipse Modeling Framework* (EMF) für den Umgang (Serialisierung, Deserialisierung, Navigation und Modifikation) mit den Modellen. Die ADT beinhaltet ein speziell auf ATL-Spezifikationen zugeschnittene Erweiterung des *Eclipse Debugging Frameworks*, das ein Debuggen des Transformationsprogramms ermöglicht. Bei einem Abbruch der Transformation aufgrund eines Fehlers, ermöglicht das ADT eine Navigation des aktuellen Quell- und Zielmodells [Ecl05a].

Analyseergebnisse

ATL bietet eine Transformationssprache, die als Antwort auf den RFP QVT der OMG entwickelt wurde. Die Spezifikation von ATL erlaubt, neben der Transformation von MOF-konformen Modellen auch eine Transformation von Modellen, die Gegenstand anderer Technologien sind (z.B. Datenbanken, XML-Dokumenten). Mit Hilfe der ATL sind sowohl Transformationen von Modellen in Modelle als auch Transformationen von Modellen in Code möglich.

ATL definiert eine hybride Transformationssprache, deren Regeln unidirektional sind und keine Kopplung der Modelle unterstützt. Eine Bidirektionalität kann nur durch

[3]Metadata Repository

explizite Definition von Transformationsregeln für die Hin- und Rückrichtung erreicht werden. Für ATL steht die ADT als Werkzeugunterstützung bereit, die auf der *Eclipse Platform* basiert und Teil des GMT[4]-Projekts ist. Die ADT bietet unter anderem Editoren mit Syntax-Highlighting und einen Debug-Modus zum Auffinden von Fehlern in der Regelspezifikation an.

Die folgende Tabelle 5.2 bietet eine Übersicht über die Analyseergebnisse.

	Atlas Transformation Language
Transformationskategorie	Modelltransformation, Codetransformation
Sprachparadigma	hybrid
Bidirektionalität	unidirektional
Kopplung der Modelle	ungekoppelt
Werkzeugunterstützung	ATL Developement Tools (eclipse)
Importmöglichkeiten	XMI
Anwenderfreundlichkeit	hoch
Umgang mit Fehlern	Debugging möglich

Tabelle 5.2: Tabellarischer Überblick der Analyseergebnisse der Atlas Transformation Language

5.2.3 Model Transformation Language (MTL)

Die *Model Transformation Language* (MTL) [INR05a] ist eine stark an die Programmierung angelehnte Transformationssprache, die am INRIA entwickelt wird und auf MOF-Technologien aufbaut. Die Grundlage der MTL bildet ein *Pivot Metamodell*, mit dessen Hilfe eine Unabhängigkeit von unterschiedlichen Standards und eine Adaptionsfähigkeit erreicht werden soll.

Das Pivot Metamodel ist durch eine einfache Sprache definiert, die in MTL *BasicMTL* genannt wird. BasicMTL stellt die wichtigsten Konzepte, wie Klassen oder Attribute bereit. Eine Erweiterung der Konzepte von BasicMTL kann durch eine Erweiterung der abstrakten Syntax und durch die Definition einer Transformation der erweiterten auf die anfängliche Syntax erreicht werden. Alle Konzepte von MTL lassen sich so auf

[4]Generative Model Transformer [Ecl05a]

den kleinen BasicMTL-Kern zurückführen. Eine Übersicht über die, in BasicMTL enthaltenen Konzepte, bietet die Tabelle 5.5 in Abschnitt 47.

Die Entwicklung und Konzeption von MTL basiert auf vier Zielen:

- Verbesserung der Wiederverwendbarkeit von umfangreichen Transformationsdefinitionen
- Unabhängigkeit der Spezifikationssprache von Modellen und Repositories
- Generierung von domänen- und anwendungsspezifischer Frameworks
- Unabhängigkeit von Programmierungsstrategien

Das in MTL verwendete Sprachparadigma ist imperativ. Transformationen werden in MTL durch Befehlssequenzen definiert, die wie Programme durch Klassen, Attribute und Operationen gekennzeichnet sind. Den Einstiegspunkt dieser Programme bildet die *main()*-Methode, die in jeder Transformationsspezifikation genau einmal enthalten sein muss. Der Ablauf kann durch die Definition von Schleifen (*while, for*) und Bedingungen (*if*) festgelegt werden. Die Ausführungskontrolle kann durch die Verwendung eines *returns* unterbrochen werden.

Teile der Transformationsspezifikation können in sogenannten *Bibliotheken* organisiert werden, um das Ziel der Wiederverwendbarkeit von Transformationsdefinitionen zu ermöglichen. Eine exemplarische Transformationsspezifikation mit der MTL finden sich in Anhang B.3. Abbildung 5.5 veranschaulicht den Übersetzungsprozess einer MTL-Transformation.

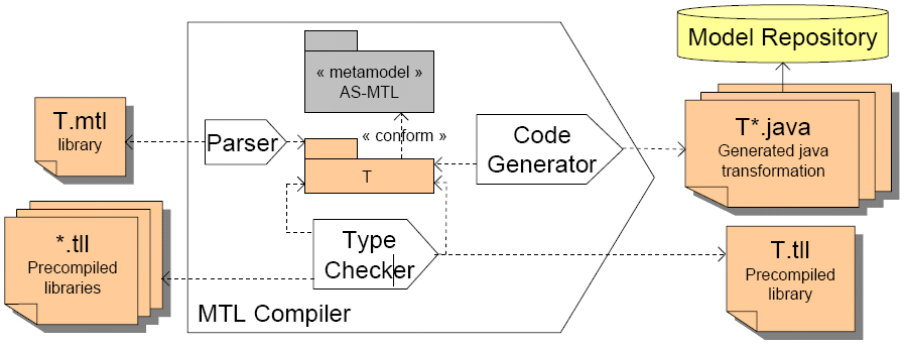


Abbildung 5.5: Übersetzungsprozess einer MTL-Transformation [SFS04]

Die Transformationsspezifikation (T) wird in einem ersten Schritt durch einen *Parser* eingelesen und in ein internes Modell, das *BasicMTL* konform ist, transformiert. Im zweiten Schritt wird das interne Modell durch den *Type Checker* um Typinformationen (z.B. für den Umgang mit Polymorphismus) angereichert. Für die Anreicherung kann

der Type Checker Bibliotheken verwenden, um vordefinierte Typen und Operationen benutzen zu können. In einem letzten Schritt werden durch den *Code Generator* aus dem internen Modell Java-Dateien erzeugt, die das Verhalten des internen Modells (T) implementieren und die Repositories, auf denen die Transformation definiert wurde, nutzen können.

Der gesamte Übersetzungsprozess basiert auf dem Metamodell der Transformation T, das zu *BasicMTL* konform ist. Die Schritte des Parsens, der Typ Überprüfung und der Code Generierung können als spezielle Transformationen auf dem MTL-Modell selbst angesehen werden [SFS04].

Eine Werkzeugunterstützung für die MTL wird von *Triskell* [IRI05] am *Institut de recherche en informatique et système aléatoires* (IRISA) entwickelt und ist aktuell in Version 0.06 als Plugin für Eclipse verfügbar. Das Werkzeug bietet eine MTL Implementierung an, die Transformationen von Modellen, die auf MOF-konformen Metamodelle basieren ermöglicht. Eine Importierung von EMF-, MDR-, ModFact- und Poseidon-Modellen wird von der aktuellen Version unterstützt. Das Werkzeug bietet außerdem Möglichkeiten zur Verwendung der Desing-Patterns Visitor, Observer und Interpreter an.

Analyseergebnisse

Die Model Transformation Language ist eine Spezifikationssprache die Code für Modelltransformationen MOF-konformer Modelle generiert. Im Rahmen der MDA kann die MTL als Spezifikationssprache für das PIM angesehen werden. Das für die Sprachspezifikation verwendete Paradigma ist imperativ und erlaubt die Definition von komplexen Transformationsregeln. Eine Bidirektionalität der Transformation wird ebenso wie eine Kopplung der Modelle in der aktuellen Version des MTL Werkzeuges nicht unterstützt. Die Werkzeugunterstützung von MTL basiert auf der Plugin-Technologie von Eclipse und stellt einen speziellen Editor für die textuelle Syntax von MTL, eine Ausführungsumgebung und eine Übersichtsdarstellung bereit.

Die vielseitige Werkzeugunterstützung erhöht die Anwenderfreundlichkeit, während die Verwendung einer imperativen Sprache für die Definition der Transformationsregeln sich negativ auswirkt. Tabelle 5.3 fasst die Analyseergebnisse der Model Transformation Language zusammen.

	Model Transformation Language
Transformationskategorie	Codetransformation
Sprachparadigma	imperativ
Bidirektionalität	unidirektional
Kopplung der Modelle	ungekoppelt

	Model Transformation Language
Werkzeugunterstützung	BasicMTL
Importmöglichkeiten	EMF, MDR, ModFact, Poseidon
Anwenderfreundlichkeit	mittel
Umgang mit Fehlern	Abbruch

Tabelle 5.3: Tabellarischer Überblick der Analyseergebnisse der Model Transformation Language

5.2.4 Bidirectional Object Oriented Transformation Language (BOTL)

Die *Bidirectional Object Oriented Transformation Language* (BOTL) [Mar04] ist eine graphische Spezifikationssprache für die Transformation objektorientierter Modelle, die am Lehrstuhl für *Software & System Engineering* der Technischen Universität München entwickelt wird. Die Spezifikation von Transformationsregeln in BOTL basiert auf einer mathematisch formalen Sprache, die es erlaubt Eigenschaften von Transformationsspezifikationen nachzuweisen. Sie bietet Techniken zur Überprüfung der Anwendbarkeit von Transformationsspezifikationen, sowie Verifikationstechniken für die Metamodellkonformität und Techniken zum Nachweis der Bidirektionalität von Transformationsregeln an.

BOTL stellt eine anwenderfreundliche, graphische Oberfläche für die Definition von Transformationsregeln bereit. Eine prototypische Implementierung von BOTL, basierend auf *ArgoUML* [Tig05] ist auf der BOTL-Webseite [Mar05] verfügbar. Abbildung 5.6 veranschaulicht die prinzipielle Funktionsweise und den Aufbau des BOTL-Werkzeuges.

Für die Modellierung von Metamodellen und Regelwerken wurde das UML-Werkzeug ArgoUML erweitert. Im oberen Teil der Abbildung ist die Werkzeugunterstützung zur Spezifikation von Transformationen dargestellt, während im unteren Teil der Abbildung der Interpreter zur Ausführung der Transformation visualisiert ist. Die Speicherung der Transformationsspezifikation erfolgt in einem XML-Dokument. Die Spezifikationen werden mittels der Verifikationskomponente[5] auf ihre Ausführbarkeit und Metamodellkonformität überprüft und eventuell auftretende Fehler an den Benutzer gemeldet. Die XML-Transformationsdokumente werden von dem *Transformer* zusammen mit den Modellen (Quellmetamodell, Zielmetamodell und Quellmodell) eingelesen und interpretiert. Mit Hilfe geeigneter Adapter können Modelle in unterschied-

[5] in der aktuellen Version nicht vollständig implementiert

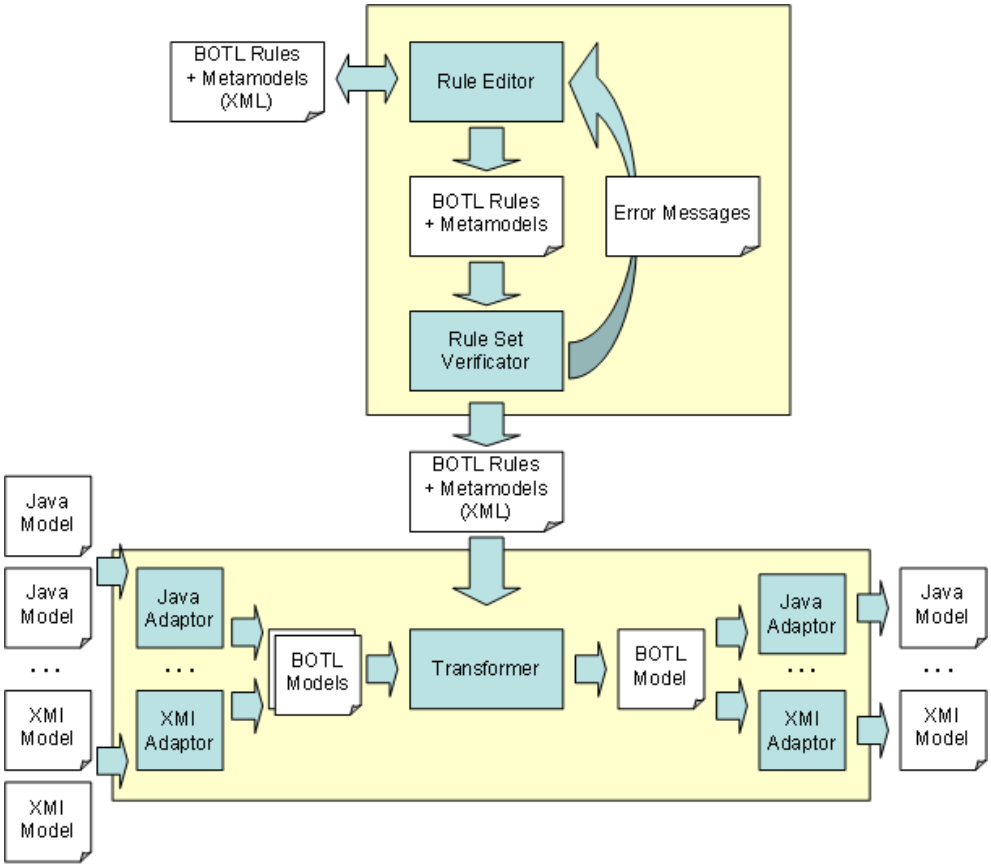


Abbildung 5.6: Architektur des BOTL Werkzeuges nach [Mar05]

lichen technischen Formaten für den *Transformer* verfügbar gemacht werden. In der aktuellen Version wird nur der Import von *Poseidon*-XMI-Modellen unterstützt.

Analyseergebnisse

Die Bidirectional Object Oriented Transformation Language stellt eine graphische Spezifikationssprache für Modelltransformationen von objektorientierten Modellen bereit. Das verwendete Sprachparadigma ist deklarativ und unterstützt eine graphische Spezifikation der Transformationsregeln, wodurch die Anwenderfreundlichkeit des Werkzeugs erhöht wird.

Die Verifikation der Bidirektionalität von Transformationen wird im Gegensatz zur der Metamodellkonformität in der aktuellen Version des Werkzeugs noch nicht unterstützt. Allerdings bietet das Werkzeug die Möglichkeit an, eine Regel automatisch zu invertieren.

