universität wien

# MASTERARBEIT

Titel der Masterarbeit

„Ein Repository für Modellierungsmethoden: Konzeption und Implementierung“

Verfasserin

Daniela Schremser, BSc

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2011

| | |
|---|---|
| Studienkennzahl lt. Studienblatt: | A 066 926 |
| Studienrichtung lt. Studienblatt: | Masterstudium Wirtschaftsinformatik |
| Betreuerin / Betreuer: | o. Univ.-Prof. Dr. Dimitris Karagiannis |

# Schriftliche Versicherung

Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.

Ich versichere, dass die Arbeit von mir selbständig verfasst wurde und dass ich keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Weiterhin wurde diese Arbeit keiner anderen Prüfungsbehörde übergeben.

Wien, den 24. Januar 2011

# Danksagung

Besonderer Dank geht an meine Eltern, Andrea und Johann, ohne deren finanzielle und moralische Unterstützung diese Arbeit sowie mein gesamtes Studium nicht möglich gewesen wären. Des Weiteren möchte ich mich bei meiner Großmutter, Luise, für ihre Unterstützung bedanken.

Hiermit möchte ich mich ebenfalls bei meinem betreuenden Professor, o. Univ.-Prof. Dr. Dimitris Karagiannis, sowie meinem Betreuer Dr. Hans-Georg Fill für die zahlreichen Stunden bedanken, welche sie für etliche Diskussionen und Erläuterungen der wissenschaftlichen Inhalte dieser Arbeit investiert haben. Weiters bedanke ich mich bei o. Univ.-Prof. Dr. Dimitris Karagiannis für die Unterstützung bei meiner persönlichen Entwicklung während der letzten vier Jahre, in denen ich in seinem Team mitwirken durfte. Außerdem möchte ich dem gesamten Team der Forschungsgruppe Knowledge Engineering sowie den Mitarbeitern der BOC Asset Management GmbH für die Hilfe bei der Entstehung der Arbeit danken.

Letztendlich möchte ich mich auch bei meinen Studienkolleginnen und -kollegen bedanken, die mich während der letzten fünf Jahre durch eine aufschlussreiche und spannende Studienzeit begleitet haben. Ein ganz herzlicher Dank geht an Helga Androsch für das Korrekturlesen dieser Arbeit.

# Kurzfassung

Im Bereich der Wirtschaftsinformatik gewinnen Modelle und vor allem die verwendeten Modellierungsmethoden immer mehr an Bedeutung. Daher wurde 2008 die Open Model Initiative gegründet, die sich mit der Entwicklung und Bereitstellung von Modellierungsmethoden und deren Anwendungsmöglichkeiten beschäftigt. Um die Entwickler von Modellierungsmethoden zu unterstützen wird nun ein Repositorykonzept benötigt. Dadurch sollen dem Anwender sowohl Verwaltungsfunktionalität als auch Analysemöglichkeiten geboten werden.

Die Konzeption dieses Repositories basiert im Gegensatz zu den in der Literatur am häufigsten auftretenden Datenbank-basierenden Repositorykonzepten auf einem Metamodellierungsansatz, wodurch sich insbesonders einige wesentliche Vorteile ergeben. Diese Vorteile sind vor allem die einfache Integration der Modellierungsumgebung für die verwalteten Modellierungsmethoden sowie die Verwendung von Metamodellierungskonzepten sowohl für das Repository als auch für die Anwendung.

Ziel dieser Arbeit ist es nun ein Konzept für ein Modellierungsmethoden-Repository zu erstellen, welches alle notwendigen Funktionalitäten für deren Verwaltung zur Verfügung stellt und an die Bedürfnisse der Methodenentwickler angepasst ist. Das Konzept soll anschließend die Spezifikationsgrundlage für eine darauffolgende Implementierung bieten, wodurch eine weitere Verwendung des Repositories innerhalb der Open Model Initiative geboten wird.

# Abstract

Models and their used modelling methods become more important in the field of business informatics. Therefore, the Open Models Initiative was founded in 2008. This initiative deals with the development and supply of modelling methods and their applicability. A repository concept is needed due to the growing number of modelling methods and to support the method developer. This ensures the provision of management functionality as well as analytical possibilities.

The conception of this repository is based on a metamodelling approach in contrast to the most common concepts based on database technologies. This results in various advantages like easy integration of the modelling methods into the modelling environment and the usage of metamodelling approaches for the repository as well as for the use.

Aim of this work is the creation of a concept for a modelling method repository. This concept specifies all necessary functionalities and is adapted to the needs of method developers. The following implementation of the repository is based on this provided specification. Through which further usage of the repository within the Open Models Initiative is given.

# Inhaltsverzeichnis

**Abkürzungsverzeichnis** xiv

**Tabellenverzeichnis** xvi

**Abbildungsverzeichnis** xx

**Listings** xxii

**1 Einleitung** 1

1.1 Motivation . . . 1
1.2 Problemstellung und Zielsetzung . . . 2
1.3 Aufbau der Arbeit . . . 2

**2 Grundlagen** 5

2.1 Modellierung und Metamodellierung . . . 5
2.2 (Modell-)Management . . . 7
2.3 Repositories . . . 8
2.3.1 Definitionen von Repositories . . . 8
2.3.2 Funktionalitäten von Repositories . . . 9
2.3.3 Eigenschaften von Repositories . . . 12
2.3.4 Inhalt von Repositories . . . 14
2.3.5 Realisierung von Repositories . . . 16
2.4 Zusammenfassung . . . 17

**3 Anforderungen an das Modellierungsmethoden-Repository (MM-Repository)** 19

3.1 Nicht-funktionale Anforderungen . . . 19
3.1.1 Funktionalität . . . 20
3.1.2 Zuverlässigkeit . . . 21
3.1.3 Benutzbarkeit . . . 22
3.1.4 Effizienz . . . 22
3.1.5 Wartbarkeit . . . 22
3.1.6 Portabilität . . . 23

3.2 Funktionale Anforderungen . . . 23
3.2.1 Verwaltung von Objekten . . . 24
3.2.2 Repräsentation von Objekten . . . 25
3.2.3 Suchfunktionalität . . . 26
3.2.4 Versionierung von Objekte . . . 27
3.2.5 Zusammenstellung von Modellierungsmethoden . . . 27
3.2.6 Exportieren von Modellierungssprachen . . . 28
3.2.7 Erstellen von Dokumentationen . . . 28
3.2.8 Benutzerrollen . . . 28

**4 Modellierungsmethoden-Repository: Konzeption 31**
4.1 Eigenschaften des MM-Repositories . . . 31
4.2 Funktionalitäten des MM-Repositories . . . 31
4.2.1 Content-Management . . . 33
4.2.2 Zugriffsmanagement . . . 34
4.2.3 Transaktions- und Locking-Management . . . 37
4.2.4 Lebenszyklus-Management . . . 41
4.2.5 Abfragenmanagement . . . 42
4.2.6 Versionsmanagement . . . 44
4.2.7 Konfigurationsmanagement . . . 46
4.3 Externe Schnittstellen . . . 47
4.3.1 Benutzerverwaltung . . . 47
4.3.2 Verwaltung von Modellierungsmethoden . . . 48
4.3.3 Suchen . . . 49
4.3.4 Konfiguration einer neuen Modellierungsmethode . . . 50
4.3.5 Erstellung von Dokumentationen . . . 50
4.4 Architektur des MM-Repositories . . . 51
4.4.1 Klassenspezifikationen . . . 51
4.4.2 Relationsspezifikationen . . . 64
4.5 Benutzerinteraktion . . . 72
4.6 Vergleich dieses Konzepts mit datenbank-basierten Konzepten . . . 74
4.6.1 Vorteile . . . 74
4.6.2 Nachteile . . . 76
4.6.3 Schlussfolgerung . . . 77

**5 Modellierungsmethoden-Repository: Implementierung 79**
5.1 Infrastruktur . . . 79

5.2 An- und Abmeldung vom MM-Repository . . . . . 80
5.3 Versionierungsmechanismus . . . . . 83
5.4 Konfigurationsmanagement . . . . . 84

**6 Anwendungsbeispiele: i*, SDbD, OM-TV 89**
6.1 Open Models Initiative . . . . . 89
6.2 Speicherung und Verwaltung: Ein i*-Anwendungsfall . . . . . 91
6.3 Speicherung und Dokumentation: Ein SDbD-Anwendungsfall . . . . . 93
6.4 Zugriff auf das MM-Repository: OM-TV . . . . . 95