Eine exemplarische Transformationsspezifikation, die mit dem Werkzeugs *ArgoUML4BOTL* erstellt wurde, findet sich in Anhang B.4. Die folgende Tabelle 5.4 fasst die Analyseergenisse für die BOTL tabellarisch zusammen.

	Bidirectional Object Oriented Transformation Language
Transformationskategorie	Modelltransformationen
Sprachparadigma	graphisch (deklarativ)
Bidirektionalität	bidirektional
Kopplung der Modelle	ungekoppelt
Werkzeugunterstützung	ArgoUML4BOTL
Importmöglichkeiten	Poseidon
Anwenderfreundlichkeit	sehr hoch
Umgang mit Fehlern	Verifikation vor Ausführung der Transformation

Tabelle 5.4: Tabellarischer Überblick der Analyseergebnisse der Bidirectional Object Oriented Transformation Language

5.3 Zusammenfassung der Analyseergebnisse

Die in den vorangegangenen Abschnitten vorgestellten Spezifikationssprachen für die Modelltransformation, entstanden aus unterschiedlichen Blickwinkeln und mit unterschiedlichen Schwerpunkten. In diesem Abschnitt werden die Analyseergebnisse der einzelnen Sprachansätze zusammengefasst und aufbauend auf die, in Abschnitt 5.1.1 vorgestellten Anforderungen, auf ihre Eignung zur Transformation von Informationsmodellen analysiert.

Entscheidender Kritikpunkt für die Eignung der Spezifikationssprache stellt ihre Unterstützung wichtiger (im Kontext der Modellierung von Informationsmodellen vgl. Abschnitt 4.5) Metamodellierungskonzepte dar. Die folgende Tabelle 5.5 gibt einen Überblick über die, von den Spezifikationssprachen EMF, ATL, MTL und BOTL unterstützten Metamodellierungskonzepte und stellt sie den Konzepten von EMOF, CMOF und UML gegenüber.

	EMOF	CMOF	UML	EMF	ATL	MTL	BOTL
Klassen	√	√	√	√	√	√	√
Abstrakte Klassen	√	√	√	√	√	√	√
Attribute	√	√	√	√	√	√	√
vordefinierte Datentypen	√	√	√	√	√	√	√
erweiterbare Datentypen		√	√	√	√	√	√
Pakete	√	√	√	√	√	√	
Paketschachtelung	√	√	√	√	√	√	
Paketimport		√	√	√	√	√	
Paketvereinigung		√	√				
Binäre Assoziationen	√[6]	√	√	√	√	√[7]	√
n-äre Assoziationen			√			√[7]	
Assoziationsnamen		√	√	√	√	√[7]	√
Rollennamen	√[6]	√	√	√	√	√[7]	√
Multiplizitäten	√[6]	√	√	√	√	√[7]	√
Navigierbarkeit		√	√	√	√	√[7]	√
Vererbung	√	√	√	√	√	√	√
Aggregation	√[6]	√	√	√	√	√[7]	√
Komposition	√[6]	√	√	√	√	√[7]	√
Assoziationsklassen			√				
Bedingungen		√	√		√		
Stereotypen			√	√			
Trigger/Event			√	√			

Tabelle 5.5: Tabellarischer Überlick über die, von Spezifikationssprachen unterstützten Modellierungskonzepte

[6] Konzept kann durch "verschränkte" Attribute nachgebildet werden.

[7] Laut [SFS04] werden Assoziationen, wie in EMOF über verschränkte Attribute abgebildet. Attribute in MTL besitzen jedoch nicht die Eigenschaften für Komposition, während die BNF (Backus Naur Form), der BasicMTL concrete Syntax [VD05] diese definiert.

Die folgenden Absätze fassen die Eignung der verschiedenen Spezifikationssprachen für die Transformation von Informationsmodellen zusammen:

Das *Eclipse Modeling Framework* wurde mit dem Schwerpunkt auf der Implementierung entwickelt. Seine Stärke ist die Generierung von leistungsfähigem und robusten Java-Code. Eine hohe Anwenderfreundlichkeit wird durch die automatische Erzeugung und Aktualisierung der einzelnen Modelle erzielt. Die Entwicklung des EMF als Plug-In für die Eclipse Platform führt zu einer sehr guten und ausgereiften Werkzeugunterstützung. Trotz der umfangreichen Unterstützung von Metamodellierungskonzepten ist das EMF als Spezifikationssprache zur Transformation von Informationsmodellen nicht geeignet, da es nur Codetransformationen (model-to-code) unterstützt.

Die *Atlas Transformation Language* stellt eine mächtige hybride Spezifikationssprache für Modelltransformationen bereit. Die ATL unterstützt Modelltransformationen (model-to-model) als Vorraussetzung für die Spezifikationssprache zur Transformation von Informationsmodellen. Die Verwendung eines akzeptierten Standards (OCL) zur Spezifikation von Transformationsregeln erhöht die Anwenderfreudlichkeit ebenso, wie die Entwicklung der Werkzeugunterstützung als Plug-In für Eclipse, die neben einem speziellen Editor auch das Debuggen von ATL Transformationsregeln ermöglicht. ATL unterstützt alle notwendigen Metamodellierungskonzepte für die Modellierung von Informationsmodellen. Die Eignung von ATL als Spezifikationssprache zur Tranformation von Informationsmodellen, erhöht sich durch die gegebene Importunterstützung von XMI-Modellen.

Die *Model Transformation Language* definiert eine Metasprache zur Modelltransformation die eng an die Programmierung angelehnt ist. Die Verwendung eines imperativen Sprachansatzes ermöglicht die Definition komplexer Transformationsregeln wirkt sich aber gleichzeitig negativ auf die Anwenderfreundlichkeit aus. Die Werkzeugunterstützung der MTL ist als Plug-In Technologie für Eclipse realisiert. Die MTL verfolgt ähnlich der EMOF einen minimalistischen Ansatz und unterstützt nur die notwendigsten Metamodellierungskonzepte. Der imperative Sprachansatz und die minimalistische Unterstützung von Metamodellierungskonzepten schränken die Eignung von MTL als Spezifikationssprache zur Modelltransformation von Informationsmodellen ein.

Die *Bidirectional Object Oriented Transformation Language* bietet eine auf die Transformation von UML-Klassendiagrammen spezialisierte Spezifikationssprache. Die Definition von Transformationsregeln erfolgt durch ein anwenderfreundliches graphisches Prinzip. Die Werkzeugunterstützung von BOTL ist durch ArgoUML4BOTL realisiert und bietet eine Importmöglichkeit für Poseidon-Modelle. Die zur Transformation von Informationsmodellen benötigten Metamodellierungskonzepte werden von BOTL größtenteils unterstützt, lediglich die Verwendung des Paketkonzepts wird nicht vollständig umgesetzt. Sieht man von den beschränkten Importmöglichkeiten und der einge-

schränkten Unterstützung von Metamodellierungskonzepten, vor allem dem fehlenden Constraint-Konzept, der BOTL ab, eignet sie sich aufgrund der hohen Anwenderfreundlichkeit für die Transformation von Informationsmodellen.

Als Spezifikationssprache zur Transformation von Informationsmodellen bietet sich aufgrund der in der Analyse gewonnenen Erkenntnisse die *Atlas Transformations Language* an. Im folgenden Kapitel 6 soll eine exemplarische Transformation von Informationsmodellen, die in diesem und den vorangegangenen Kapiteln entwickelten Konzepte validieren.

6 Transformation von Informationsmodellen

Nachdem in Kapitel 4 die Analyse existierender und relevanter Informationsmodelle vorgestellt und in Kapitel 5 existierende Spezifikationssprachen zur Modelltransformation auf ihre Eignung zur Transformation von Informationsmodellen hin überprüft wurden, soll in diesem Kapitel die Anwendbarkeit des MOF-Ansatzes zur Modellierung von Informationsmodellen und als Metamodell zur Modell-basierten Transformation von Informationsmodellen validiert werden.

Im folgende Abschnitt 6.1 wird anhand einer Anforderungsanalyse aus einem existierenden Informationsmodell eines Projektpartners des Forschungsprojektes Softwarekartorgraphie ein neues Teilinformationsmodell, als Instanz von CMOF, zur automatischen Generierung bestimmter Softwarekartentypen entwickelt. Aufbauend auf der Konzeption des Teilinformationsmodells wird in Abschnitt 6.2 eine exemplarische Modelltransformation des ursprünglichen Informationsmodells, in das neu entwickelte Teilinformationsmodell durchgeführt, um die in den vorangegangenen Kapiteln vorgestellten Konzepte zu verifizieren. Ein Resümee der Transformationsformationsergebnisse in Abschnitt 6.3 bildet den Abschluss dieses Kapitels.

6.1 Konzeption eines exemplarischen Informationsmodells

Ausgehend von der Vorstellung eines existierenden Informationsmodells eines Projektpartners in Abschnitt 6.1.1 und der Anforderungsanalyse an ein Informationsmodell anhand von zwei Softwarekartentypen in Abschnitt 6.1.2 wird in Abschnitt 6.1.3 ein neues Teilinformationsmodell entwickelt, das als Basis für die automatische Generierung der vorgestellten Softwarekartentypen dienen kann.

6.1.1 Vorstellung eines existierenden Informationsmodells

Ausgangspunkt für die Konzeption eines Teilinformationsmodells zur automatischen Generierung bestimmter Softwarekarten bildet der folgende Ausschnitt (vgl. Abbildung 6.1) eines existierenden Informationsmodells eines Projektpartners des Forschungsprojektes Softwarekartographie.

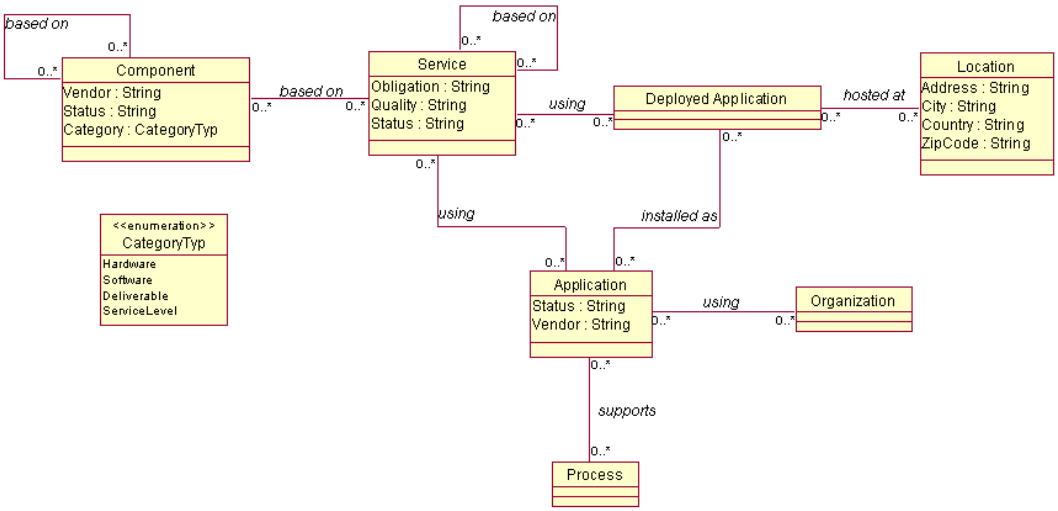


Abbildung 6.1: Ausschnitt aus dem Informationsmodell eines Projektpartners

Das vorgestellte Informationsmodell visualisiert den logischen Aufbau eines *Services* aus *Components* und die Unterstützung eines *Processes* durch *Applications*. Alle Assoziationen des Informationsmodells haben eine Multiplizität von 0..*, da das vom Projektpartner verwendete Werkzeug zum Enterprise Architecture Management keine explizite Angabe von Multiplizitäten unterstützt. Eine Simplifizierung des Informationsmodells wird durch das Weglassen der obligatorischen Attribute *Name* und *Description* vom Typ String erreicht, die in der abstrakten Superklasse *AllObjects* definiert sind, die in Abbildung 6.1 nicht dargestellt wird.

Die zentralen Aspekte des Informationsmodells stellen die Klassen Service und Application dar. Ein Service erbringt Leistungen, die von einem Benutzer oder einer Application verwendet werden können. Er kann auf anderen Services und/oder Components basieren. Eine Component kann mittels unterschiedlicher Typen klassifiziert werden: Hardware-, Software-, ServiceLevel-Components und Deliverables. Components können ebenso wie Services auf anderen Components aufbauen.

Eine Application ist eine versionierte Software, die festgelegte Funktionalität für einen Process bereitstellt und so einzelne Prozessschritte unterstützt. Prozesse unterteilen sich in Management-, Geschäfts- und Supportprozesse. Eine Application kann von einer *Organization*, die der internen Strukturierung des Unternehmens in logische Einheiten dient, verwendet werden. Eine Application ist als *DeployedApplication* an einer physikalischen *Location* installiert. Locations werden in physikalische (Berlin, München, etc.) und virtuelle Locations (Internet, Intranet) unterschieden.

6.1.2 Anforderungsanalyse anhand von Softwarekartentypen

Die Konzeption eines Teilinformationsmodells zur Softwarekartographie basiert auf der Anforderungsanalyse von zwei existierenden und häufig verwendeten Softwarekartentypen. Im Folgenden werden das *Service Struktur Diagramm* und die *Prozessunterstützungskarte* vorgestellt und ihre Anforderungen an ein Informationsmodell zur automatischen Generierung von Softwarekarten untersucht.

Service Struktur Diagramm

Das *Service Struktur Diagramm* wird von einem Projektpartner verwendet, um die Komponenten, aus denen ein Service besteht und seine Abhängigkeit von anderen Services, zu visualisieren. Der Aufbau des Kartentyps zeichnet sich durch die Verortung von Informationsobjekten (Komponenten) anhand von logischen Einheiten (Services) aus.

Dabei bildet immer ein einzelner Service den Hauptaspekt der Softwarekarte. Alle Komponenten, auf denen der Service basiert, werden in einen Cluster, der den Service darstellt zusammengefasst. Basiert der Service auf der Funktionalität anderer Services so werden diese ebenfalls am Rand der Softwarekarte dargestellt und mit dem zentralen Service verbunden. Eine Klassifizierung der dargestellten Komponenten in Hardware-, Software-, Servicelevelkomponenten und Deliverables, kann durch die Verwendung einer Farbcodierung erreicht werden [Ers05].

Abbildung 6.2 zeigt exemplarisch den Aufbau eines Service Struktur Diagramms.

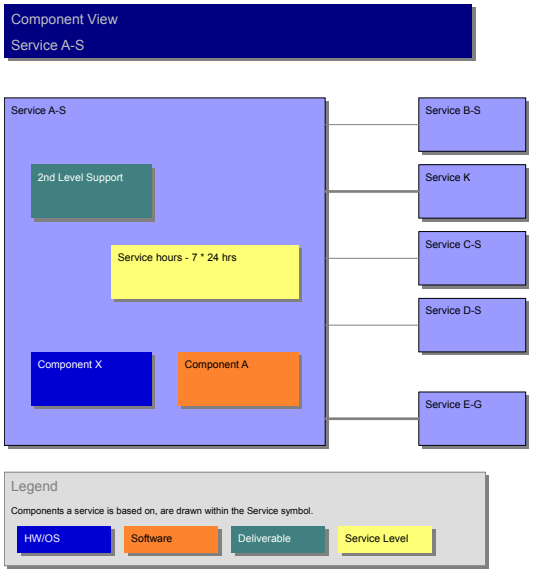


Abbildung 6.2: Service Struktur Diagramm

Mit Hilfe des Service Struktur Diagramms können unterschiedliche Fragestellungen über die Funktionalität und den Aufbau eines Services beantwortet werden. Exemplarische Fragestellungen, die durch das Service Struktur Diagramm kommuniziert werden können sind:

- Auf welchen Komponenten basiert ein Service?
- Welche anderen Services verwendet ein Service?
- Welche Service Level Agreements[1] werden von einem Service erfüllt?

Die folgende Abbildung 6.3 visualisiert ein geeignetes Informationsmodell zur automatischen Generierung von Softwarekarten vom Typ Service Struktur Diagramm.

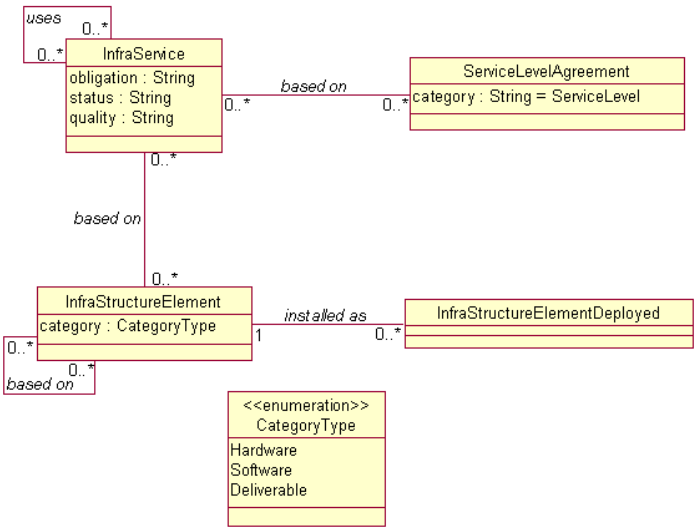


Abbildung 6.3: Anforderung an ein Informationsmodell zur Generierung eines Service Struktur Diagramms

Die zentralen Klassen des Informationsmodells bildet die Klasse *InfraService*[2] und die Klasse *InfraStructureElement*. Ein InfraService kann andere InfraServices verwenden, um seine Funktionalität zu erbringen und verschiedene *ServiceLevelAgreements* zu erfüllen. Ein InfraService basiert auf einer beliebigen Anzahl von InfraStructureElements.

Ein InfraStructureElement ist ein deployable Element, das beliebig oft als *InfraStructureElementDeployed* deployed werden kann. Ein InfraStructureElement kann auf einer

[1] dt. Dienstgütevereinbarung, Leistungsvertrag
[2] Infra wird als Abkürzung für Infrastructure verwendet.

beliebigen Anzahl von InfraStructureElementen basieren und ist genau einem *CategoryType* zugeordnet, der seine Klassifizierung in Hardware-, Softwarekomponenten und Deliverables vornimmt.

Prozessunterstützungskarte

Einen weiteren häufig verwendeten Softwarekartentyp stellt die Prozessunterstützungskarte dar, die Anwendungssysteme anhand, der von ihnen unterstützten (betrieblichen) Prozesse verortet. Für die Darstellung auf der Prozessunterstützungskarte wird ein Prozess als Wertschöpfungskette angesehen und legt typischerweise die x-Achse der Softwarekarte fest. Die horizontale Ausdehnung eines Rechtecks, das eine Anwendung repräsentiert, visualisiert hierbei die Unterstützung der Anwendung für bestimmte Prozesse.

Steht die Nutzung einer Anwendung durch eine bestimmte Organisationseinheit neben der Unterstützung eines Prozessschritts, im Vordergrund der Analyse, so werden die betroffenen Organisationseinheiten auf der y-Achse angeordnet. Die Position und vertikale Ausdehnung eines Rechtecks, visualisiert hierbei die Nutzung der Anwendung in einer bestimmten Organisationseinheit [LMW05b, MW04b].

Die folgende Abbildung 6.4 zeigt eine exemplarische Darstellung einer Prozessunterstützungskarte.

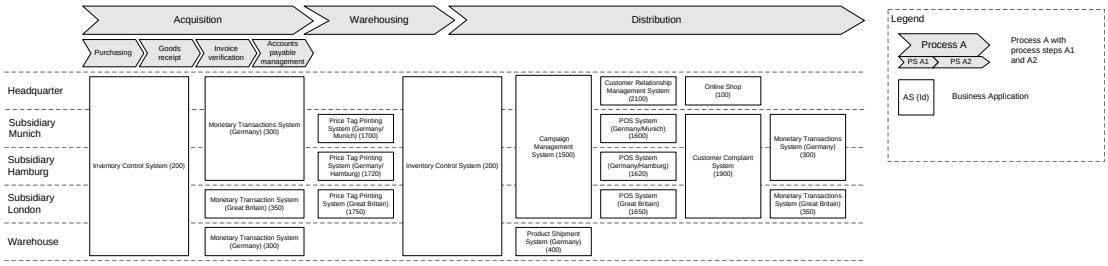


Abbildung 6.4: Prozessunterstützungskarte nach [seb05a]

Mit Hilfe der Prozessunterstützungskarte können unterschiedliche Fragestellungen über die Prozessunterstützung durch Anwendungen beantwortet werden. Exemplarische Fragestellungen, die durch die Verwendung einer Prozessunterstützungskarte kommuniziert werden können sind:

- Von welchen Anwendungen wird ein bestimmter Prozessschritt unterstützt?
- Von welcher Organisationseinheit wird eine bestimmte Anwendung, die einen bestimmten Prozessschritt unterstützt, verwendet?

Ein Informationsmodell, das die Beantwortung dieser Fragestellungen und die automatische Generierung von Softwarekarten vom Typ Prozessunterstützungskarte ermöglicht, zeigt Abbildung 6.5.

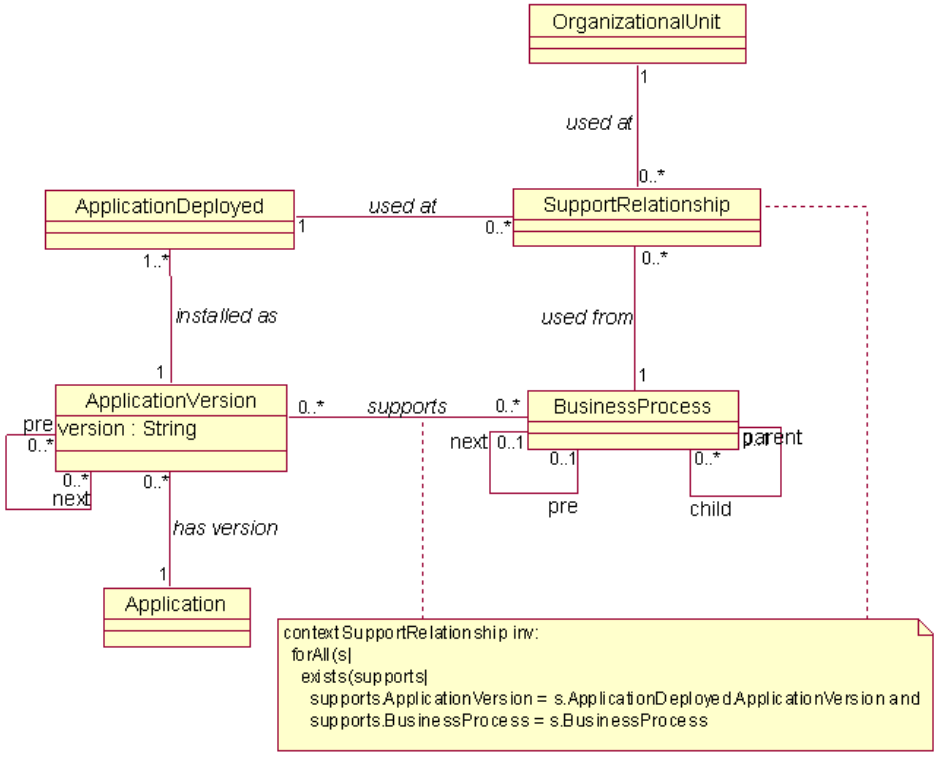


Abbildung 6.5: Anforderungen an ein Informationsmodell zur Generierung einer Prozessunterstützungskarte

Einen zentralen Aspekt des Informationsmodells bildet die Klasse *SupportRelationship*. Sie ermöglicht eine eindeutige Zuordnung, des von einer *ApplicationDeployed* unterstützten *BusinessProcesses* zu der *OrganizationalUnit*, die diese Unterstützung verwendet.

Den zweiten zentrale Aspekt des Informationsmodells neben der Klasse SupportRelationship stellt die Klasse *Application* dar. Einer Application können eine beliebige Anzahl von *ApplicationVersions* zugeordnet werden, die einer Vorgänger-Nachfolger-Ordnung unterliegen können. Eine ApplicationVersion ist ein einsetzbares Element, das beliebig oft als ApplicationDeployed verwendet werden kann.

Ein BusinessProcess verwendet die Funktionalität einer beliebigen Anzahl von ApplicationVersions. BusinessProcesses können einen Vorgänger und einen Nachfolger besitzten und zusätzlich einer hierarchischen Ordnung unterliegen.

6.1.3 Konsolidierung der Teilmodelle zu einem Zielmetamodell

Anhand der, in den vorangegangenen Abschnitten durchgeführten Anforderungsanalyse wird in diesem Abschnitt durch Konsolidierung des Teilmodells des Service Struktur Diagramms mit dem Teilmodell der Prozessunterstützungskarte ein geeignetes Informationsmodell zur automatischen Generierung von Softwarekarten konzipiert.

Die folgende Abbildung 6.6 visualisiert das durch die Konsolidierung entstandene Informationsmodell, welches durch weitere Modellelementa su dem Forschungsprojekt Softwarekartographie angereichert wurde.

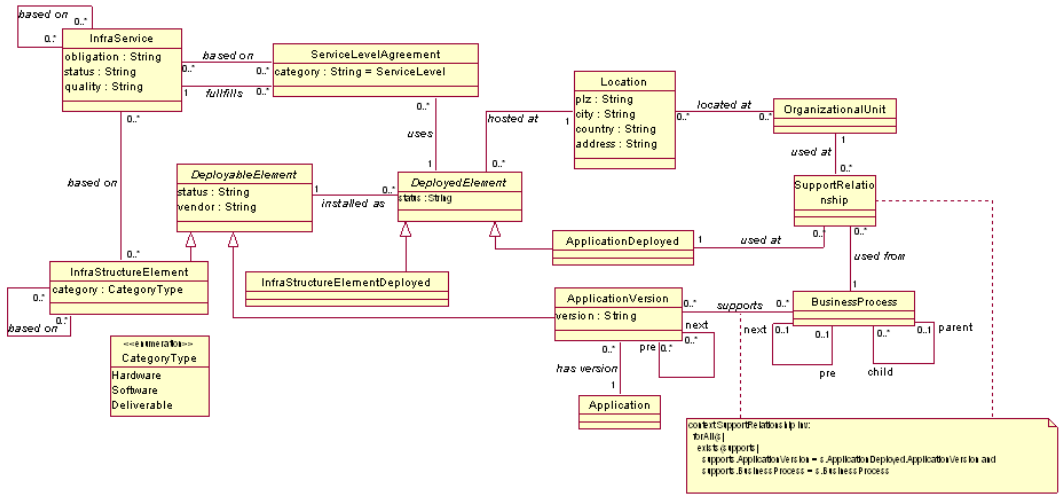


Abbildung 6.6: Informationsmodell zur automatischen Generierung von Softwarekarten

Bei der Konsolidierung der Teilmodelle zu einem Informationsmodell bietet sich die Einführung von zwei neuen abstrakten Superklassen an. Zusätzlich wird noch die Klasse *Location* eingeführt, die einen physikalischen oder virtuellen Standort beschreibt. Die neu eingeführte Superklasse *DeployableElement* repräsentiert einsetzbare Elemente, während die abstrakte Superklasse *DeployedElement* Elemente beschreibt, die an genau einer Location in Betrieb sind. Zu der Klasse der DeployableElements gehört das InfraStructureElement und die ApplicationVersion. Die Klassen ApplicationDeployed und InfraStructureElementDeployed sind Subklassen der Klasse DeployedElement. Nicht in Abbildung 6.6 dargestellt ist die abstrakte Superklasse *ModelElement*, die die für alle Klassen obligatorischen Attribute *name* und *description* vom Typ String enthält.

6.2 Exemplarische Transformation zwischen Informationsmodellen

Aufbauend auf den in den vorangegangenen Abschnitten vorgestellten, Quellmetamodell (siehe Abbildung 6.1) und Zielmetamodell (siehe Abbildung 6.6) soll in diesem Abschnitt eine exemplarische Transformation durchgeführt werden, die die Anwendbarkeit, der in den vorangegangenen Kapiteln entwickelten Konzepte, validiert.

Als Spezifikationssprache für die Transformation wird die in Abschnitt 5.2.2 vorgestellte Atlas Transformation Language verwendet, für die eine Werkzeugunterstützung in Form eines Plug-Ins für Eclipse existiert. Basis der Transformation bilden die ecore-Modelle des Quell- und Zielmetamodells (siehe Anhang C).

Im folgenden Abschnitt 6.2.1 werden die Regeln entwickelt, die die Transformation eines Quellmodells, das eine Instanz des in Abschnitt 6.1.1 vorgestellten Quellmetamodells darstellt, in ein Zielmodell, das konform zu dem in Abschnitt 6.1.3 definierten Zielmetamodells ist, ermöglichen. Abschnitt 6.2.2 überprüft das Ausgabemodell der Transformation auf seine Metamodellkonformität zu dem Zielmetamodell.