**7 Zusammenfassung und Ausblick 97**

**Literaturverzeichnis 99**

# Abkürzungsverzeichnis

| | |
|---|---|
| **API** | Application Programming Interface |
| **AST** | Abstract Syntax Tree |
| **CPU** | Central Processing Unit |
| **DB** | Datenbank |
| **DBMS** | Datenbankmanagementsystem |
| **DBS** | Datenbanksystem |
| **DOC** | DOCument |
| **EAD** | Encoded Archival Description |
| **EJB** | Enterprise JavaBeans |
| **HTML** | HyperText Markup Language |
| **ID3** | IDentify an MP3 |
| **i*** | iStar |
| **ISBN** | International Standard Book Number |
| **LOM** | Learning Objects Metadata |
| **MARC** | MAchine-Readable Cataloging |
| **METS** | Metadata Encoding and Transmission Standard |
| **MIME-Typ** | Multipurpose Internet Mail Extensions-Typ |
| **MM** | Modellierungsmethode |
| **MM-Repository** | Modellierungsmethoden-Repository |
| **MOF** | Meta Object Facility |
| **MPEG** | Moving Picture Experts Group |

| | |
|---|---|
| **OMG** | Object Management Group |
| **PDF** | Portable Document Format |
| **PREMIS** | PREservation Metadata Implementation Strategies |
| **RDF** | Resource Description Framework |
| **REST** | REpresentational State Transfer |
| **RSS** | Really Simple Syndication |
| **SCM** | Software Configuration Management |
| **SDbD** | Semantic Database Design |
| **SOAP** | Simple Object Access Protocol |
| **SQL** | Structured Query Language |
| **URI** | Uniform Resource Identifier |
| **XBMF** | Exchange Broadcast Binary and Metadata Format |
| **XML** | Extensible Markup Language |
| **XMP** | Extensible Metadata Platform |

# Tabellenverzeichnis


4.1 Schablone für die Spezifikation von Funktionen . . . . . 32
4.2 Funktionsspezifikation: Objekt erzeugen . . . . . 33
4.3 Funktionsspezifikation: Datei hochladen . . . . . 33
4.4 Funktionsspezifikation: Objekt aktualisieren . . . . . 34
4.5 Funktionsspezifikation: Benutzer anlegen . . . . . 34
4.6 Funktionsspezifikation: Benutzergruppe anlegen . . . . . 35
4.7 Funktionsspezifikation: Benutzer löschen . . . . . 35
4.8 Funktionsspezifikation: Benutzergruppe löschen . . . . . 35
4.9 Funktionsspezifikation: Rechte zuordnen . . . . . 36
4.10 Funktionsspezifikation: Benutzer anmelden . . . . . 36
4.11 Funktionsspezifikation: Benutzer abmelden . . . . . 37
4.12 Funktionsspezifikation: Transaktion anlegen . . . . . 38
4.13 Funktionsspezifikation: Transaktion starten . . . . . 38
4.14 Funktionsspezifikation: Transaktion rückgängig machen . . . . . 39
4.15 Funktionsspezifikation: Objekte einer Transaktion sperren . . . . . 39
4.16 Funktionsspezifikation: Objekte einer Transaktion sperren . . . . . 40
4.17 Funktionsspezifikation: Lesender Zugriff auf ein Objekt . . . . . 40
4.18 Funktionsspezifikation: Schreibender Zugriff auf ein Objekt . . . . . 41
4.19 Funktionsspezifikation: Objekt freigeben . . . . . 41
4.20 Funktionsspezifikation: Objekt archivieren . . . . . 42
4.21 Funktionsspezifikation: Aktuelle Version des Objekt festlegen . . . . . 42
4.22 Funktionsspezifikation: Abfrage erstellen . . . . . 43
4.23 Funktionsspezifikation: Abfrage aufrufen . . . . . 43
4.24 Funktionsspezifikation: Abfrage löschen . . . . . 44
4.25 Funktionsspezifikation: Versionsnummer automatisch vergeben . . . . . 44
4.26 Funktionsspezifikation: Versionen eines Objekts zurückgeben . . . . . 45
4.27 Funktionsspezifikation: Abhängigkeiten eines Objekts zurückgeben . . . . . 45
4.28 Funktionsspezifikation: Branch erzeugen . . . . . 45
4.29 Funktionsspezifikation: Konfiguration erstellen . . . . . 46
4.30 Funktionsspezifikation: Objekte von einer Konfiguration entfernen . . . . . 46

4.31 Funktionsspezifikation: Objekte zu einer Konfiguration hinzufügen . . . . . 47
4.32 Funktionsspezifikation: Konfiguration erstellen . . . . . 47
4.33 Schablone für die Spezifikation von Klassen . . . . . 51
4.34 Klassenspezifikation: MM-Repository Object . . . . . 54
4.35 Klassenspezifikation: Annotateable Object . . . . . 54
4.36 Klassenspezifikation: Versioned Object . . . . . 54
4.37 Klassenspezifikation: Configuration . . . . . 55
4.38 Klassenspezifikation: Topic . . . . . 55
4.39 Klassenspezifikation: OM Project . . . . . 55
4.40 Klassenspezifikation: OM Community . . . . . 55
4.41 Klassenspezifikation: OM User . . . . . 56
4.42 Klassenspezifikation: Location . . . . . 56
4.43 Klassenspezifikation: Metadata . . . . . 57
4.44 Klassenspezifikation: Modeldata . . . . . 57
4.45 Klassenspezifikation: Resource . . . . . 58
4.46 Klassenspezifikation: Tool . . . . . 58
4.47 Klassenspezifikation: Image . . . . . 59
4.48 Klassenspezifikation: Modellingmethod . . . . . 59
4.49 Klassenspezifikation: Mechanism . . . . . 60
4.50 Klassenspezifikation: Procedure . . . . . 60
4.51 Klassenspezifikation: Phase . . . . . 61
4.52 Klassenspezifikation: Step . . . . . 61
4.53 Klassenspezifikation: Modellinglanguage . . . . . 62
4.54 Klassenspezifikation: Modeltype . . . . . 62
4.55 Klassenspezifikation: Class . . . . . 63
4.56 Klassenspezifikation: Relation . . . . . 63
4.57 Klassenspezifikation: Attribute . . . . . 64
4.58 Klassenspezifikation: Notation . . . . . 64
4.59 Schablone für die Spezifikation von Relationen . . . . . 65
4.60 Relationsspezifikation: belongs to . . . . . 65
4.61 Relationsspezifikation: has topic . . . . . 65
4.62 Relationsspezifikation: has location . . . . . 65
4.63 Relationsspezifikation: has user . . . . . 66
4.64 Relationsspezifikation: has community . . . . . 66
4.65 Relationsspezifikation: has logo . . . . . 66
4.66 Relationsspezifikation: has creator . . . . . 66
4.67 Relationsspezifikation: has predecessor . . . . . 67

4.68 Relationsspezifikation: has components . . . . . 67
4.69 Relationsspezifikation: used tool . . . . . 67
4.70 Relationsspezifikation: has originalfile . . . . . 67
4.71 Relationsspezifikation: has specification . . . . . 68
4.72 Relationsspezifikation: has modellingmethod . . . . . 68
4.73 Relationsspezifikation: used modellingmethod . . . . . 68
4.74 Relationsspezifikation: has image . . . . . 68
4.75 Relationsspezifikation: has procedure . . . . . 69
4.76 Relationsspezifikation: has phase . . . . . 69
4.77 Relationsspezifikation: has step . . . . . 69
4.78 Relationsspezifikation: used mechanism . . . . . 69
4.79 Relationsspezifikation: used modeltype . . . . . 69
4.80 Relationsspezifikation: has mechanism . . . . . 70
4.81 Relationsspezifikation: has modellinglanguage . . . . . 70
4.82 Relationsspezifikation: has modeltype . . . . . 70
4.83 Relationsspezifikation: has class . . . . . 70
4.84 Relationsspezifikation: is superclass . . . . . 71
4.85 Relationsspezifikation: has attribute . . . . . 71
4.86 Relationsspezifikation: has notation . . . . . 71
4.87 Relationsspezifikation: from . . . . . 71
4.88 Relationsspezifikation: to . . . . . 72

# Abbildungsverzeichnis


2.1 Modell- und Sprachenebenen (Karagiannis u. Kühn [2002]) . . . . . 5
2.2 Komponenten einer Modellierungsmethode (Karagiannis u. Kühn [2002]) . . . . 6
2.3 Zusammenhänge der Funktionalitäten von Repositories . . . . . 10
2.4 Aufbau und Zugriff eines Datenbanksystems (vgl. Saake u. a. [2008]) . . . . . 17

3.1 Nicht-funktionale Anforderungen (vgl. ISO/IEC [2001], Balzert [1998], Mylopoulos u. a. [1999] und Mairiza u. a. [2010]) . . . . . 20

4.1 Beispiel eines Deadlock . . . . . 38
4.2 Klassenhierarchie des MM-Repository-Metamodells . . . . . 52
4.3 Relationen innerhalb des MM-Repository-Metamodells . . . . . 53
4.4 Use-Case Diagramm . . . . . 73

5.1 Infrastruktur . . . . . 80
5.2 Screenshot: Login-Seite . . . . . 81
5.3 Screenshot: Überblick über die MM-Repositoryfunktionalitäten mit Logout-Möglichkeit . . . . . 81
5.4 Screenshot: Übersicht zum Konfigurationsmanagement . . . . . 85
5.5 Screenshot: Konfiguration anzeigen . . . . . 85
5.6 Screenshot: Konfiguration editieren . . . . . 86

6.1 Struktur der Open Models Initiative (vgl. Karagiannis u. a. [2010]) . . . . . 90
6.2 Screenshot: Auszug aus der iStar (i*)-Methode . . . . . 92
6.3 Screenshot: Diverse Interpretationen der i*-Methode . . . . . 93
6.4 Screenshot: Vorgehen der Semantic Database Design (SDbD)-Methode . . . . . 94

# Listings

5.1 Session überprüfen . . . . . 81
5.2 Login . . . . . 82
5.3 Verbindung zu aServer . . . . . 82
5.4 Abmelden . . . . . 82
5.5 Versionsnummer automatisch erhöhren . . . . . 83
5.6 Liste mit verfügbaren Konfigurationen ausgeben . . . . . 87

# 1 Einleitung

Ein Ziel der Open Models Initiative[1] ist die Entwicklung von Modellierungsmethoden, um domänenspezifische Modelle entwickeln zu können. Daher gewinnt die Verwaltung von Modellierungsmethoden und deren Anwendung immer mehr an Bedeutung. Diese Arbeit befasst sich aus diesem Grund mit einem Repository (MM-Repository), welches für die Verwaltung von Modellierungsmethoden und deren Modellen innerhalb der Open Models Initiative entwickelt wird.