6.2.1 Transformationsregeln

Die folgenden Transformationsregeln werden für die Transformation eines Quellmodells in ein Zielmodell benötigt:

- Für jede Instanz der Klasse *Service* erzeuge eine Instanz der Klasse *InfraService*.
- Für jeden Link zwischen Instanzen der Klasse *Service* verbinde die entsprechenden Instanzen der Klasse *InfraService*.
- Für jede Instanz c der Klasse *Component* mit c.Category != ServiceLevel:
  - Erzeuge eine Instanz der Klasse *InfraStructureElement*.
  - Für jeden Link zwischen Instanzen der Klasse *Component* verbinde die entsprechenden Instanzen der Klasse *InfraStructureElement*.
  - Erzeuge eine Instanz der Klasse *InfraStructureElementDeployed*.
  - Erzeuge einen Link zwischen den neu erzeugten Instanzen.
- Für jede Instanz c der Klasse *Component* mit c.Category = ServiceLevel erzeuge eine Instanz s der Klasse *ServiceLevelAgreement* mit s.category = ServiceLevel.
- Für jeden Link zwischen einer Instanz der Klasse *Service* und einer Instanz c von *Component* erzeuge eine entsprechende Verbindung zwischen einer Instanz der Klasse *InfraService* und

  - der entsprechenden Instanz von *ServiceLevelAgreement*, falls c.Category = ServiceLevel oder
  - der entsprechenden Instanz von *InfraStructureElement*, falls c.Categroy != ServiceLevel.

- Für jede Instanz der Klasse *DeployedApplication* erzeuge eine Instanz der Klasse *ApplicationDeployed*.
- Für jede Instanz der Klasse *Location* erzeuge eine Instanz der Klasse *Location*.
- Für jeden Link zwischen einer Instanz der Klasse *DeployedApplication* und *Location* erzeuge eine entsprechende Verbindung zwischen Instanzen der Klassen *ApplicationDeployed* und *Location*.
- Für jede Instanz der Klasse *Application*:
  - Erzeuge eine Instanz av der Klasse *ApplicationVersion* mit av.version = "xxx".
  - Erzeuge eine Instanz der Klasse *Application*.
  - Erzeuge einen Link zwischen den neu erzeugten Instanzen.
- Für jeden Link zwischen einer Instanz der Klasse *DeployedApplication* und *Application* erzeuge einen entsprechenden Link zwischen Instanzen der Klassen *ApplicationDeployed* und *ApplicationVersion*.
- Für jede Instanz der Klasse *Process* erzeuge eine Instanz der Klasse *BusinessProcess*.
- Für jeden Link zwischen Instanzen der Klassen *Application* und *Process* erzeuge eine entsprechende Verbindung zwischen Instanzen der Klassen *ApplicationVersion* und *BusinessProcess*.
- Für jede Instanz der Klasse *Organization* erzeuge eine Instanz der Klasse *OrganizationalUnit*.

Die vollständige Transformationsspezifikation für die Transformation, der vorgestellten Informationsmodelle, besteht aus acht Regeln und kann im Anhang C eingesehen werden.

6.2.2 Metamodellkonformität des entstandenen Ausgabemodells

Im folgenden Abschnitt soll das, aus der Transformation entstandene Zielmodell, auf seine Metamodellkonformität überprüft werden. Die folgende Abbildung 6.7 visualisert das Ausgabemodell, das mit Hilfe der in Abschnitt 6.2.1 definierten Transformationsregeln erzeugt werden kann[3].

[3]Blau dargestellte Klassen oder Assoziationen können nicht automatisch generiert werden.

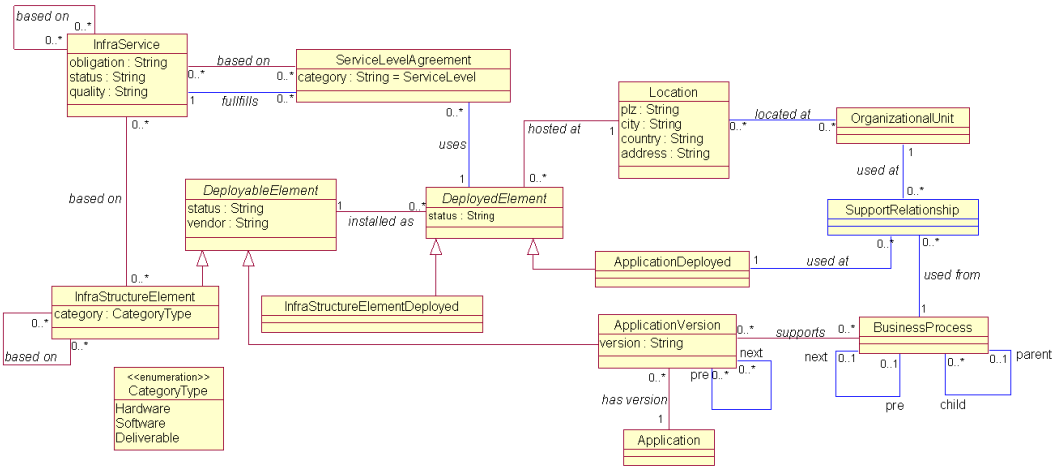


Abbildung 6.7: Ausgabemodell der Transformation

Das erzeugte Ausgabemodell stellt keine gültige Instanz des Zielmetamodells dar. Um eine Metamodellkonformität zu erreichen, müsste das Erzeugen von zwei weiteren Verbindungen durch die Transformationsspezifikation unterstützt werden:

- Jedes *InfraStructureElementDeployed* muss einen Link zu genau einer *Location* besitzen, an der es betrieben wird.
- Jedes *ServiceLevelAgreement* muss einen Link zu genau einem *DeployedElement* besitzen, das seine Funktionalität nutzt.

Die automatische Generierung dieser Beziehungsobjekte kann von der Transformationsspezifikation nicht unterstützt werden, weil die entsprechenden Informationen nicht im Quellmodell enthalten sind.

Alle anderen Assoziationen, die nicht durch die Transformationsspezifikation erzeugt werden können (in Abbildung 6.7 blau dargestellt), sind für die Metamodellkonformität des Ausgabemodells nicht von Bedeutung, da ihre Multiplizitäten eine Existenz nicht erzwingen.

6.3 Zusammenfassung der Transformationsergebnisse

Die Mächtigkeit der Quell- und Zielmetamodelle einer Transformation bestimmt, ob durch die Transformation ein Informationsgewinn oder -verlust stattfindet. Eine Klas-

sifikation von Transformationen anhand der Mächtigkeit der Quell- und Zielmetamodelle bietet Blaha et al. [BP96].

- **Equivalence Transformation (Gleichwertige Transformation):** Bei einer gleichwertigen Transformation besitzen die Quell- und Zielmetamodelle die gleiche Mächtigkeit. Instanzen des Quellmetamodells lassen sich korrespondierenden Instanzen des Zielmetamodells exakt zuordnen und vice versa. Eine gleichwertige Transformation kann nebensächliche Informationen (wie z.B. Assoziationsnamen oder Rollennamen) verlieren oder hinzugewinnen, die Äquivalenz der Mächtigkeit der Modelle bleibt aber erhalten.
- **Information-losing Transformation (Informationsverkürzende Transformation):** Bei einer informationsverkürzenden Transformation besitzen die Quellmetamodelle eine höhere Mächtigkeit als die Zielmetamodelle. Alle Instanzen des Zielmetamodells lassen sich durch Instanzen des Quellmetamodells beschreiben, im Gegensatz dazu können einige Instanzen des Quellmetamodells nicht durch Instanzen des Zielmetamodells beschrieben werden. Bei einer informationsverkürzenden werden Informationen, die im Quellmodell vorhanden sind, im Zielmodell weggelassen.
- **Information-gaining Transformation (Informationsergänzende Transformation):** Bei einer informationsgewinnenden Transformation besitzten die Zielmetamodelle eine höhere Mächtigkeit als die Quellmetamodelle. Alle Instanzen des Quellmetamodells lassen sich durch Instanzen des Zielmetamodells beschreiben, im Gegensatz dazu können einige Instanzen des Zielmetamodells nicht durch Instanzen des Quellmetamodells beschrieben werden. Bei der informationsergänzenden Transformationsspezifikation müssen zusätzliche Informationen manuell hinzugefügt werden.

Die in Abschnitt 6.2 vorgestellte Transformation zwischen semantischen Informationsmodellen stellt eine Information-gaining Transformation dar. Das größte Problem bei dieser Art von Transformationen stellt der Umgang mit fehlender Information dar. Informationen, die im Quellmodell nicht enthalten sind, im Zielmodell aber für die Metamodellkonformität benötigt werden, verlangen eine spezielle Beachtung.

Für den Umgang mit fehlenden Informationen wurden zwei Lösungsansätze gewählt:

- Die explizite Unterscheidung von Anwendungsversionen und Anwendungen wurde innerhalb des Quellmetamodells nicht unterstützt. Um diese Unterscheidung im Zielmetamodell treffen zu können, wurde in der Transformationsspezifikation zu jedem Anwendungsobjekt ein Anwendungsversionsobjekt erstellt. Die automatisch erstellten Beziehungen zwischen Anwendungsversionen und Anwendungen müssen nachträglich manuell verbessert und eventuell entstandene Anwendungsobjektduplikate gelöscht werden, um eine korrekte Zuordnung zwischen Anwendungen, Anwendungsversionen und deren

Nachfolger-/Vorgänger-Ordnung zu erhalten. Durch die automatische Generierung von Dummyobjekten wird im Kontext der Anwendungen die Metamodellkonformität des Ausgabemodells nicht gestört und ein maximaler Transport von Informationen ermöglicht.

- Die eindeutige Zuordnung, der Unterstützung einer eingesetzten Anwendung für einen Prozessschritt und der Nutzung dieser Unterstützung durch eine Organisationseinheit, kann mit Hilfe der im Quellmetamodell enthaltenen Informationen nicht getroffen werden. Durch die automatische Generierung eines Dummy-Relationship-Objektes im Ausgabemodell, das diese eindeutige Zuordnung ermöglicht, wäre die Metamodellkonformität des Ausgabemodells nicht mehr gegeben, da die Verbindungen des Dummy-Objektes zu der entsprechenen Organisationseinheit, der eingesetzten Anwendung und dem entsprechenden Geschäftsprozess nicht automatisch erzeugt werden könnten. Um die Metamodellkonformtität des Ausgabemodells zu gewährleisten ist ein Weglassen dieses Informationsobjektes und eine manuelle Nachbearbeitung des Ausgabemodells sinnvoll.

Prinzipiell sind bei der Transformation von Informationsmodellen weitere Lösungsansätze denkbar, die sowohl einen verkürzenden Charakter als auch einen *ergänzenden* Charakter haben.

7 Zusammenfassung und Ausblick

Abschließend sollen in diesem Kapitel die Schwerpunkte und erreichten Ziele dieser Arbeit zusammengefasst und den Zielsetzungen aus der Einleitung gegenübergestellt werden (Abschnitt 7.1). Ein Ausblick auf zukünftige Arbeiten und mögliche Anknüpfungspunkte in Abschnitt 7.2 rundet die Arbeit ab.

7.1 Zusammenfassung

Das Kapitel 1 diente der Motivation und Einführung in die Aufgabenstellung der Arbeit. Als Zielsetzungen wurden die folgenden Problemstellungen definiert:

- Identifizierung und Konsolidierung einer Obermenge, der zur Modellierung von Informationsmodellen zum Enterprise Architecture Management verwendeten Konzepte
- Verifikation des MOF-Ansatzes als Metamodell zur Modellierung von Informationsmodellen und als Basis für Modell-basierte Transformationen
- Analyse existierender Spezifikationssprachen zur Transformation von Informationsmodellen

In Kapitel 2 wurde eine Einbettung der Arbeit in ihr thematisches Umfeld vorgenommen, indem Einführungen in die Themengebiete Enterprise Architecture und die Softwarekartographie vorgenommen und zueinander in Beziehung gesetzt wurden. Eine Vorstellung von Modellen und Modellierungssprachen, die für das Verständnis der weiteren Arbeit benötigt wurden, erfolgte in Kapitel 3. Die eingeführten Themengebiete umfassten den Modellbegriff im Allgemeinen und in der Softwarekartographie, sowie existierende Modellierungssprachen und die Model Driven Architecture.

Die erste Zielsetzung dieser Arbeit, die Identifizierung und Konsolidierung einer Obermenge, der zur Modellierung von Informationsmodellen zum Enterprise Architecture Management verwendeten Konzepte, wurde in Kapitel 4 vorgenommen. Zu diesem Zweck wurde einleitend eine Einordnung, der von den Modellierungssprachen EMOF, CMOF und UML unterstützten Modellierungskonzepte, vorgenommen und darauf

aufbauend existierende Informationsmodelle von mittelständischen und großen Unternehmen, sowie von Werkzeugherstellern, auf die, von ihnen verwendeten bzw. unterstützten Konzepte, analysiert. Zusätzlich wurden standardisierte Ansätze aus dem Bereich des IT-Infrastruktur- und IT-Portfoliomanagement herangezogen, um eine umfassende Analyse zu ermöglichen. Ein Abgleich mit der Meta-Objekt-Facility rundet das Kapitel ab und erfüllt Zielsetzung zwei, durch die Verifikation des CMOF-Ansatzes als Metamodell zur Modellierung von Informationsmodellen und als Basis für Modellbasierte Transformationen.

Der Analyse existierender Informationsmodelle schließt sich die Analyse bestehender Spezifikationssprachen zur Modell-basierten Transformation, als dritte Zielsetzung, in Kapitel 5 an. Die Klassifikation anhand unterschiedlicher Eigenschaften von Transformationssprachen und die Unterstützung, der in Kapitel 4 definierten Modellierungskonzepte, bilden die Grundlage der Analyse. Die analysierten Spezifikationssprachen zur Modelltransformation umfassen das Eclipse Modeling Framework (EMF), die Atlas Transformation Language (ATL), die Model Transformation Language (MTL) und die Bidirectional Object Oriented Transformation Language (BOTL). Ein Resümee über die Eignung der Spezifikationssprachen, welches die Transformationssprache Atlas Transformation Language als geeignet hervorhebt, zur Transformation von Informationsmodellen runden das Kapitel ab.

Kapitel 6 stellt eine exemplarische Transformation von Informationsmodellen mit Hilfe der Atlas Transformation Language vor. Anhand dieser Transformation wurden die, die in den vorangegangenen Kapiteln entwickelten Konzepte auf ihre Eignung überprüft.