## 1.1 Motivation

Die Verwaltung von Modellierungsmethoden und deren Modellen spielt eine immer wichtigere Rolle, um die Entwicklung dieser besser unterstützen zu können. In diesem Kapitel sollen zuerst die Unterschiede zwischen Evolution, Verwaltung und Versionierung von Modellen und Modellierungsmethoden definiert werden, um ein allgemeines Verständnis für diese Begriffe zu erhalten. In weiterer Folge wird für Modelle und Modellierungsmethoden meist der Überbegriff Objekte verwendet, um eine bessere Lesbarkeit zu erzeugen.

Wenn sich Objekte ändern, werden neue Versionen erstellt, diesen Vorgang nennt man Evolution. Es gibt verschiedene Gründe für die Evolution von Objekten, diese können auf technische Weiterentwicklungen oder fachliche Erkenntnisse zurückgeführt werden. Technische Gründe sind unter anderem Tool-Weiterentwicklungen. Fachliche Gründe können externe oder interne Einflussfaktoren haben. Externe Faktoren sind Änderungen der rechtlichen Grundlagen, der Wettbewerbsbedingungen oder der Kundenbedürfnisse, interne Faktoren sind Änderungen oder Optimierung der Unternehmensstrategie, der Prozesse, der Unternehmenskultur oder Unternehmensstruktur. Daher werden neue Versionen von Objekten erstellt, welche verwaltet werden müssen.

Bei der Verwaltung von diesen Objekten wird ein gemeinsamer Datenbestand, also ein Repository benötigt, in welchem die unterschiedlichen Modelle und Modellierungsmethoden gespeichert und repräsentiert werden.

[1] www.wikimethods.org

Die Versionierung von Objekten umfasst die unterschiedlichen Ergebnisse der Evolution und die Unterstützung zur Erstellung der diversen Versionen. Bei der Versionierung ist vor allem zu beachten, dass es sich dabei meist um Mehr-Benutzer-Systeme handelt, die einen gemeinsamen Zugriff und eine gemeinsame Überarbeitung der Objekte gewährleisten soll.

## 1.2 Problemstellung und Zielsetzung

Dieses Repository für Modellierungsmethoden und deren Modelle soll sowohl Modellierungsmethoden als auch dazugehörige Ressourcen verwalten können. In weiterer Folge wird dieses Repository mit MM-Repository abgekürzt. Dabei beschränkt sich die Verwaltung nicht nur auf beschreibende Daten der unterschiedlichen Objekte, sondern es sollen dem Benutzer auch Implementierungen zur Verfügung gestellt werden, um diese Werkzeuge, Modellierungsmethoden und Modelle verwenden zu können. Dabei ist es wichtig, dass der Zugriff zu jedem Zeitpunkt auf alle vorhandenen Versionen im MM-Repository sichergestellt wird.

Weitere wichtige Aufgaben des MM-Repositories sind die Darstellung der Entwicklung von Modellierungsmethoden und die Unterstützung des Benutzers bei der Versionierung. Dafür sollen dem Benutzer beim Ablegen von Objekten Vorschläge für die Vergabe der Versionsnummern gemacht werden. Durch eine grafische Aufbereitung des Entwicklungsvorganges können die unterschiedlichen Objekte vom Benutzer analysiert werden.

In dieser Arbeit soll gezeigt werden, dass das Metamodellierungskonzept nicht nur für die Entwicklung einer Modellierungsmethode verwendet werden kann, sondern auch für die Umsetzung eines MM-Repositories. Daher wird ein Konzept für ein MM-Repository vorgestellt, dass die unterschiedlichen Inhalte der Open Model Initiative, wie PRojekte, Modellierungsmethoden und deren Modelle, verwaltet. Des Weiteren soll das Konzept auf Basis von ADOxx®[2] umgesetzt werden, um die Vorteile einer Metamodellierungsplattform nutzen zu können.

## 1.3 Aufbau der Arbeit

Die weitere Arbeit gliedert sich in die Kapitel Grundlagen (Kapitel 2), Anforderungen an das Modellierungsmethoden-Repository (Kapitel 3), Modellierungsmethoden-Repository: Konzeption (Kapitel 4), Modellierungsmethoden-Repository: Implementierung (Kapitel 5), Anwendungsbeispiele innerhalb der Open Models Initiative (Kapitel 6) sowie Zusammenfassung und Ausblick (Kapitel 7).

[2] registrierte Marke der BOC AG

Um ein gemeinsames Verständnis zu bekommen, werden in Kapitel 2 (Grundlagen) Modellierung und Metamodellierung sowie Repositories genauer erörtert. Dabei werden Repositories anhand folgender Kriterien analysiert: Definitionen, Funktionalitäten, Eigenschaften, Inhalt und Realisierung. Im Anschluss wird eine Zusammenfassung über vorhandene Repositories dargebracht.

Kapitel 3 definiert sowohl nicht-funktionale als auch funktionale Anforderungen an das MM-Repository.

Der Hauptbestandteil des Repositorykonzepts wird in Kapitel 4 beschrieben. Dies beinhaltet die allgemeinen Eigenschaften des MM-Repositories, Funktionalitäten, externe Schnittstellen, Architektur und Benutzerinteraktion. Des Weiteren wird in diesem Kapitel ein Vergleich zwischen dem vorgestellten Konzept und einer datenbank-basierten Lösung ausgearbeitet.

Die Implementierung dieser Arbeit wird in Kapitel 5 aufgelistet. Dabei wird zuerst die Infrastruktur vorgestellt und anschließend die implementierten Funktionen beschrieben. Diese Funktionalitäten sind die An- und Abmeldung, ein Versionsmechanismus sowie das Konfigurationsmanagement.

Die Verwendung des MM-Repositories wird anhand unterschiedlicher Anwendungsbeispiele innerhalb der Open Models Initiative in Kapitel 6 erläutert.

Den Abschluss bildet Kapitel 7 mit Zusammenfassung und Ausblick dieser Arbeit.

# 2 Grundlagen

In diesem Kapitel werden die Grundlagen für diese Arbeit festgelegt. Daher werden zuerst alle notwendigen Konzepte, wie Modellierung, Metamodellierung und Grundlagen von Repositories erläutert. Die Grundlagen von Repositories werden in Definitionen, Funktionalitäten, Eigenschaften, Inhalte sowie Realisierungsmöglichkeiten unterteilt. Abschließend werden die notwendigen Schritte zur Erstellung eines Repositories definiert.

## 2.1 Modellierung und Metamodellierung

Eine wichtige Basis für diese Diplomarbeit liefert das Metamodellierungskonzept von Karagiannis u. Kühn [2002], da dadurch eine flexible, adaptierbare und für unterschiedliche Modellierungsmethoden offene Plattform zur Verfügung gestellt wird. Dadurch können unterschiedliche Modelle gebildet werden, welche eine Abbildung der Realität darstellt. Wichtige Merkmale von Modellen sind wie bereits erwähnt Abbildung eines Originals, Verkürzung des Originals, da nicht alles abgebildet werden kann, und das pragmatische Merkmal, wodurch einem Modell das Original nicht eindeutig zugeordnet werden kann (Stachowiak [1973]). Diese Definition gilt auch für die Modelle der Informatik, welche hier behandelt werden. Eine weitere Definition von Modell, die in der Informatik und im Speziellen in der Wirtschaftsinformatik verwendet wird, stammt

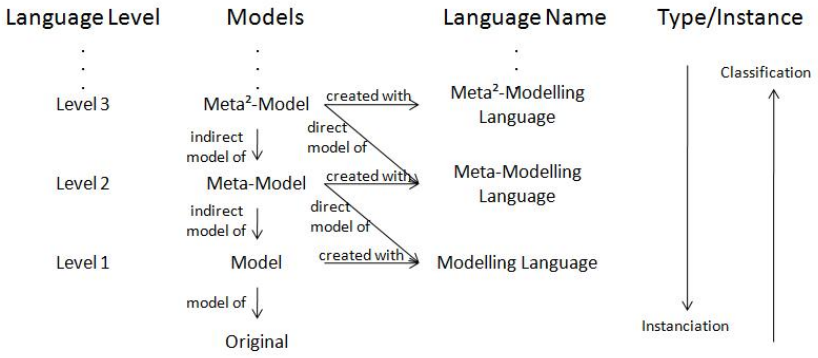



Abbildung 2.1: Modell- und Sprachenebenen (Karagiannis u. Kühn [2002])

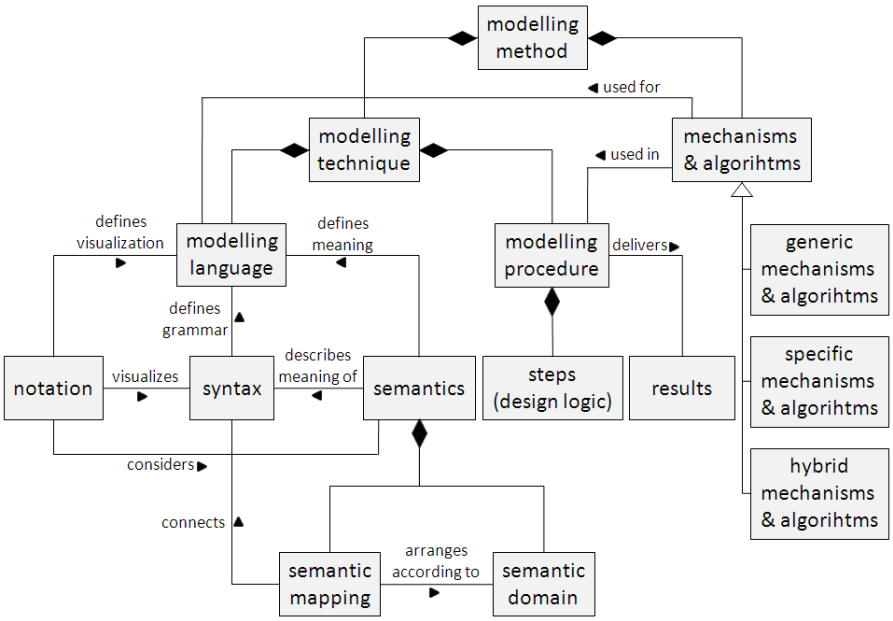



Abbildung 2.2: Komponenten einer Modellierungsmethode (Karagiannis u. Kühn [2002])

von Bézivin [2001] und definiert ein Modell als Vereinfachung eines Systems, welches für ein bestimmtes Ziel gebildet wurde.

Durch eine Modellierungshierarchie (siehe Abbildung 2.1) werden die Zusammenhänge zwischen Original, Modell, Metamodell und Meta$^2$-Modell beschrieben. Modelle, die ein Original abbilden, werden auf Basis einer Modellierungssprache erzeugt. Dieses Modell basiert auf einem Metamodell, welches durch eine Metamodellierungssprache erzeugt wurde usw. Da sich die Abstraktion je öfter man dies weiterführt erhöht, wird meist eine vier-schichtige Architektur gewählt (Karagiannis u. Kühn [2002]).

Karagiannis u. Kühn [2002] unterteilen eine Modellierungsmethode in verschiedene Komponenten, welche in Abbildung 2.2 abgebildet sind. Dabei besteht eine Modellierungsmethode aus den Komponenten Modellierungstechnik sowie Mechanismen und Algorithmen. Modellierungstechnik besteht aus einer Modellierungssprache sowie einer Modellierungsvorgehensweise, welche aus unterschiedlichen Schritten zur Erzeugung von Resultaten durch die Modellierungssprache zusammengesetzt wird. Die unterschiedlichen Mechanismen und Algorithmen, diese sind in generisch, spezifisch und hybrid unterteilt, werden von der Vorgehensweise verwendet und dienen der Anwendung der Modellierungssprache. Die Modellierungssprache wird durch die Notation, Syntax und Semantik definiert. Durch die Notation wird die grafische Repräsentation der Modellierungssprache beschrieben. Die Syntax beinhaltet die Elemente und Regeln für das Erzeugen von Modellen und wird durch eine Grammatik formuliert. Die Semantik drückt die

Bedeutung der Modellierungssprache aus. Die unterschiedlichen Mechanismen und Algorithmen sind für die Modellverwaltung von großer Bedeutung, da dadurch die benötigten Funktionalitäten realisiert werden können. Generische Mechanismen basieren auf dem Meta$^2$-Modell und sind so für alle Modellierungssprachen anwendbar, die auf demselben Meta$^2$-Modell aufbauen. Dies können etwa externe Mechanismen für die Versionierung sein. Spezifische Mechanismen wurden für ein bestimmtes Metamodell implementiert und können so nicht für andere Metamodelle verwendet werden. Dies können etwa Versionierungsattribute, wie Änderungsdatum oder -autor, sein. Die letzte mögliche Form von Mechanismen ist der hybride Ansatz. Diese werden auf Meta$^2$-Modell-Ebene implementiert und auf Metamodell-Ebene adaptiert. Metamodellierung als Konzept wird in Karagiannis [2007] sowie Karagiannis u. Höfferer [2006] genauer betrachtet.

## 2.2 (Modell-)Management

Da es im Bereich Metamodellierung noch wenige Ansätze zur Verwaltung von Modellierungsmethoden gibt, werden nun die ersten Ansätze im Bereich Modellmanagement analysiert, welche bereits in den frühen 80er Jahren entwickelt wurden. Dabei handelt es sich bei diesen verwalteten Modellen meist um eine formale Beschreibung von Metadaten-Artefakten, wie Ontologien, Datenbankschemata oder Objektdiagramme (Melnik [2004]). In Elam u. a. [1980] und Blanning [1982] wurden zum ersten Mal die Unterschiede zum Datenmanagement präsentiert, wobei der größte Unterschied in der Empfindlichkeitsanalyse besteht, um den Entscheidungsfindungsprozess in Unternehmen zu unterstützen. In diesen Arbeiten war jedoch noch immer stark der Bezug zu Datenmanagement ersichtlich.

Die Verwendung von Modellen nimmt noch immer stärker zu, wodurch auch das Modellmanagement noch immer an Bedeutung gewinnt. Hauptsächlich wird beim Modellmanagement von der Verwaltung der Änderungen gesprochen, jedoch darf die Transformation von Modellen in andere Modelle nicht vernachlässigt werden. Bernstein u. a. [2000a] identifizieren und erläutern die wichtigsten Operationen für Modellmanagement, welche in einfache und komplexe Operationen unterteilt werden können. Elementare Operationen dienen unter anderem zur Manipulation der Modelle, wie create, update, delete, select, project, select, setDifference, applyFunction, ModelGen und copy, sowie die Navigation durch Modelle. Zu den komplexen Operationen zählen matching & differencing, merge sowie die Zusammensetzung der Übereinstimmungen (mapping composition) (Bernstein u. a. [2000b]; Bernstein u. Rahm [2000]; Bernstein [2003]). Auf die unterschiedlichen Funktionen wird nicht mehr genauer eingegangen, da diese Auflistung lediglich die Vielfalt an Funktionen ausdrücken soll.

Wichtige Erkenntnisse für die weitere Forschung und Entwicklung von Modellmanagement

stammen aus der Evolution und Mapping von Schemata (Kensche u. a. [2009]; Pottinger u. Bernstein [2008]), der Informations- und Datenintegration (Calvanese u. a. [2009]; Haas u. a. [2009]; Bernstein u. Haas [2008]) sowie aus dem Clio-Projekt, welches im Bereich Datenintegration und Schema Mapping angesiedelt ist(Miller u. a. [2001]; Fagin u. a. [2009]). Ansätze für ein generisches Modellmanagement stammen vor allem von Melnik [2004] sowie Rondo als Programmierplattform für generisches Modellmanagement (Melnik u. a. [2003]) und GeRoMe, ein generisches, rollen-basiertes Metamodell für Modellmanagement (Kensche u. a. [2007b,a]). Im Bereich Modellmanagement ist es jedoch die Feststellung entscheidend, dass Modellmanagement nicht als Ersatz für Datenbanksysteme gesehen werden darf, sondern als Erweiterung dieser (Bernstein u. a. [2000b]).

Anhand dieser Analyse von Modellmanagement können für diese Arbeit im Bereich Modellierungsmethoden folgende essentiellen Anforderungen abgeleitet werden: Evolution und Mapping von Modellierungsmethoden sowie Integration und Transformation von Modellierungsmethoden.

## 2.3 Repositories

Der Begriff Repository wird sehr häufig und für die unterschiedlichsten Datensammlungen verwendet, daher erfolgt in diesem Kapitel eine Definition des Begriffs sowie Analyse von existierenden Repositoryansätzen. Die Hauptunterscheidungsmerkmale von Repositoryansätzen werden in die folgenden Kategorien unterteilt: Funktionalitäten, Eigenschaften, Inhalte und Realisierung.