Abschließend lässt sich zusammenfassen, dass durch die Analyse existierender Informationsmodelle in Kapitel 4 und die Untersuchung der, von Spezifikationssprachen unterstützten Modellierungskonzepte in Kapitel 5, die Korrektheit des CMOF-Ansatzes als Metamodell von Informationsmodellen und als Basis für die Modell-basierte Transformation bestätigt werden kann.

7.2 Ausblick

Die in dieser Arbeit vorgestellten Konzepte und Einblicke in die Themengebiete der Modellierung und Modell-basierten Transformation ergeben eine Vielzahl von Perspektiven für die Wiederverwendung und Weiterentwicklung.

Im Bereich der Informationsmodellierung selbst können die, in dieser Arbeit entwickelten Ergebnisse, als Leitfaden für die Verwendung von Modellierungskonzepten zur Entwicklung von ”best practice” Informationsmodellen verwendet werden. Durch die Analyse existierender Informationsmodelle aus unterschiedlichen Anwendungsbereichen - Anwender, Hersteller und Forschung - ergibt sich ein repräsentativer Querschnitt der verwendeten und unterstützten Konzepte.

Ein weiteres interessantes Themengebiet, das sich für eine Wiederverwendung, der in dieser Arbeit entwickelten Konzepte und untersuchten Ansätze anbietet, ist die Konzeption eines Werkzeugs zur Softwarekartographie. Der immense Aufwand, den die manuelle Erstellung von Softwarekarten in sich birgt, kann durch eine automatische Kartengenerierung minimiert werden. In der aktuellen Version des Werkzeuges wird die Transformation zwischen dem semantischen und dem symbolischen Modell duch eine Java-Implementierung vorgenommen. Die Ersetzung dieser Implementierung durch eine Transformationsspezifikation in einer deklarativen Sprache ermöglicht eine einfache Anpassung der Transformationsregeln und der durch sie erzeugten Karten, an die jeweiligen Bedürfnisse der Benutzer.

Im Kontext des Softwarekartographiewerkzeuges sind weitere Eigenschaften von Transformationssprachen interessant, die in dieser Arbeit nicht näher untersucht wurden. So müssen bei der automatischen Generierung von Softwarekarten sehr große Mengen an Informationen verarbeitet werden. Die Fähigkeit mit einer großen Anzahl von Informationsobjekten noch effektiv und effizient arbeiten zu können spielt in diesem Zusammenhang eine wichtige Rolle. Die weiterführende Analyse von Transformationssprachen stellt eine über diese Arbeit hinausführende Aufgabe dar.

Der Bereich der Modell-basierten Transformationen ist ein Gebiet das aktuell ständigen Neuerungen unterworfen ist und eine Vielzahl unterschiedlicher Ansätze bietet. Die Ergebnisse dieser Arbeit und die Ausblicke zeigen die vielfältige Verwendungsmöglichkeit von MOF-basierten Transformationen zum Management von Anwendungslandschaften.

A Analyse existierender und relevanter Informationsmodelle

Im Rahmen dieser Arbeit wurden Informationsmodelle von Unternehmen, Werkzeugen und Standardisierungsgremien für das Enterprise Architecture Management auf die, von ihnen verwendeten Metamodellierungskonzepte untersucht. Die Analyse erfolgte in 4 Teilschritten:

- **Modellierungssprachen**: In einem ersten Schritt wurden die Modellierungssprachen Essential MOF (EMOF), Complete MOF (CMOF) und die Unified Modeling Language (UML) auf die in ihnen definierten Modellierungskonzepte untersucht.
- **Informationsmodelle von Unternehmen:** In einem zweiten Schritt wurden existierende Informationsmodelle von mittelständischen und großen Unternehmen (U_1, U_2, U_3 und U_4) auf die, für ihre Modellierung verwendeten Modellierungskonzepte analysiert.
- **Informationsmodelle existierender Werkzeuge:** In einem dritten Schritt erfolgte eine Analyse der Informationsmodelle existierender Werkzeuge. Die untersuchten Werkzeuge umfassen den Corporate Modeler (CM) der Firma Casewise, das Tool MEGA der MEGA SA international und den System Architect (SA) von Telelogic.
- **Standardisierte Informtionsmodelle:** In einem letzen Schritt wurden standardisierte Informationsmodelle auf ihre Verwendung von Modellierungskonzepten analysiert. In die Analyse eingeschlossen waren das Common Information Model (CIM) der DMTF [DMT05] und das IT Portfolio Management Facility (ITPMF) der DMTF [OMG04a].

Die Tabelle A liefert eine Übersicht über die in Kapitel 4 gewonnenen Analyseergebnisse.

	EMOF	CMOF	UML	U_1	U_2	U_3	U_4	CM	MEGA	SA	CIM	ITPMF
Klassen	√	√	√	√	√	√	√	√	√	√	√	√
Abstrakte Klassen	√	√	√	√	√	√	√	√[a]			√	√
Attribute	√	√	√	√	√	√	√	√	√	√	√	√
vordef. Datentypen	√	√	√	√	√	√	√	√	√	√	√	√
erw. Datentypen		√	√	√	√	√	√				√	√
Pakete	√	√	√	√	√	√			√[b]		√	√
Paketschachtelung	√	√	√		√	√					√	√
Paketimport		√	√	√	√	√					√	√
Paketvereinigung		√	√									
Binäre Assoziationen	√[c]	√	√	√	√	√	√	√	√	√	√	√
n-äre Assoziationen			√									
Assoziationsnamen		√	√	√	√	√	√	√	√	√	√	√
Rollennamen	√[c]	√	√	√	√	√		√	√	√	√	√
Multiplizitäten	√	√	√	√	√	√			√	√	√	√
Navigierbarkeit		√	√	√	√							
Vererbung	√	√	√	√	√	√	√		√		√	√
Aggregation	√[c]	√	√	√	√	√			√		√	√
Komposition	√[c]	√	√			√			√		√	√
Assoziationsklassen			√	√	√			√[d]		√[d]		
Bedingungen		√	√		√							√
Stereotypen			√									√
Trigger/Event			√								√	

Tabelle A.1: Tabellarischer Überblick der von Modellierungssprachen unterstützen Konzepte

[a] Klassifizierung eines Objekt Typs als *Template* ist möglich.
[b] Die Definition von Namensräumen wird unterstützt.
[c] Konzept kann durch "verschränkte" Attribute nachgebildet werden.
[d] Es werden nur attributierte Assoziationen und kein vollständiges Assoziationsklassenkonzept unterstützt.

B Analyse existierender Transformationssprachen

Die in Kapitel 6 vorgestellten Spezifikationssprachen zur Modelltransformation sollen in diesem Kapitel anhand des folgenden einfachen Beispiels näher untersucht werden. Ausgangspunkt für die exemplarischen Transformationen bildet das folgende, in Abbildung B.1 dargestellte, Metamodell, das den Aufbau einer *Company* aus *OrganizationalUnits* beschreibt. Wie in Abbildung B.1 dargestellt, besteht eine *Company* aus einen Namen während eine *OrganizationalUnit* einen Namen und eine Anzahl an Mitarbeitern, die sie beschäftigt, als Attribute besitzt.

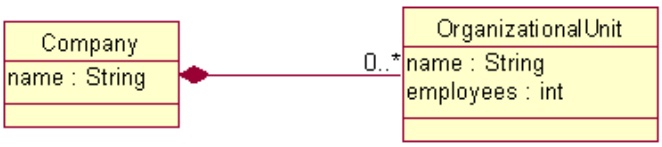


Abbildung B.1: Quellmetamodell einer Company, die aus einer beliebigen Anzahl von OrganizationalUnits besteht

Dieses Metamodell ist Ausgangspunkt für die Analyse der Spezifikationssprachen zur Modelltransformation. Im Rahmen der Analyse von Codetransformationen ist die Angabe eines Quellmetamodells ausreichend, da das Zielmetamodell durch die Zielsprache (im folgenden Java) bereits vorgegeben ist. Im Gegensatz dazu, wird für die Analyse von Modelltransformationssprachen zusätzlich noch die Angabe eines Zielmetamodells benötigt. Das folgende in Abbildung B.2 dargestellte Metamodell, soll als Zielmetamodell für die Analyse der Modelltransformationssprachen dienen.

Das Zielmetamodell modelliert eine *Enterprise*, die aus einem Namen und einer Anzahl von Mitarbeiter besteht. Das Zielmodell entsteht aus dem Quellmodell, indem aus jeder *Company*-Instanz eine *Enterprise*-Instanz mit dem gleichen Namen gebildet wird. Die Anzahl der Mitarbeiter einer *Enterprise*-Instanz entspricht der Summe der Mitarbeiter aller, zu einer *Company*-Instanz, gehörenden *OrganizationalUnit*.

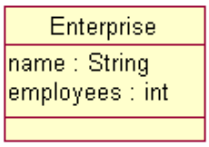


Abbildung B.2: Zielmetamodell einer Enterprise

In den folgenden Abschnitten sollen exemplarische Transformationen, mit Hilfe der in diesem Abschnitt vorgestellten Modelle, eine Analyse der unterschiedlichen Spezifikationssprachen vervollständigen.

B.1 Exemplarische Codetransformation mit dem Eclipse Modeling Framework (EMF)

Ein einfaches Beispiel soll die Funktionsweise des *Eclipse Modeling Framework* näher erläutern. Ausgangspunkt für die Transformation bildet das im vorigen Abschnitt vorgestellte UML-Diagramm aus Abbildung B.1, das mit Rational Rose erstellt und in EMF exportiert wird. Nach der Importierung erzeugt EMF aus der Definition des Rose-Modells ein EMF-Modell, das eine Instanz des Metamodells ecore darstellt (siehe Listing B.1).

Listing B.1: Ecore Modell des Quellmetamodells

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi
    ="http://www.w3.org/2001/XMLSchema-instance" xmlns:ecore="http://www
    .eclipse.org/emf/2002/Ecore" name="organizationModel" nsURI="http:
    ///organizationModel.ecore" nsPrefix="organizationModel">
3   <eClassifiers xsi:type="ecore:EClass" name="OrganizationalUnit">
4     <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
        EString" defaultValueLiteral=""/>
5     <eStructuralFeatures xsi:type="ecore:EAttribute" name="employees"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf
        /2002/Ecore#//EInt"/>
6     <eStructuralFeatures xsi:type="ecore:EReference" name="company" lowerBound=
        "1" eType="#//Company" eOpposite="#//Company/organizationalunit"/>
7   </eClassifiers>
8   <eClassifiers xsi:type="ecore:EClass" name="Company">
```

```
9     <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" lowerBound="1"
          eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
          EString" defaultValueLiteral=""/>
10    <eStructuralFeatures xsi:type="ecore:EReference" name="organizationalunit
          " upperBound="-1" eType="#//OrganizationalUnit" containment="true"
          eOpposite="#//OrganizationalUnit/company"/>
11  </eClassifiers>
12 </ecore:EPackage>
```

Aus diesem EMF-Modell lassen sich automatisch die entsprechenden Java-Klassen und Interfaces generieren. Listing B.2 zeigt die entsprechenden Java-Klassen und Interfaces[1].

Listing B.2: Java-Klassen und Interfaces

```
1  package organizationModel;
2
3  import org.eclipse.emf.common.util.EList;
4  import org.eclipse.emf.ecore.EObject;
5
6  /**
7   * @model
8   * @generated
9   */
10 public interface Company extends EObject {
11         /**
12          * @model default="" required="true"
13          * @generated
14          */
15         String getName();
16
17         /**
18          * @generated
19          */
20         void setName(String value);
21
22          * @model type="organizationModel.OrganizationalUnit"
                opposite="company" containment="true"
23          * @generated
24          */
25         EList getOrganizationalunit();
26 }
27
28 package organizationModel;
```

[1] aus Platzgründen wurden die meisten Kommentare entfernt

```
29 import org.eclipse.emf.ecore.EObject;
30
31 /**
32  * @model
33  * @generated
34  */
35 public interface OrganizationalUnit extends EObject {
36         /**
37          * @model default="" required="true"
38          * @generated
39          */
40         String getName();
41
42         /**
43          * @generated
44          */
45         void setName(String value);
46
47         /**
48          * @model required="true"
49          * @generated
50          */
51         int getEmployees();
52
53         /**
54          * @generated
55          */
56         void setEmployees(int value);
57
58         /**
59          * @model opposite="organizationalunit" required="true"
60          * @generated
61          */
62         Company getCompany();
63
64         /**
65          * @generated
66          */
67         void setCompany(Company value);
68 }
```

Die vierte und letzte Darstellung, die sich mit EMF generieren lässt, oder als Ausgangspunkt für die Transformationen genutzt werden kann, bildet die, in folgendem Listing B.3 dargestellte XML Schema Definition.

Listing B.3: XML Schema des Quellmetamodells

```
<xsd:schema targetNamespace="http:///organization.ecore"
    xmlns="http:///organization.ecore"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="OrganizationalUnit">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="location" type="Location"
        minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:complexType name="Location">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="address" type="xsd:string"/>
      <xsd:element name="country" type="xsd:string"/>
      <xsd:element name="organizationalunit" type="OrganizationalUnit"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

B.2 Exemplarische Transformation eines Modells mit der Atlas Transformation Language (ATL)

Im folgenden soll eine einfache Transformation mit Hilfe der *Atlas Transformation Language* näher erläutert werden. Für eine Transformation des Quellmodells in das Zielmodell müssen folgende Regeln spezifiziert werden:

Für jede *Company*-Instanz des Quellmodells muss eine neue Instanz der Klasse *Enterprise* im Zielmodell erzeugt werden. Das Attribut *name* der *Enterprise*-Instanz wird auf den Wert des *Company.name*-Attributs gesetzt. Die Anzahl von Mitarbeitern einer *Enterprise* ergibt sich aus der Summe der Anzahl der Mitarbeiter der einzelnen *OrganizationalUnits*.

Für die Transformation des Quellmodells in das Zielmodell wird eine Regel benötigt, siehe Listing B.4. Diese Regel verwendet einen *Helper*, um die Summe der Mitarbeiter einer *Enterprise* zu ermitteln.

Listing B.4: ATL Modul für die Transformation Company2Enterprise

```
module Company2Enterprise;
create OUT : Enterprise from IN : Company;

helper context Company!Company def : getSumEmployees() : Integer =
```

```
5           self.organizationalunits->collect(o|o.employees).sum()
6   ;
7
8   rule Company2Enterprise {
9           from
10                  c : Company!Company
11          to
12                  out : Enterprise!Enterprise (
13                          name <- c.name,
14                          employees <- c.getSumEmployees()
15                  )
16  }
```

B.3 Exemplarische Transformation eines Modells mit der Model Transformation Language (MTL)

Der im Listing B.5 dargestellte Pseudocode, soll eine Transformationsspezifikation, des in Abbildung B.1 dargestellten Quellmetamodells, in das in Abbildung B.2 gezeigten Zielmetamodell mit Hilfe der *Model Transformation Language*, veranschaulichen.