### 2.3.1 Definitionen von Repositories

Laut Jacobs u. a. [2009] muss ein Repository unterschiedliche spezifische Repository-Funktionalitäten zur Verfügung stellen, um Artefakte definieren und verwalten zu können. Diese Funktionen sind jedoch je nach Anwendungsgebiet und Prioritäten der Anwender unterschiedlich. So werden in Bernstein u. Dayal [1994] sowohl DBMS-Funktionalitäten wie Speicherung, Abfragen, Sichten, Integritätskontrolle, Zugriffskontrolle und Transaktionsmanagement als auch zusätzliche Repository-Funkionalitäten wie Checkout-/Checkin-Management, Versionskontrolle, Konfigurationskontrolle, Benachrichtigungen, Kontextmanagement und Lifecyclemanagement angeführt.

In Zivkovic u. a. [2009] wird eine ontology-aware Metamodellierungsplattform vorgestellt, die auf einige wichtige Funktionalitäten eines Modell-Repositories eingeht. Allgemeine Funktio-

nen die erwähnt werden sind Import-/Exportfunktionalität, Sicherheitsfunktionen sowie eine Web-Schnittstelle zu anderen Systemen. Spezielle Funktionen für Modell-Repositories sind etwa integrierte Abfragen, Metamodell-Mapping, Modell Transformation, Modell Merge, Diff, Copy und Global Change, Modell Versionierung, Modell Validierung, Gewährleistung von Nachvollziehbarkeit bei Änderungen und Durchführen von unterschiedlichen Mechanismen. Es werden jedoch auch andere Funktionen und Komponenten genannt, die für die weitere Arbeit nicht relevant sind und deshalb weggelassen werden.

Eine weitere Definition beschreibt ein Repository als verteilte Datenbank mit Informationen über technische Artefakte, welche von einer Organisation produziert oder verwendet werden (Bernstein u. Dayal [1994]). Dabei ist zu beachten, dass diese Artefakte für andere Applikationen und Systeme zur Verfügung gestellt werden und von einem Menschen erzeugt wurden.

### 2.3.2 Funktionalitäten von Repositories

Repositories müssen sowohl allgemeine DBMS-Funktionalitäten als auch spezifische Repository-Funktionalitäten zur Verfügung stellen. Zu den DBMS-Funktionalitäten zählen unter anderem Datenverwaltung, Datenabfragen, Datensichten, Integritätsmanagement, Zugriffsmanagement und Transaktionsmanagement. Spezifische Funktionalitäten des Repositories sind unter anderen Checkout-/Checkin-Management, Versionsmanagement, Konfigurationsmanagement, Benachrichtigungsmanagement, Kontextmanagement und Lifecyclemanagement (Yan u. a. [2009]).

Obwohl die nachfolgende Auflistung die Funktionalitäten des Repositories trennt, können die Funktionalitäten nicht getrennt voneinander betrachtet werden und weisen Abhängigkeiten untereinander auf, welche in Abbildung 2.3 dargestellt werden.

Die nachfolgende Auflistung von Funktionalitäten ist nicht vollständig, sondern lediglich ein Auszug der Funktionalitäten, die in der Literatur und Praxis am häufigsten angeboten werden.

#### Datenverwaltung

Aufgaben der Datenverwaltung sind das Anlegen, Aktualisieren und Löschen von Daten des Systems.

#### Datenabfragen und -sichten

Mithilfe von Abfragen und Sichten kann der Inhalt des Systems durchsucht und dargestellt werden. Dabei soll es die Möglichkeit geben, individuelle Abfragen erstellen zu können.

| | Datenverwaltung | Datenabfragen und -sichten | Integritätsmanagement | Zugriffsmanagement | Transaktionsmanagement | Locking-Management | Versionsmanagement | Konfigurationsmanagement | Benachrichtigungsmanagement | Kontextmanagement | Lifecyclemanagement |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Datenverwaltung | | | x | x | x | x | x | | x | | x |
| Datenabfragen und -sichten | | | x | x | x | x | | x | x | x | x |
| Integritätsmanagement | x | x | | x | x | x | x | x | x | x | x |
| Zugriffsmanagement | x | x | x | | x | x | x | x | x | x | x |
| Transaktionsmanagement | x | x | x | x | | x | x | x | x | x | x |
| Locking-Management | x | x | x | x | x | | x | x | x | x | x |
| Versionsmanagement | x | | x | x | x | x | | x | x | x | x |
| Konfigurationsmanagement | | x | x | x | x | x | x | | x | | x |
| Benachrichtigungsmanagement | x | x | x | x | x | x | x | x | | x | x |
| Kontextmanagement | | x | x | x | x | x | x | | x | | x |
| Lifecyclemanagement | x | x | x | x | x | x | x | x | x | x | |

Abbildung 2.3: Zusammenhänge der Funktionalitäten von Repositories

### Integritätsmanagement

Durch das Integritätsmanagement wird sichergestellt, dass die Daten vom Benutzer stammen und unverändert im System verwendet werden.

### Zugriffsmanagement

Das Zugriffsmanagement legt fest, welche Benutzer mit welchen Rechten auf das System zugreifen können. Dabei können die Rechte für unterschiedliche Aufgaben sowie Objekte vergeben werden. Eine weitere Aufgabe des Zugriffsmanagement ist das An- und Abmelden dieser Benutzer. Dadurch wird sichergestellt, dass nur berechtigte Benutzer Zugriff auf die unterschiedlichen Funktionen und Inhalte des Systems haben.

### Transaktionsmanagement

Mehrere Funktionsaufrufe innerhalb des Repositories können zu einer Transaktion zusammengefasst werden. Dies bedeutet, dass alle Funktionen innerhalb einer Transaktion entweder ganz oder gar nicht ausgeführt werden müssen. Falls ein Fehler in einer Funktion der Transaktion

auftritt müssen alle bereits durchgeführten Funktionen rückgängig gemacht werden. Dadurch wird gewährleistet, dass das System immer in einem konsistenten Zustand ist.

#### Locking-Management und Checkout-/Checkin-Management

Bei Mehrbenutzer-Repositories ist es notwendig, dass Objekte, die von einem Benutzer verwendet werden, gesperrt werden, um inkonsistente Zustände im Repository zu verhindern. Beim Checkout eines Objekts wird dieses gesperrt. Die Sperre wird beim Checkin eines Objekts wieder aufgehoben. Dabei gibt es zwei unterschiedliche Arten des Sperren, pessimistisches und optimistisches Sperren. Beim pessimistischen Sperren kann das Objekt von keinem anderen Benutzer verwendet werden. Daher muss der Benutzer warten bis die Sperre aufgehoben ist. Beim optimistischen Sperren kann ein weiterer Benutzer das Objekt ebenfalls verwenden. Wenn die Objekte wieder ins Repository geladen werden muss jedoch überprüft werden, ob Konflikte aufgetreten sind. Falls Konflikte vorhanden sind müssen diese vom Benutzer oder System behandelt werden.

#### Versionsmanagement

In einem Repository muss es weiters möglich sein unterschiedliche Versionen eines Objekts verwalten zu können. Für die Verwaltung dieser Versionen wird ein Versionsmanagement benötigt. Mögliche Aufgaben für die Verwaltung von Versionen sind die Vergabe von Versionsbezeichnungen (Versionsnummern), Repräsentation der Versionen, Ableitung von neuen Versionen anhand vorhandener Objekte, Relationen zwischen versionierten Objekten darstellen sowie Erzeugen von Entwicklungszweigen (Branches) und Zusammenführen von Versionen.

#### Konfigurationsmanagement

Mithilfe des Konfigurationsmangements können unterschiedliche Versionen von Objekten zusammengefasst werden. Diese Konfigurationen können dadurch repräsentiert und verändert werden. Dabei können Objekte zu einer Konfiguration hinzugefügt oder entfernt werden. Die Veränderungen einer Konfiguration müssen ebenfalls repräsentiert werden.

#### Benachrichtigungsmanagement

Da Objekte innerhalb eines Repositories voneinander abhängig sein können, muss es möglich sein, die abhängigen Objekte von Änderungen zu informieren um gegebenfalls Operationen

durchführen zu können. Eine Operation auf ein Objekt kann Operationen auf andere Objekte benötigen. Ein Beispiel hierfür wäre die Löschweitergabe von zusammenhängenden Objekten. Des Weiteren können Benutzer benachrichtigt werden, wenn andere Benutzer dasselbe Objekt bearbeiten. Konfigurationen müssen benachrichtigt werden, wenn sich Objekte der Konfiguration geändert haben. Bei Versionen kann es notwendig sein, dass Nachfolgerversionen benachrichtigt werden, wenn sich die Vorgängerversion ändert. Diese Benachrichtigungsregeln müssen konfigurierbar sein und sollen um weitere Regeln erweitert werden können.