Listing B.5: Ausschnitte aus der MTL Transformationsspezifkikation

```
1  library Company2Enterprise;
2
3  model source_model : Company;
4  model target_model : Enterprise;
5
6  main() : Standard::Void
7  {
8           /*
9           * Pseudocode
10          */
11          for each companyInstance c in Company do
12                  instantiate enterprise in Enterprise with e.name = c.
                        name;
13                  if (c.getOrganizationalUnits not empty)
14                          while (c.getOrganizationlUnits.getNext not
                                empty) do
15                                  e.employees = e.employees + o.
                                        employees;
16                          done
17                  else
18                          e.employees = 0;
```

```
19                endIf
20            done
21 }
```

B.4 Exemplarische Transformation eines Modells mit der Bidirectional Object Oriented Transformation Language (BOTL)

Der folgende Abschnitt soll die Arbeits- und Funktionsweise der *Bidirectional Object Oriented Transformation Language* anhand des am Anfang dieses Kapitels vorgestellten Beispiels erläutern.

Die Werkzeugunterstützung von BOTL, ArgoUML4BOTL stellt eine graphische Benutzeroberfläche bereit, mit der sowohl die Modelle, als auch die Transformationsregeln erstellt werden können. Für die Transformation eines Quellmodells in ein Zielmodell, die Instanzen der am Anfang des Kapitels definierten Metamodelle sind, wird lediglich eine Transformationsregel benötigt, vgl. Abbildung B.3.

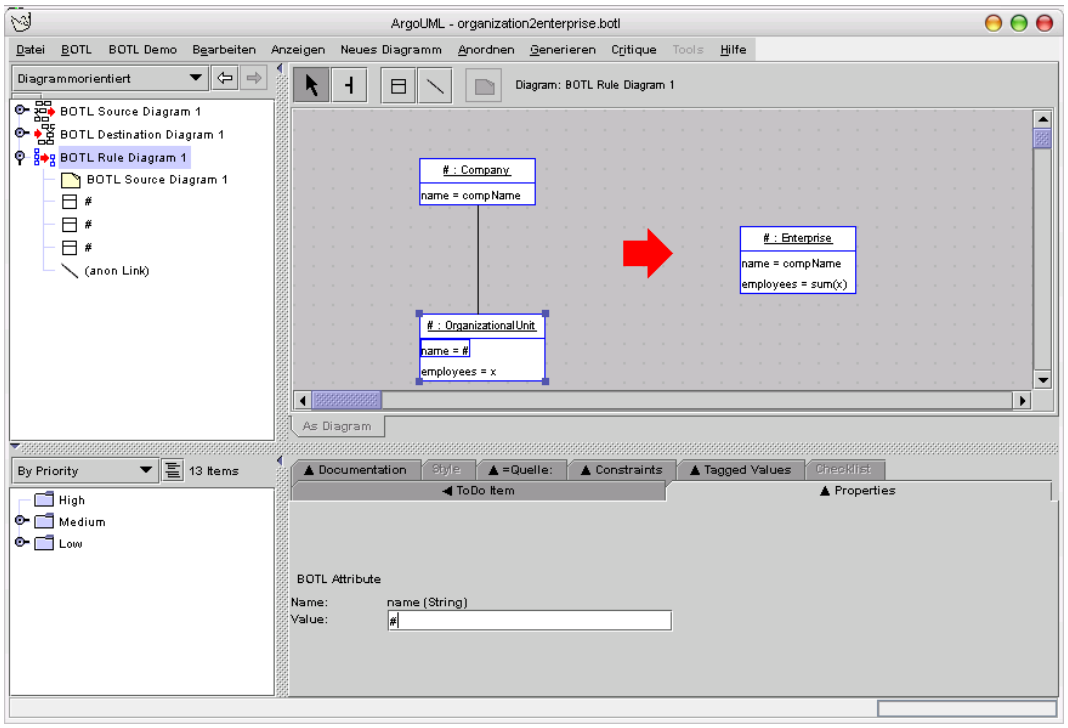


Abbildung B.3: Graphische Regeldefinition in ArgoUML4BOTL

Die Definition von mathematischen Funktionen (vgl. sum()) wird von ArgoUML4BOTL mit Hilfe des JEP[2] unterstützt. Eine eingeschränkte Verifikationsmöglichkeit der Anwendbarkeit und Metamodellkonformität wird in dem aktuell verfügbaren Werkzeug angeboten. Listing B.6 visualisiert Auschnitte des Verifikationsprotokolls der obigen Regeldefinition.

Listing B.6: BOTL Verifikationsprotokoll

```
1  ================== BOTL Verification Protocol ==================
2          Verification of rule set organization2enterprise
3  ================================================================
4
5      Verification of the syntax of rule BOTL Rule Diagram 1:
6        OK - There are no expressions in the identifiers of target
             object variables.
7        OK - There are no # values in primary key attribute variables
             of of target object variables.
8        OK - There are no tuple values in attribute variables of of
             target object variables.
9        OK - There are no tuple values in attribute variables of of
             source object variables.
10       OK - There are string expressions in attribute variables of of
             source object variables.
11       OK - No variable name starts with "botl".
12       OK - The source model variable contains no custom functions.
13     OK Rule BOTL Rule Diagram 1 is syntactically correct.
14
15 OK Rule set organization2enterprise is syntactically correct.
16 Verification of the applicability of rule set organization2enterprise
       :
17   Verification of the applicability of every rule in
       organization2enterprise:
18   Verification of the applicability of rule BOTL Rule Diagram 1
19   Verification of CreatesValidFragments(BOTL Rule Diagram 1)
       according to Theorem 4.1.2
20
21 Verification of the lower bounds conformance of rule set
     organization2enterprise:
22
23   Verfication of lbConform(BOTL Rule Diagram 1):
24     Calculation of varLbConform(BOTL Rule Diagram 1|mv1)
25     varLbConform(BOTL Rule Diagram 1|mv1) = true
26
27 => OK Rule set organization2enterprise is lower bounds conform.
28
29 Verification of metamodel conformance according to Theorem 4.2.7:
```

[2] Java Mathematical Expression Parser

```
30   OK Rule set organization2enterprise is applicable.
31   OK lbConf(organization2enterprise)
32   OK ubConf(organization2enterprise)
33 OK Rule set organization2enterprise is metamodel conform.
```

C Transformationsspezifikation zwischen Informationsmodellen

Die in dieser Arbeit entwickelten Konzepte sollen anhand einer exemplarischen Transformation von Informationsmodellen validiert werden. Als Basis dienen die in Kapitel 6 vorgestellten Quell- und Zielmetamodelle. Für die Transformation mit der Atlas Transformation Language müssen diese Modelle im ecore Format bereitgestellt werden. Das folgende Listing C.1 zeigt das ecore-Modell, des in Abschnitt 6.1.1 vorgestellten Quellmetamodells.

Listing C.1: Ecore-Modell des Quellmetamodells

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="
       http://www.w3.org/2001/XMLSchema-instance" xmlns:ecore="http://www.
       eclipse.org/emf/2002/Ecore">
3    <ecore:EPackage name="PrimitiveTypes">
4      <eClassifiers xsi:type="ecore:EDataType" name="Integer"/>
5      <eClassifiers xsi:type="ecore:EDataType" name="String"/>
6    </ecore:EPackage>
7    <ecore:EPackage name="Unternehmen">
8      <eClassifiers xsi:type="ecore:EClass" name="AllObjects" abstract="true">
9        <eStructuralFeatures xsi:type="ecore:EAttribute" name="Name" ordered="
             false" lowerBound="1" eType="#/0/String"/>
10       <eStructuralFeatures xsi:type="ecore:EAttribute" name="Description"
             ordered="false" lowerBound="1" eType="#/0/String"/>
11     </eClassifiers>
12     <eClassifiers xsi:type="ecore:EClass" name="Service" eSuperTypes="#/1/
         AllObjects">
13       <eStructuralFeatures xsi:type="ecore:EAttribute" name="Obligation" ordered
             ="false" lowerBound="1" eType="#/0/String"/>
14       <eStructuralFeatures xsi:type="ecore:EAttribute" name="Status" ordered="
             false" lowerBound="1" eType="#/0/String"/>
15       <eStructuralFeatures xsi:type="ecore:EAttribute" name="Quality" ordered="
             false" lowerBound="1" eType="#/0/String"/>
16       <eStructuralFeatures xsi:type="ecore:EReference" name="basedon" ordered="
             false" upperBound="-1" eType="#/1/Service"/>
17       <eStructuralFeatures xsi:type="ecore:EReference" name="component" ordered=
             "false" upperBound="-1" eType="#/1/Component"/>
```

```
18      <eStructuralFeatures xsi:type="ecore:EReference" name="
            deployedapplication" ordered="false" upperBound="-1" eType="#/1/
            DeployedApplication"/>
19      <eStructuralFeatures xsi:type="ecore:EReference" name="application"
            ordered="false" upperBound="-1" eType="#/1/Application"/>
20    </eClassifiers>
21    <eClassifiers xsi:type="ecore:EClass" name="Component" eSuperTypes="#/1/
          AllObjects">
22      <eStructuralFeatures xsi:type="ecore:EAttribute" name="Vendor" ordered="
            false" lowerBound="1" eType="#/0/String"/>
23      <eStructuralFeatures xsi:type="ecore:EAttribute" name="Category" ordered="
            false" lowerBound="1" eType="#/0/String"/>
24      <eStructuralFeatures xsi:type="ecore:EAttribute" name="Status" ordered="
            false" lowerBound="1" eType="#/0/String"/>
25      <eStructuralFeatures xsi:type="ecore:EReference" name="basedon" ordered="
            false" upperBound="-1" eType="#/1/Component"/>
26      <eStructuralFeatures xsi:type="ecore:EReference" name="service" ordered="
            false" upperBound="-1" eType="#/1/Service"/>
27    </eClassifiers>
28    <eClassifiers xsi:type="ecore:EClass" name="DeployedApplication"
          eSuperTypes="#/1/AllObjects">
29      <eStructuralFeatures xsi:type="ecore:EReference" name="location" ordered="
            false" upperBound="-1" eType="#/1/Location"/>
30      <eStructuralFeatures xsi:type="ecore:EReference" name="service" ordered="
            false" upperBound="-1" eType="#/1/Service"/>
31      <eStructuralFeatures xsi:type="ecore:EReference" name="application"
            ordered="false" upperBound="-1" eType="#/1/Application"/>
32    </eClassifiers>
33    <eClassifiers xsi:type="ecore:EClass" name="Location" eSuperTypes="#/1/
          AllObjects">
34      <eStructuralFeatures xsi:type="ecore:EAttribute" name="Address" ordered="
            false" lowerBound="1" eType="#/0/String"/>
35      <eStructuralFeatures xsi:type="ecore:EAttribute" name="City" ordered="
            false" lowerBound="1" eType="#/0/String"/>
36      <eStructuralFeatures xsi:type="ecore:EAttribute" name="ZipCode" ordered="
            false" lowerBound="1" eType="#/0/String"/>
37      <eStructuralFeatures xsi:type="ecore:EAttribute" name="Country" ordered="
            false" lowerBound="1" eType="#/0/String"/>
38      <eStructuralFeatures xsi:type="ecore:EReference" name="
            deployedApplication" ordered="false" upperBound="-1" eType="#/1/
            DeployedApplication"/>
39    </eClassifiers>
40    <eClassifiers xsi:type="ecore:EClass" name="Application" eSuperTypes="#/1/
          AllObjects">
41      <eStructuralFeatures xsi:type="ecore:EAttribute" name="Status" ordered="
            false" lowerBound="1" eType="#/0/String"/>
42      <eStructuralFeatures xsi:type="ecore:EAttribute" name="Vendor" ordered="
            false" lowerBound="1" eType="#/0/String"/>
```

```
43      <eStructuralFeatures xsi:type="ecore:EReference" name="
            deployedApplication" ordered="false" upperBound="-1" eType="#/1/
            DeployedApplication"/>
44      <eStructuralFeatures xsi:type="ecore:EReference" name="service" ordered="
            false" upperBound="-1" eType="#/1/Service"/>
45      <eStructuralFeatures xsi:type="ecore:EReference" name="organization"
            ordered="false" upperBound="-1" eType="#/1/Organization"/>
46      <eStructuralFeatures xsi:type="ecore:EReference" name="process" ordered="
            false" upperBound="-1" eType="#/1/Process"/>
47    </eClassifiers>
48    <eClassifiers xsi:type="ecore:EClass" name="Organization" eSuperTypes="
          #/1/AllObjects">
49      <eStructuralFeatures xsi:type="ecore:EReference" name="application"
            ordered="false" upperBound="-1" eType="#/1/Application"/>
50    </eClassifiers>
51    <eClassifiers xsi:type="ecore:EClass" name="Process" eSuperTypes="#/1/
          AllObjects">
52      <eStructuralFeatures xsi:type="ecore:EReference" name="application"
            ordered="false" upperBound="-1" eType="#/1/Application"/>
53    </eClassifiers>
54  </ecore:EPackage>
55 </xmi:XMI>
```

Als Zielmetamodell dient das in Abschnitt 6.1.3 konzipierte Informationsmodell. Das folgende Listing C.2 zeigt das ecore-Modell des Zielmetamodells.

Listing C.2: Ecore-Modell des Zielmetamodells

```
1 <?xml version="1.0" encoding="ASCII"?>
2 <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="
      http://www.w3.org/2001/XMLSchema-instance" xmlns:ecore="http://www.
      eclipse.org/emf/2002/Ecore">
3   <ecore:EPackage name="Sebis">
4     <eClassifiers xsi:type="ecore:EClass" name="ModelElement" abstract="true">
5       <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" ordered="
            false" lowerBound="1" eType="#/1/String"/>
6       <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
            ordered="false" lowerBound="1" eType="#/1/String"/>
7     </eClassifiers>
8     <eClassifiers xsi:type="ecore:EClass" name="InfraService" eSuperTypes="
          #/0/ModelElement">
9       <eStructuralFeatures xsi:type="ecore:EAttribute" name="status" ordered="
            false" lowerBound="1" eType="#/1/String"/>
10      <eStructuralFeatures xsi:type="ecore:EAttribute" name="quality" ordered="
            false" lowerBound="1" eType="#/1/String"/>
11      <eStructuralFeatures xsi:type="ecore:EAttribute" name="obligation" ordered
            ="false" lowerBound="1" eType="#/1/String"/>
12      <eStructuralFeatures xsi:type="ecore:EReference" name="basedon" ordered="
            false" upperBound="-1" eType="#/0/InfraService"/>
```