#### Kontextmanagement

Durch das Kontextmanagement können Objekte und bestimmte Attribute zusammengefasst werden. Dadurch können nur die für einen Anwendungsfall notwendigen Informationen der Objekte aufgelistet werden. Hier ist es wichtig, dass der Kontext persistent gespeichert wird.

#### Lifecyclemanagement

Objekte eines Repositories können unterschiedliche Phasen im Lebenszyklus durchlaufen. Mithilfe des Lifecyclemanagements soll es möglich sein, diese Phasen zu definieren und den Status der Objekte zu ändern.

### 2.3.3 Eigenschaften von Repositories

Repositories werden in den unterschiedlichsten Anwendungsgebieten und mit den unterschiedlichsten Eigenschaften verwendet. Zu den Anwendungsgebieten zählen unter anderem: Medienarchivierung (Beer u. a. [2009]), Verwaltung, Analyse und Verwendung von Prozessmodellen (La Rosa u. a. [2009]), Refactoring von Prozessmodellen (Weber u. Reichert [2008]), Speicherung und Abfragen von XML-Dokumente (Alkhatib u. Scholl [2008]), Zusammenarbeit in verteilten Dokumenten (Ignat u. Norrie [2006]), Analyse und Überwachung von Service-basierten Systemen (Holmes u. a. [2010]), Verwaltung von Lerninhalten (Millard u. a.) sowie Analyse von Proteinstrukturen (Kiefer u. a. [2009]). Es ist jedoch zu beachten, dass diese Auflistung lediglich ein Auszug der Möglichkeiten zur Verwendung von Repositories liefert. Dies soll zeigen, dass die Einsatzgebiete von Repositories innerhalb der Informationsverarbeitung beinahe unbegrenzt ist.

Durch die unterschiedlichen Anwendungsbereiche ergeben sich ebenfalls sehr unterschiedliche Eigenschaften von Repositories, die nun kurz erläutert werden.

#### Zentralisiertes Repository

Dabei wird dem Benutzer nur ein Zugangspunkt zum Repository bereitgestellt (Palanisamy u. a. [2010]).

#### Online Repository

Der Zugriff auf das Repository erfolgt über das Internet.

#### Metadaten-Repository

In diesem Repository werden Metadaten zu einem Objekt und eine Referenz zu diesem Objekt gespeichert. Dabei wird das Objekt nicht in das Repository gespeichert.

#### Objekt-Repository

In einem Objekt-Repository werden sowohl Informationen zu dem Objekt, also Metadaten, als auch das Objekt selbst im Repository abgelegt. In de Sompel u. a. [2005] wird ein Beispiel für ein Digital Object Repository beschrieben.

#### Heterogenes Repository

Dabei muss unterschieden werden, ob die Daten oder die Datenquellen nicht einheitlich sind. Bei heterogenen Daten muss das Datenmodell so generisch gehalten werden, dass alle Daten abgebildet werden können. Wenn es sich um unterschiedliche Datenquellen handelt, müssen Integrationsmechanismen zur Verfügung gestellt werden. So beschäftigt sich Baralis u. Fiori [2008] mit heterogenen biologischen Daten und Datenquellen.

#### Homogenes Repository

Im Gegensatz zu heterogenen Repositories beinhalten homogene Repositories gleich aufgebaute Daten und Datenquellen.

#### Verteilte Datenquellen

Ein Repository kann unterschiedliche Datenquellen haben (Palanisamy u. a. [2010]).

#### Weitere Eigenschaften

Bei der Entwicklung eines Repositories spielen weitere Eigenschaften eine zentrale Rolle. So müssen Überlegungen zu den folgenden Punkten angestellt werden: Größe, Skalierbarkeit, Erweiterbarkeit (des Datenmodells), Flexibilität sowie Vertrauenswürdigkeit. Aufgrund dieser Eigenschaften werden unterschiedliche Implementierungen oder Funktionalitäten benötigt.

### 2.3.4 Inhalt von Repositories

Inhalte von Repositories können ebenfalls sehr unterschiedlich sein. So können alle beliebige Arten von Objekten und Daten in einem Repository gespeichert und mit unterschiedlichen Funktionalitäten verwendet werden. Diese Inhalte können folgendermaßen eingeteilt werden: Dokumente, Informationen, Modelle, Wissen, mathematische Inhalte, Multimedia Inhalte, Software, Services, Events, Lernobjekte, Publikationen und vieles mehr.

Hierbei ist es wichtig, dass es unterschiedliche Möglichkeiten gibt, diese Inhalte zu beschreiben, um unterschiedliche Funktionalitäten, wie Suche, ausführen zu können. Diese Formate, für die Speicherung von zusätzlichen, beschreibenden Informationen, wurden für einige Objekte genauer betrachtet. Diese sind Dokumente und Objekte als übergreifende Kategorie sowie die Multimedia Objekte wie Bilder, Videos und Ton.

#### Dokumente und Objekte

Die wichtigsten und am häufigsten verwendeten Formate für die Speicherung von Objekten und Dokumenten in einem Repository sind Dublin Core® (Park u. Childress [2009]; Apps u. Macintyre [2000]), Really Simple Syndication (RSS)[3] und BibTex-Format (Patashnik [2010]). Dublin Core® kann für die Beschreibung unterschiedlichster Dokumente verwendet werden. Elemente davon sind unter anderem Titel, Ersteller, Subjekt, Beschreibung, Herausgeber, Mitwirkende, Datum, Typ, Format, Identifier, Quelle, Sprache, Beziehung, Umfang und Rechte. RSS wird für die Beschreibung von Neuigkeiten von Webseiten verwendet. Wobei eine Frequenz (Channel) einen Titel, Link zur Homepage, Beschreibung, Sprache, Autor, Erstellungsdatum und Bild beinhaltet. Innerhalb einer Frequenz (Channel) befinden sich Einträge, die einen Titel, Link zum Eintrag, Beschreibung, Autor, Identifier und Erstellungsdatum haben. Das BibTex-Format wird für die Beschreibung von Literatur wie Artikel, Buch, Druckwerk, Konferenz, technische Dokumentation, Abschlussarbeit (z.B. Diplomarbeit, Promotionsarbeit), Konferenzbericht, veröffentlichter Bericht einer Institution und unveröffentlichtes Dokument verwendet. Diese

[3] http://www.rssboard.org/rss-specification

Einträge bestehen aus diversen Elementen, wie Autor, Titel, Jahr, Herausgeber, Buchtitel, Journal, Ausgabe, Adresse, International Standard Book Number (ISBN), Notizen uvm. Eine vollständige Liste der Literaturarten und der Elemente finden sie in Patashnik [2010]. In Haslhofer u. Klas [2010] werden die unterschiedlichen Metadaten-Formate und -Standards genauer untersucht und können als Erweiterung der hier aufgeführten Metadaten-Formate gesehen werden.

Weitere Formate sind unter anderem Resource Description Framework (RDF)[4], PREservation Metadata Implementation Strategies (PREMIS)[5], Metadata Encoding and Transmission Standard (METS)[6], MAchine-Readable Cataloging (MARC)[7], Encoded Archival Description (EAD)[8] und Learning Objects Metadata (LOM) (Hodgins u. Duval [2002]). Es ist zu berücksichtigen, dass diese Liste von Formaten lediglich ein Auszug aus den möglichen Formaten für die Speicherung von Informationen zu Dokumenten und Objekten ist.

#### Multimedia Objekte

Unter dem Begriff Multimedia Objekte werden in dieser Arbeit Objekte, wie Videos, Bilder und Ton zusammengefasst. Im Bereich Multimedia gibt es die unterschiedlichsten Standards von Formaten für zusätzliche Informationen. Daher soll diese Auflistung einen Überblick über die wichtigsten Informationsbeschreibungen in Bezug auf Reposities geben. Video- und Audio-Inhalte können mittels Moving Picture Experts Group (MPEG) beschrieben werden. Der von Adobe Systems entwickelte Standard für Metadaten, Extensible Metadata Platform (XMP) (Incorporated [2005]), wird häufig für die Beschreibung von Bildern und Fotografien verwendet, wobei XMP die Metadaten in die Originaldatei einbettet. Im Umfeld von Repositories wird ebenfalls das Exchange Broadcast Binary and Metadata Format (XBMF)[9] für die Beschreibung von Multimedia Objekten öfters verwendet. Für die Beschreibung von Audiodateien können die Informationen des ID3-Tags von MP3[10]-Dateien verwendet werden, welche Elemente wie Songtitel, Interpret, Album, Erscheinungsjahr, Kommentar und Genre beinhaltet. Audio-Dateien wurden jedoch innerhalb von Repositories selten verwendet.