```
13      <eStructuralFeatures xsi:type="ecore:EReference" name="
            servicelevelagreement" ordered="false" upperBound="-1" eType="
            #/0/ServiceLevelAgreement"/>
14      <eStructuralFeatures xsi:type="ecore:EReference" name="
            infrastructureelement" ordered="false" upperBound="-1" eType="
            #/0/InfraStructureElement"/>
15    </eClassifiers>
16    <eClassifiers xsi:type="ecore:EClass" name="ServiceLevelAgreement"
          eSuperTypes="#/0/ModelElement">
17      <eStructuralFeatures xsi:type="ecore:EAttribute" name="category" ordered="
            false" lowerBound="1" eType="#/1/String"/>
18      <eStructuralFeatures xsi:type="ecore:EReference" name="infraservice"
            ordered="false" upperBound="-1" eType="#/0/InfraService"/>
19      <eStructuralFeatures xsi:type="ecore:EReference" name="deployedelement"
            ordered="false" lowerBound="1" eType="#/0/DeployedElement"/>
20    </eClassifiers>
21    <eClassifiers xsi:type="ecore:EClass" name="DeployableElement" abstract="
          true" eSuperTypes="#/0/ModelElement">
22      <eStructuralFeatures xsi:type="ecore:EAttribute" name="status" ordered="
            false" lowerBound="1" eType="#/1/String"/>
23      <eStructuralFeatures xsi:type="ecore:EAttribute" name="vendor" ordered="
            false" lowerBound="1" eType="#/1/String"/>
24      <eStructuralFeatures xsi:type="ecore:EReference" name="deployedelement"
            ordered="false" upperBound="-1" eType="#/0/DeployedElement"/>
25    </eClassifiers>
26    <eClassifiers xsi:type="ecore:EClass" name="InfraStructureElement"
          eSuperTypes="#/0/DeployableElement">
27      <eStructuralFeatures xsi:type="ecore:EAttribute" name="category" ordered="
            false" lowerBound="1" eType="#/1/String"/>
28      <eStructuralFeatures xsi:type="ecore:EReference" name="basedon" ordered="
            false" upperBound="-1" eType="#/0/InfraStructureElement"/>
29      <eStructuralFeatures xsi:type="ecore:EReference" name="infraservice"
            ordered="false" upperBound="-1" eType="#/0/InfraService"/>
30    </eClassifiers>
31    <eClassifiers xsi:type="ecore:EClass" name="DeployedElement" abstract="true
          " eSuperTypes="#/0/ModelElement">
32      <eStructuralFeatures xsi:type="ecore:EAttribute" name="status" ordered="
            false" lowerBound="1" eType="#/1/String"/>
33      <eStructuralFeatures xsi:type="ecore:EReference" name="deployableelement
            " ordered="false" lowerBound="1" eType="#/0/DeployableElement"/>
34      <eStructuralFeatures xsi:type="ecore:EReference" name="
            servicelevelagreement" ordered="false" upperBound="-1" eType="
            #/0/ServiceLevelAgreement"/>
35      <eStructuralFeatures xsi:type="ecore:EReference" name="location" ordered="
            false" lowerBound="1" eType="#/0/Location"/>
36    </eClassifiers>
37    <eClassifiers xsi:type="ecore:EClass" name="InfraStructureElementDeployed
          " eSuperTypes="#/0/DeployedElement"/>
```

```
38    <eClassifiers xsi:type="ecore:EClass" name="Location" eSuperTypes="#/0/
          ModelElement">
39      <eStructuralFeatures xsi:type="ecore:EAttribute" name="plz" ordered="false
            " lowerBound="1" eType="#/1/String"/>
40      <eStructuralFeatures xsi:type="ecore:EAttribute" name="city" ordered="
            false" lowerBound="1" eType="#/1/String"/>
41      <eStructuralFeatures xsi:type="ecore:EAttribute" name="country" ordered="
            false" lowerBound="1" eType="#/1/String"/>
42      <eStructuralFeatures xsi:type="ecore:EAttribute" name="address" ordered="
            false" lowerBound="1" eType="#/1/String"/>
43      <eStructuralFeatures xsi:type="ecore:EReference" name="deployedelement"
            ordered="false" upperBound="-1" eType="#/0/DeployedElement"/>
44      <eStructuralFeatures xsi:type="ecore:EReference" name="
            organizationalunit" ordered="false" upperBound="-1" eType="#/0/
            OrganizationalUnit"/>
45    </eClassifiers>
46    <eClassifiers xsi:type="ecore:EClass" name="ApplicationDeployed"
          eSuperTypes="#/0/DeployedElement">
47      <eStructuralFeatures xsi:type="ecore:EReference" name="
            supportrelationship" ordered="false" upperBound="-1" eType="#/0/
            SupportRelationship"/>
48    </eClassifiers>
49    <eClassifiers xsi:type="ecore:EClass" name="ApplicationVersion"
          eSuperTypes="#/0/DeployableElement">
50      <eStructuralFeatures xsi:type="ecore:EAttribute" name="version" ordered="
            false" lowerBound="1" eType="#/1/String"/>
51      <eStructuralFeatures xsi:type="ecore:EReference" name="application"
            ordered="false" lowerBound="1" eType="#/0/Application"/>
52      <eStructuralFeatures xsi:type="ecore:EReference" name="businessprocess"
            ordered="false" upperBound="-1" eType="#/0/BusinessProcess"/>
53      <eStructuralFeatures xsi:type="ecore:EReference" name="next" ordered="
            false" upperBound="-1" eType="#/0/ApplicationVersion"/>
54      <eStructuralFeatures xsi:type="ecore:EReference" name="pre" ordered="false
            " upperBound="-1" eType="#/0/ApplicationVersion"/>
55    </eClassifiers>
56    <eClassifiers xsi:type="ecore:EClass" name="Application" eSuperTypes="#/0/
          ModelElement">
57      <eStructuralFeatures xsi:type="ecore:EReference" name="
            applicationversion" ordered="false" upperBound="-1" eType="#/0/
            ApplicationVersion"/>
58    </eClassifiers>
59    <eClassifiers xsi:type="ecore:EClass" name="BusinessProcess" eSuperTypes="
          #/0/ModelElement">
60      <eStructuralFeatures xsi:type="ecore:EReference" name="
            applicationversion" ordered="false" upperBound="-1" eType="#/0/
            ApplicationVersion"/>
61      <eStructuralFeatures xsi:type="ecore:EReference" name="pre" ordered="false
            " eType="#/0/BusinessProcess"/>
```

```
62      <eStructuralFeatures xsi:type="ecore:EReference" name="next" ordered="
            false" eType="#/0/BusinessProcess"/>
63      <eStructuralFeatures xsi:type="ecore:EReference" name="parent" ordered="
            false" eType="#/0/BusinessProcess"/>
64      <eStructuralFeatures xsi:type="ecore:EReference" name="child" ordered="
            false" upperBound="-1" eType="#/0/BusinessProcess"/>
65      <eStructuralFeatures xsi:type="ecore:EReference" name="
            supportrelationship" ordered="false" upperBound="-1" eType="#/0/
            SupportRelationship"/>
66    </eClassifiers>
67    <eClassifiers xsi:type="ecore:EClass" name="SupportRelationship"
          eSuperTypes="#/0/ModelElement">
68      <eStructuralFeatures xsi:type="ecore:EReference" name="
            applicationdeployed" ordered="false" lowerBound="1" eType="#/0/
            ApplicationDeployed"/>
69      <eStructuralFeatures xsi:type="ecore:EReference" name="businessprocess"
            ordered="false" lowerBound="1" eType="#/0/BusinessProcess"/>
70      <eStructuralFeatures xsi:type="ecore:EReference" name="
            organizationalunit" ordered="false" lowerBound="1" eType="#/0/
            OrganizationalUnit"/>
71    </eClassifiers>
72    <eClassifiers xsi:type="ecore:EClass" name="OrganizationalUnit"
          eSuperTypes="#/0/ModelElement">
73      <eStructuralFeatures xsi:type="ecore:EReference" name="location" ordered="
            false" upperBound="-1" eType="#/0/Location"/>
74      <eStructuralFeatures xsi:type="ecore:EReference" name="
            supportrelationship" ordered="false" upperBound="-1" eType="#/0/
            SupportRelationship"/>
75    </eClassifiers>
76  </ecore:EPackage>
77  <ecore:EPackage name="PrimitiveTypes">
78    <eClassifiers xsi:type="ecore:EDataType" name="Integer"/>
79    <eClassifiers xsi:type="ecore:EDataType" name="String"/>
80    <eClassifiers xsi:type="ecore:EDataType" name="Boolean"/>
81  </ecore:EPackage>
82 </xmi:XMI>
```

Das folgende Listing C.3 stellt die für die Transformation zwischen den vorgestellten Metamodellen notwendigen Regeln in ATL dar.

Listing C.3: ATL Regelspezifikation

```
1 module Unternehmen2Sebis;
2 create OUT : Sebis from IN : Unternehmen;
3
4 rule Service2InfraService {
5         from
6                 s : Unternehmen!Service
7         to
8                 out : Sebis!InfraService (
```

```
9                       name <- s.Name,
10                      description <- s.Description,
11                      status <- s.Status,
12                      quality <- s.Quality,
13                      obligation <- s.Obligation,
14                      basedon <- s.basedon,
15                      infrastructureelement <- s.component,
16                      servicelevelagreement <- s.component
17              )
18  }
19
20  rule Component2InfraStructureElement {
21          from
22                  c : Unternehmen!Component (
23                          c.Category <> 'ServiceLevel'
24                  )
25          to
26                  is : Sebis!InfraStructureElement (
27                          name <- c.Name,
28                          description <- c.Description,
29                          category <- c.Category,
30                          basedon <- c.basedon,
31                          infraservice <- c.service,
32                          deployedelement <- ised
33
34                  ),
35                  ised : Sebis!InfraStructureElementDeployed (
36                          name <- c.Name,
37                          description <- c.Description,
38                          status <- 'deployed',
39                          deployableelement <- is
40                  )
41  }
42
43  rule Component2ServiceLevelAgreement {
44          from
45                  c : Unternehmen!Component (
46                          c.Category = 'ServiceLevel'
47                  )
48          to
49                  out : Sebis!ServiceLevelAgreement (
50                          name <- c.Name,
51                          description <- c.Description,
52                          category <- 'ServiceLevel',
53                          infraservice <- c.service
54                  )
55  }
56
57  rule Location2Location {
```

```
58          from
59                  l : Unternehmen!Location
60          to
61                  out : Sebis!Location (
62                          name <- l.Name,
63                          description <- l.Description,
64                          plz <- l.ZipCode,
65                          city <- l.City,
66                          country <- l.Country,
67                          address <- l.Address,
68                          deployedElement <- l.deployedApplication
69                  )
70  }
71
72  rule Organization2OrganizationalUnit {
73          from
74                  o : Unternehmen!Organization
75          to
76                  out : Sebis!OrganizationalUnit (
77                          name <- o.Name,
78                          description <- o.Description
79                  )
80  }
81
82  rule Process2BusinessProcess {
83          from
84                  p : Unternehmen!Process
85          to
86                  out : Sebis!BusinessProcess (
87                          name <- p.Name,
88                          description <- p.Description,
89                          applicationversion <- p.application
90                  )
91  }
92
93  rule Application2ApplicationVersion {
94          from
95                  a : Unternehmen!Application
96          to
97
98                  app : Sebis!Application (
99                          name <- a.Name,
100                         description <- a.Description,
101                         applicationversion <- av
102                 ),
103                 av : Sebis!ApplicationVersion (
104                         name <- a.Name,
105                         description <- a.Description,
106                         status <- a.Status,
```

```
107                         vendor <- a.Vendor,
108                         version <- 'xxx',
109                         businessprocess <- a.process,
110                         application <- app,
111                         deployedelement <- a.deployedApplication
112                 )
113 }
114
115 rule DeployedApplication2ApplicationDeployed {
116         from
117                 d : Unternehmen!DeployedApplication
118         to
119                 out : Sebis!ApplicationDeployed (
120                         name <- d.Name,
121                         description <- d.Description,
122                         status <- 'deployed',
123                         applicationversion <- d.application,
124                         location <- d.location
125                 )
126 }
```

Literaturverzeichnis

[Bal01] Balzert, H.: *Lehrbuch der Software-Technik*. 2. Auflage. Berlin, Heidelberg : Spektrum Akademischer Verlag, 2001.

[Bey04] Beyer, N.: *Kennzahlen zur Beschreibung von Anwendungslandschaften und ihre Visualisierung auf Softwarekarten*. München, Technische Universität München, Fakultät für Informatik, Bachelor-Arbeit, 2004.

[Bic04] Bichler, L.: *Codegeneratoren für MOF-basierte Modellierungssprachen*. München, Universität der Bundeswehr München, Fakultät für Informatik, Dissertation, 2004.

[BNS03] Bernus, P. ; Nemes, L. ; Schmidt, G.: *Handbook on Enterprise Architecture*. Berlin, Heidelberg : Springer-Verlag, 2003.

[BP96] Blaha, M. ; Premerlani, W.: *A Catalog of Object Model Transformations*. In: Proceedings of WCRE'96. Monterey, CA, USA : IEEE Computer Society Press, 1996.

[Bre05] Brendebach, K.: *Integrierte Modelle und Sichten für das IT-Management*, Technische Universität München, Fakultät für Informatik, Diplomarbeit, 2005.

[BS04] Becker, J. ; Schütte, R.: *Handelsinformationssysteme*. 2. Auflage. Frankfurt am Main : Verlag Moderne Industrie, 2004.

[Bud03] Budinsky, F. et al.: *Eclipse Modeling Framework - A Developer's Guide*. Boston San Francisco New York : Addison-Wesley, 2003.

[Buh04] Buhl, H. U.: *Unternehmensarchitekturen in der Praxis - Architekturdesign am Reißbrett vs. situationsbedingte Realisierung von Informationssystemen*. In: Wirtschaftsinformatik 4 (2004). Vieweg Verlag, 2004.

[Cas89] Case, J. et al.: *A simple network management protocol*. Menlo Park, Ca: Network Information Center, SRI International, 1989.

[CH03] Czarnecki, K. ; Helsen, S.: *Classification of Model Transformation Approaches*. In: Proceedings of the 2nd OOPSLA'03 - Workshop on Generative Techniques in the Context of Model-Driven Architecture. Anaheim, California, 2003.

[Der03] Dern, G.: *Management von IT-Architekturen*. Wiesbaden : Vieweg Verlag, 2003.

[DMT03a] DMTF: *CIM Concepts White Paper CIM - Version 2.4+* Distributed Management Task Force, 2003.

[DMT03b] DMTF: *Desktop Management Interface Specification - Version 2.0.1s*. Distributed Management Task Force, 2003.

[DMT05] DMTF: *Common Information Model (CIM) Infrastructure Specification - Version 2.3 Final*. Distributed Management Task Force, 2005.

[Dum05] Dumke, R. R.: *UML-Tutorial*. 2005. <http://ivs.cs.uni-magdeburg.de/dumke/UML/> (abgerufen 2005-10-22).

[Ecl05a] Eclipse Foundation: *eclipse project universal tool platform*. 2005. <http://www.eclipse.org/gmt> (abgerufen 2005-11-12).

[Ers05] Ernst, A. M. et al.: *Visualizing Application Landscapes utilizing a Symbolic Model for Software Cartography* . Technische Universität München, Fakultät für Informatik, Lehrstuhl für Informatik 19. Technischer Bericht, 2005.

[Ess02] Esser, M.: *Komplexitätsbeherrschung in dynamischen Diskurswelten*. Lohmar, Köln : Josef Eul Verlag, 2002.

[Fed01] Federal Architecture Working Group (FAWG): *A Practical Guide to Federal Enterprise Architecture*. Enterprise Interoperability and Emerging Information Technology Group Committee (EIEITC). Report, 2001.

[Fra94] Frank, U.: *Multiperspektivische Unternehmensmodellierung. Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung*. München : Oldenbourg Verlagsgruppe, 1994.

[Fra05] Fraunhofer Institut Arbeitswirtschaft und Organisation: *Office Performance*. 2005. <http://www.office21.de> (abgerufen 2005-11-24).

[Gae04] Gaertner, W.: *Ansatz für eine erfolgreiche Enterprise Architecture im Bereich Global Banking Devision*. In: Wirtschaftsinformatik 4 (2004). Vieweg Verlag, 2004.

[Gar03] Gardner, T. et al.: *A review of OMG 2.0 Query / Views / Transformations Submissions and Recommondations towards the final Standard*. IBM Zurich Research Laboratory. Report, 2003.

[Hal04] Halbhuber, T.: *Entwicklung eines Informationsmodells für das IT-Management*. Technische Universität München, Fakultät für Maschinenwesen, Diplomarbeit, 2004.

[Hit05] Hitz, M. et al.: *UML@Work*. 3. Auflage. Heidelberg : dpunkt.verlag, 2005.

[IEE00] IEEE: *IEEE Std 1471-2000: Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE Computer Society, 2000.

[INR05a] INRIA: *Model transformation at INRIA ModelWare*. 2005. <http://modelware.inria.fr> (abgerufen 2005-11-12).

[INR05b] INRIA: *The French National Institute for Research in Computer Science and Control*. 2005. <http://www.inria.fr> (abgerufen 2005-11-12).

[IRI05] IRISA: *Triskell Project*. 2005. <http://www.irisa.fr/triskell> (abgerufen 2005-11-12).

[Jec00] Jeckle, M.: *Konzepte zur Metamodellierung - Zum Begriff Metamodell*. In: Softwaretechnik-Trends Band 20 (2). Gesellschaft für Informatik, 2000.

[JK05] Jouault, F. ; Kurtev, I.: *Transformating Models with ATL*. In: MoDELS 2005 Conference: Modeltransformations in Practice Workshop. Montego Bay, Jamaica, 2005.

[KE04] Kemper, A. ; Eickler, A.: *Datenbanksysteme*. München, Wien : Oldenbourg Verlag, 2004.

[KHB05] Kath, O. ; Holz, E. ; Born, M.: *Softwareentwicklung mit UML 2.0 - Die neuen Entwurfstechniken UML 2, MOF 2 und MDA*. Addison-Wesley, 2005.

[Kle04] Klement, P.: *Integrated Enterprise Architecture*. Deutschland : Microsoft Press, 2004.

[Krc05] Krcmar, H.: *Informationsmanagement*. 4. Auflage. Berlin, Heidelberg : Springer-Verlag, 2005.

[KWB03] Kleppe, A. ; Warmer, J. ; Bast, W.: *MDA explained: The Model Driven Architecture: Practice and Promise*. Addison Wesley Professional, 2003.

[Lau04] Lauschke, S.: *Softwarekartographie: Analyse und Darstellung der IT-Landschaft eines mittelständischen Unternehmens*. Technische Universität München, Fakultät für Informatik, Bachelor-Arbeit, 2004.

[LMW05a] Lankes, J. ; Matthes, F. ; Wittenburg, A.: *Architekturbeschreibungen von Anwendungslandschaften: Softwarekartographie und IEEE Std 1471-2000*. In (Leggismeyer, P.; Pohl, K.; Goedicke, M. Hrsg.): Software Engineering 2005. Essen, 2005.

[LMW05b] Lankes, J. ; Matthes, F. ; Wittenburg, A.: *Softwarekartographie als Beitrag zum Architekturmanagement*. In (Aier, S.; Schönherr, M. Hrsg.): Unternehemensarchitekturen und Systemintegration. Band 3. Berlin : GITO-Verlag, 2005.

[LMW05c] Lankes, J. ; Matthes, F. ; Wittenburg, A.: *Softwarekartographie: Systematische Darstellungen von Anwendungslandschaften*. In (Ferstl, O. et al. Hrsg.): Wirtschaftsinformatik 2005. Bamberg, 2005.

[LW04] Langenberg, K. ; Wegmann, A.: *Enterprise Architecture: What Aspects is Current Research Targeting*. Ecole Polytechnique Fédérale de Lausanne (EPFL). Technischer Bericht IC/2004/77, 2004.

[Mar04] Marschall, F.: *Modelltransformationen als Mittel der modellbasierten Entwicklung von Software-Systemen*. Technische Universität München, Fakultät für Informatik, Dissertation, 2004.

[Mar05] Marschall, F.: *The BOTL Tool*. 2005. <http://www4.in.tum.de/ marschal/botl/index.htm> (abgerufen 2005-11-13).

[McG03] McGovern, J. et al.: *A Practical Guide to Enterprise Architecture*. Prentice Hall PTR, 2003.

[MEGA05] MEGA: *MEGA Designer - System, Component and Information Design*. <http://www.mega.com/en/product/mega_designer/index.asp?l=en> (abgerufen 2005-12-14).

[MM03] Miller, J. ; Mukerji, J.: *MDA Guide Version 1.0.1*. Object Management Group, 2003.

[MS03] McKeen, J. D. ; Smith, H. A.: *Makint IT Happen*. Chichester : Wiley, 2003.

[MW04a] Matthes, F. ; Wittenburg, A.: *Softwarekarten zur Visualisierung von Anwendungslandschaften und ihrer Aspekte*. Technische Universität München, Fakultät für Informatik, Lehrstuhl für Informatik 19. Technischer Bericht 0401, 2004.

[MW04b] Matthes, F. ; Wittenburg, A.: *Softwarekartograhie: Visualisierung von Anwendungslandschaften und ihrer Schnittstellen*. In (Dadam, P.; Reichert, M. Hrsg.): Informatik 2004 - Jahrestagung der GI. Ulm, 2004.

[New02] Newcomb, R.: *Architecture of the Enterprise - Filling the role of enterprise software architect can help your organization stay competitive*. 2002. <http://www.intelligententerprise.com/020308/505e_business1_1.jhtml> (abgerufen 2005-10-04)

[Oes04] Oestereich, B.: *Die UML 2.0 Kurzreferenz für die Praxis*. 3. Auflage. München, Wien : Oldenbourg Verlag, 2004.

[OMG04a] OMG: *IT Portfolio Management Facility (ITPMF) Specification - Final Submission*. Object Management Group, 2004.

[OMG04b] OMG: *Meta Object Facility (MOF) 2.0 Core Specifications - ptc/04-10-15*. Object Management Group, 2004.

[OMG04c] OMG: *UML 2.0 Infrastructure Specification - ptc/04-10-14*. Object Management Group, 2004.

[OMG04d] OMG: *UML 2.0 Superstructure Specification - formal/05-07-04*. Object Management Group, 2005.

[OMG05a] OMG: *OCL 2.0 Specification - ptc/05-06-06*. Object Management Group, 2005.

[OMG05b] OMG: *Revised submission for MOF 2.0 Query / Views / Transformations RFP - ad/2002-04-10*. Object Management Group, 2005.

[Plü02] Plümicke, M.: *Software-Engineering im ARIS-Konzept als Ansatz der Integration der IT-Landschaft von Unternehmen*. Berufsakademie Stuttgart, Arbeitskreis Geschäftsprozesse und Workflowsysteme. Technischer Bericht, 2002.

[PRW03] Picot, A. ; Reichwald, R. ; Wigand, R.: *Die Grenzenlose Unternehmnung*. 5. Auflage. Wiesbaden : Gabler, 2003.

[PSS06] Prakash, N. ; Srivastava, S. ; Sabharwal, S.: *The Classification Framework for Model Transformation*. In: Journal of Computer Sience 2 (2), 2006.

[RHQ05] Rupp, C. ; Hahn, J. ; Queins, S.: *UML 2 glasklar*. München : Hanser Fachbuchverlag, 2005.

[Sch91] Scheer, A.-W.: *Architektur integrierter Informationssysteme*. Berlin : Springer-Verlag, 1991.

[Sch98] Scheer, A.-W.: *Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse*. 2. Auflage. Berlin, Heidelberg : Springer-Verlag, 1998.

[Sch01] Scheer, A.-W.: *ARIS-Modellierungsmethoden, Metamodelle, Anwendungen*. 4. Auflage. Berlin, Heidelberg : Springer-Verlag, 2001.

[Sch05] Schallert, M.: *Successfully Building Enterprise Architecture*. Leonardo Consulting. White Paper, 2005.

[seb05a] sebis: *Enterprise Architecture Management Tool Survey 2005*. Technische Universität München, Chair for Informatics 19 (sebis), 2005.

[seb05b] sebis: *Softwarekartographie-Wiki/Glossar*. Technische Universität München, Institut für Informatik, Lehrstuhl für Informatik 19, 2005. <http://wiki.softwarekartographie.de/index.php/Glossar> (abgerufen 2005-10-10).

[seb05d] sebis: *Softwarekartographie-Wiki/SoCaTool*. Technische Universität München, Insitut für Informatik, Lehrstuhl für Informatik 19, 2005. <http://wiki.softwarekartographie.de/index.php/SoCaTool> (abgerufen 2005-10-01).

[Sek05] Sekatzek, P.: *Visualisierung von IT-Bebauungsplänen in Form von Softwarekarten - Konzeption und prototypische Umsetzung*. Technische Universität München, Fakultät für Informatik, Diplomarbeit, 2005.

[SFS04] Silaghi, R. ; Fondement, F. ; Strohmeier, A.: *"Weaving" MTL Model Transformations*. INRIA, Technischer Bericht, 2004.

[Sin96] Sinz, E.: *Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme*. In (Heilmann, H.; Heinrich, L. J.; Roithmayr, F. Hrsg.): Information Engineering. München : Oldenbourg Verlag, 1996.

[Sin02] Sinz, E.: *Architektur von Informationssystemen*. In (Rechenber, P.; Pomberger, G. Hrsg.): Informatik-Handbuch. 3. Auflage. München : Hanser Verlag, 2002.

[Sin04] Sinz, E.: *Unternehmensarchitekturen in der Praxis - Architekturdesign am Reißbrett vs. situationsbedingte Realisierung von Informationssystemen*. In: Wirtschaftsinformatik 4 (2004). Vieweg Verlag, 2004.

[Sta73] Stachowiak, H.: *Allgemeine Modelltheorie*. Wien, New York : Springer-Verlag, 1973.

[Sys05] System and Software Consortium: *Architecture Frameworks: Zachman*. <http://www.software.org/pub/architecture/zachman.asp> (abgerufen 2005-10-08).

[SZ92] Sowa, J. F. ; Zachman, J. A.: *Extending and formalizing the framework for information system architecture*. In: IBM System Journal 31 (3), 1992.

[Teu99] Teubner, A.: *Organisations- und Informationssystemgestaltung*. Wiesbaden : Deutscher Universitäts Verlag, 1999.

[Tho05] Thomas, O.: *Modellverständnis in der Wirtschaftsinformatik: Historie, Literaturanalyse und Begriffsexplikation*. In (Scheer, A.-W. Hrsg.): Veröffentlichung des Instituts für Wirtschaftsinformatik Nr. 184, Saarbrücken, Universität des Saarlandes, 2005.

[Tig05] Tigris: *Tigris.org: Open Source Software Engineering Tools*. <http://argouml.tigris.org> (abgerufen 2005-11-12).

[Tor04] Torre, L. Van d. et al.: *Landscape Maps for Enterprise Architectures*. Information Centre of Telematica Instituut. Niederlande, Technischer Bericht TI/RS/2004/016, 2004.

[Uhl04] Uhl, J.: *"Unternehmensarchitekturen" ist ein Dauerthema - aber die Ziele bzw. Motivation und damit Schwerpunkte ändern sich, vor allem mit wirtschaftlichen Randbedingungen*. In: Wirtschaftsinformatik 4 (2004). Vieweg Verlag, 2004.

[Uni05a] Université de Nantes: *The AMMA Platform*. <http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT> (abgerufen 2005-11-09).

[Uni05b] Université de Nantes: *The ATL home page*. <http://www.sciences.univ-nantes.fr/lina/atl> (abgerufen 2005-11-09).

[VD05] Vojtisek, D. ; Dzale, S.: *BasicMTL User Manual*. INRIA/IRISA. Report, 2005.

[Win04] Winter, R.: *Architektur braucht Management*. In: Wirtschaftsinformatik 46 (2004). Vieweg Verlag, 2004.

[Zac87] Zachman, J. A.: *A framework for information system architecture*. In: IBM System Journal 26 (3) (1987).

[Zac97] Zachman, J. A.: *Enterprise Architecture - The Issue of the Century*. 1997. <http://www.zifa.com/articles.htm> (abgerufen 2005-10-07).

[Zac99] Zachman, J. A.: *Enterprise Architecture - Looking Back and Looking Ahead*. 1999. <http://www.zifa.com/articles.htm> (abgerufen 2005-10-07).

[Zac05] Zachman Institute for Framework Advancement: *Mission Statement*. <http://www.zifa.com> (abgerufen 2005-10-08).