[4] http://www.w3.org/RDF/
[5] http://www.loc.gov/standards/premis/
[6] http://www.loc.gov/standards/mets/
[7] http://www.loc.gov/marc/
[8] http://www.loc.gov/ead/
[9] http://sotf.sourceforge.net/documents/XBMF_V0-31.pdf
[10] Standards: ISO/IEC 11172-3, ISO/IEC 13818-3

### 2.3.5 Realisierung von Repositories

Die Realisierung von Repositories ist ebenfalls sehr unterschiedlich und unterscheidet sich konzeptionell in daten-basierte, datenbank-basierte und modell-basierte Repositories. Daten-basierte Repositories können entweder Extensible Markup Language (XML)-Dateien oder andere strukturierte Dateien enthalten, welche durch unterschiedliche Mechanismen und Funktionalitäten zur Verfügung gestellt werden. Datenbank-basierte Repositories verwenden entweder relationale, objekt-relationale oder objekt-orientierte Datenbanken als Basis für die Speicherung von Inhalten. Bei modell-basierten Repositories wird der Metamodellierungsansatz verwendet, so können hier unterschiedliche Meta$^2$-Modelle wie Meta Object Facility (MOF)[11] oder ADOxx® als Basis verwendet werden.

Für die Umsetzung von Repositories wird meist der datenbank-basierte Ansatz gewählt. Daher wird nun genauer auf die Begriffe Datenbank, Datenbanksystem und Datenbankmanagementsystem eingegangen.

#### Datenbank, Datenbanksystem und Datenbankmanagementsystem

Bei einer Datenbank (DB) handelt es sich um eine Kollektion von gespeicherten, operationalen Daten, die von einem Anwendungssystem verwendet werden (Engles [1970]). Diese Definition bezieht sich vor allem darauf, dass es sich um konkrete Daten handelt und diese von einem bestimmten System verwendet werden. Die DB an sich beinhaltet lediglich Daten, wobei ein Datenbanksystem (DBS) aus einer DB und einem Datenbankmanagementsystem (DBMS) besteht (Saake u. a. [2008]). Der Aufbau eines Datenbanksystems und der Zugriff der unterschiedlichen Anwendungen wird in Abbildung 2.4 dargestellt. Die Datenmodelle einer DB können entweder hierarchisch, relational, objektrelational oder objektorientiert sein. Die Struktur einer hierarchischen Datenbank ist baumartig, jedoch wird dieses Datenmodell kaum eingesetzt. Objektrelationale Datenbanken haben eine relationale Datenstruktur, welche um objektorientierte Eigenschaften ergänzt wurde. Durch objektorientierte Programmiersprachen wurden auch objektorientierte Datenbanken entwickelt, welche sich jedoch noch nicht durchgesetzt haben. Am häufigsten werden weiterhin relationale Datenmodelle verwendet. Das DBMS stellt verschiedene Funktionalitäten, wie Datenzugriff, Sicherheit, Datenintegrität, Transaktionsverwaltung zur Verfügung. Daraus ergeben sich einige Vorteile bei der Verwendung eines DBMS. Diese sind Redundanzkontrolle, Zugriffsbeschränkungen, persistente Speicherung von Programmobjekten und Datenstrukturen, Ableitungen und Aktionen anhand von Regeln, Mehrbenutzermöglichkeit, Darstellung komplexer Beziehungen zwischen Daten, Datenintegrität, Sicherheit und Recovery

[11] entwickelt von Object Management Group (OMG)

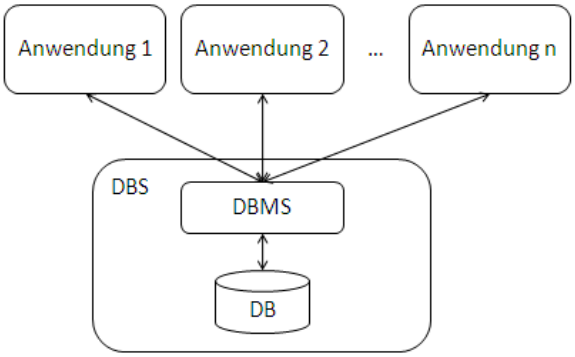



Abbildung 2.4: Aufbau und Zugriff eines Datenbanksystems (vgl. Saake u. a. [2008])

(Elmasri u. Navathe [2009], Kemper u. Eickler [2009], Meier [2010]). Durch den Einsatz von Datenbanken wird die Nutzung von Standards erleichtert, wodurch die Entwicklungszeiten von Anwendungen gekürzt werden. Des Weiteren führen Datenbanksysteme zu einer höheren Flexibilität und Verfügbarkeit der Anwendungen. Dies steigert wiederum die Wirtschaftlichkeit der Anwendungen.

Steiner definiert eine Datenbank als eine selbständige und auf Dauer ausgelegte Datenorganisation, welche einen Datenbestand sicher und flexibel verwalten kann (Steiner [2006]). Dadurch ergeben sich unterschiedliche Aufgaben einer Datenbank, wie der Zugriff auf die gespeicherten Daten, Zugriffsbeschränkungen für die Gewährleistung der Sicherheit sowie die Möglichkeit zur Änderung der internen Datenorganisation, ohne Anpassungen auf Benutzerseite vornehmen zu müssen. In Steiner [2006] werden Datenbankgrundsätze definiert, die teilweise auch in weiterer Folge für andere Umsetzungsmöglichkeiten wie Repositories verwendet werden können. Eine redundanzfreie Struktur oder eine Struktur mit kontrollierten Redundanzen führt zu einem übersichtlichen und konsistenten System. Anwendungen müssen datenunabhängig funktionieren und können auf bestehenden Daten entwickelt werden. Des Weiteren müssen Datenintegrität und Zugriffsbeschränkungen gewährleistet werden.

## 2.4 Zusammenfassung

Aufgrund der Analyse der existierenden Repositoryansätze kann gesagt werden, dass die Definition des Datenmodells eine zentrale Rolle bei der Entwicklung eines Repositories einnimmt. Welche Funktionalitäten ein Repository anbietet ist sehr stark vom Anwendungsfall abhängig, wobei die Verwaltung und das Suchen von Informationen und Objekten innerhalb des Repositories eine zentrale Rolle spielen.

Für die Entwicklung eines Repositorykonzepts ist es daher wichtig folgende Entscheidungen zu treffen:

- Welche Objekte, Daten und Metadaten müssen gespeichert werden? Wie sieht das Datenmodell aus?
- Welche Aufgaben muss das Repository erfüllen? Über welche Eigenschaften soll das Repository verfügen?
- Welche Benutzer verwenden das Repository? Über welche Kenntnisse verfügen diese?
- Welche Technologie wird verwendet?

Aus diesen Fragestellungen können nun folgende Schritte für die Einführung eines Repositories abgeleitet werden:

1. Identifizieren der Anwendungsfälle,
2. Ableiten der nicht-funktionalen und funktionalen Anforderungen an das Repository,
3. Treffen von Entscheidungen, wie Definition der Eigenschaften, Metadaten-Formate und Technologie,
4. Erstellen von beschreibenden Modellen und Diagrammen, wie Metamodell für die Definition der Datenstruktur, Interaktionsdiagramm für die Darstellung der Zusammenhänge und Architekturdiagramm für den technischen Überblick,
5. Definition der Repository-Funktionalitäten,
6. Definition der Services für den externen Zugriff der Benutzer sowie
7. Implementierung des erstellten Konzepts.

# 3 Anforderungen an das Modellierungsmethoden-Repository (MM-Repository)

In Karagiannis u. a. [2010] wurden bereits funktionale Anforderungen an ein Modell-Repository definiert. Diese waren die Auflistung und Navigation durch die Modelle und Modellierungsmethoden sowie eine Suchfunktionalität. Eine genauere Spezifikation der Anforderungen wurde jedoch noch nicht vorgenommen. Aus diesem Grund werden in diesem Kapitel die Anforderungen an ein MM-Repository genauer betrachtet und auf Basis vorhandener Repositories sowie den Anforderungen der Open Models Initiative erweitert. Die Anforderungen gliedern sich in nicht-funktionale und funktionale Anforderungen. Dabei ist zu beachten, dass die Anforderungen aus Sicht des Benutzers definiert werden und noch keine Informationen zur Umsetzung des MM-Repositories beinhalten.

## 3.1 Nicht-funktionale Anforderungen

Es gibt zahlreiche Anforderungen, die das MM-Repository erfüllen muss. In diesem Kapitel werden die nicht-funktionalen Anforderungen des MM-Repositories erörtert. Quellen für diese Auflistung der Anforderungen sind stark an die Anforderungen an Softwaresysteme angelehnt. Diese sind Mairiza u. a. [2010], [Balzert, 1998, Seite 258ff], Mylopoulos u. a. [1999] und Mylopoulos u. a. [1992]. Mairiza u. a. [2010] beschäftigt sich vor allem mit Definition und Terminologie von nicht-funktionalen Anforderungen, den unterschiedlichen Typen sowie einer Zuordnung der Typen zu den Anwendungssystemen. Balzert [1998] beinhaltet Software Qualitätsmerkmale nach ISO 9126 (ISO/IEC [2001]), welche sich in sechs Hauptmerkmale unterteilen lassen. Mylopoulos u. a. [1999] beschäftigt sich mit der Entwicklung einer zielorientierten Anforderungsanalyse, die sich aus einer nicht-funktionalen Anforderungsanalyse, einer funktionalen Anforderungsanalyse und einer Konfliktanalyse zusammensetzt, wobei aus diesem Ansatz vor allem die nicht-funktionalen Anforderungen für diese Arbeit relevant sind. Aus Mylopoulos u. a. [1992] wurden ebenfalls vor allem die nicht-funktionalen Anforderungen extrahiert und in diese Arbeit eingebettet.

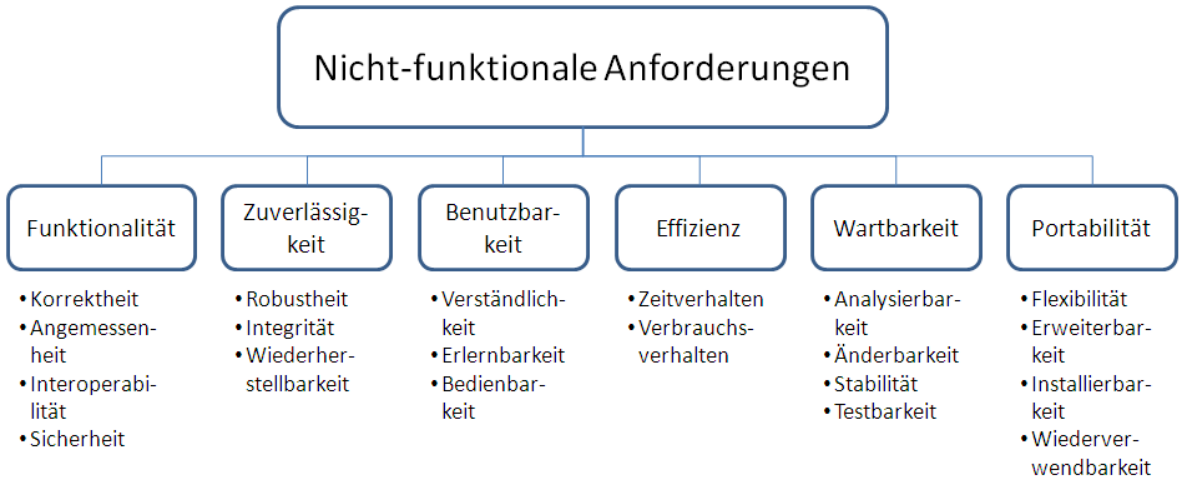



Abbildung 3.1: Nicht-funktionale Anforderungen (vgl. ISO/IEC [2001], Balzert [1998], Mylopoulos u. a. [1999] und Mairiza u. a. [2010])

Da es keine komplette Liste von nicht-funktionale Anforderungen gibt (Mylopoulos u. a. [1992]), werden hier nicht-funktionalen Anforderungen an das Repository diskutiert. Einen Überblick über die Anforderungen wird in Abbildung 3.1 dargestellt. Die Anforderungen oder auch Qualitätsmerkmale/-attribute genannt, werden in die sechs Hauptmerkmale unterteil: Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Wartbarkeit und Portabilität. Diese werden nun in den nachfolgenden Unterkapiteln genauer erörtert.

### 3.1.1 Funktionalität

Das Qualitätsmerkmal Funktionalität soll nicht beschreiben was das MM-Repository machen soll, sondern beinhaltet Anforderungen zu der Funktionsweise des MM-Repositories, wie Korrektheit, Angemessenheit, Interoperabilität und Sicherheit. Die Korrektheit beschreibt, dass die Funktionen das richtige bzw. gewünschte Ergebnis liefern müssen. Diese gelieferten Ergebnisse können mit den gewünschten Ergebnissen verglichen werden, wenn diese übereinstimmen ist die Korrektheit für die getestete Funktion in einem bestimmten Fall gegeben. Die Korrektheit einer Funktion bzw. eines Systems kann jedoch nicht 100-prozentig gewährleistet werden, da nicht alle möglichen Eingaben und Verhaltensweisen des Benutzers durchgeführt werden können. Daher muss ein geeignetes Maß an Testfällen in einem Testplan definiert werden, um die Korrektheit zu einem bestimmten Maß zu überprüfen.

Angemessenheit im Bezug auf die Funktionalität eines MM-Repositories bedeutet, dass die benötigten Funktionen, im gewünschten Umfang bereitgestellt werden. Daher ist es notwendig, dass der Umfang der Funktionalität des MM-Repositories definiert wird und sich die Umsetzung

der Funktionen danach richtet. Durch die Definition der Use Cases und der funktionalen Anforderungen an das MM-Repository kann die Angemessenheit der Funktionsumfangs bestimmt werden.

Durch die Interoperabilität soll gewährleistet werden, dass das MM-Repository mit anderen Systemen kommunizieren kann. Daher muss das MM-Repository ein geeignetes Datenformat für den Export von Objekten zur Verfügung stellen. Des Weiteren können in das MM-Repository beliebige Formate der Objekte gespeichert werden, wodurch Interoperabilität gewährleistet wird.

Unter der nicht-funktionalen Anforderung Sicherheit wird vor allem der berechtigte Datenzugriff verstanden. Dies bedeutet, dass das MM-Repository unberechtigten Zugriff auf die gespeicherten Objekte verhindern muss. Daher benötigt das System sowohl ein User- als auch Zugriffsmanagement.

Die angeführten nicht-funktionalen Anforderung weisen eine hohe Priorität auf, da die Funktionsweise des MM-Repositories davon abhängig ist.

### 3.1.2 Zuverlässigkeit

Zu Zuverlässigkeit zählen Robustheit, Integrität und Wiederherstellbarkeit. Unter Robustheit versteht man, dass das MM-Repository auch bei hoher Auslastung und bei unvorhergesehen Situationen, wie fehlerhaften Verhalten von Benutzern, das gewünschte Ergebnis liefert. Die Robustheit kann durch geeignete Testszenarien überprüft werden.

Die Integrität gewährleistet, dass die Daten unverändert und vollständig vom Benutzer im MM-Repository angelangt sind und auch diese unveränderten und vollständigen Daten verwendet werden. Dies kann durch Prüfsummen bei der Übertragung realisiert werden.

Auch die Wiederherstellbarkeit des MM-Repositories ist eine wesentliche nicht-funktionale Anforderung. Da das MM-Repository nach einem Systemfehler oder einem Angriff wiederhergestellt werden muss. Die Wahrscheinlichkeit eines Systemfehlers oder eines Angriffs muss jedoch zuvor auf ein Minimum reduziert werden.

Zuverlässigkeit ist zwar ein notwendige Anforderung, wird aber im ersten Schritt der Umsetzung nur beschränkt beachtet, da es sich im ersten Schritt um eine prototypische Implementierung des MM-Repositories handelt. In weiterer Folge sind diese nicht-funktionalen Anforderungen nicht außer Acht zu lassen.

### 3.1.3 Benutzbarkeit

Die Benutzbarkeit oder Usability des MM-Repositories ist ebenfalls eine nicht-funktionale Anforderung. Darunter versteht man eine einfache Verständlichkeit, Erlernbarkeit und Bedienbarkeit des MM-Repositories. Benutzer sollen das MM-Repository ohne erheblichen Einschulungsaufwand verwenden können. Dabei ist es wichtig, dass die unterschiedlichen Funktionen des MM-Repositories verständlich beschrieben sind und die Bedienbarkeit für Benutzer intuitiv und attraktiv gestaltet wird.

Auf die Benutzbarkeit des MM-Repositories muss bereits in den ersten Schritten der Entwicklung geachtet werden, da eine nachträgliche Erhöhung der Benutzbarkeit nur mit erheblichen Mehraufwand möglich ist.

### 3.1.4 Effizienz

Unter dem Hauptmerkmal Effizienz werden die Qualitätsmerkmale Zeitverhalten und Verbrauchsverhalten zusammengefasst. Dabei soll die Bearbeitungszeit bzw. Antwortzeit des MM-Repositories möglichst gering gehalten werden. Das Zeitverhalten kann durch diverse Testszenarien, wie Load- und Stresstests, überprüft werden. Das Verbrauchsverhalten bezieht sich unter anderem auf Central Processing Unit (CPU)-Auslastung, Festplattenzugriffe und Arbeitsspeicher.

Effizienz ist in der ersten Phase der Umsetzung des MM-Repositories nicht relevant.

### 3.1.5 Wartbarkeit

Zur Wartbarkeit zählt vor allem die Analysierbarkeit, Änderbarkeit, Stabilität und Testbarkeit des MM-Repositories. Die Analysierbarkeit des MM-Repositories beschreibt den Aufwand, um Fehler im MM-Repository zu entdecken und die betroffene Funktion im MM-Repository zu definieren. Hierbei ist es essentiell, dass sowohl während der Entwicklung die Dokumentation ausführlich durchgeführt wird als auch nach der Inbetriebnahme die Funktionsweise des MM-Repositories protokolliert wird. Diese Dokumentation kann sowohl den Entwicklern bei der Weiterentwicklung, als auch den Benutzern bei der Verwendung des MM-Repositories behilflich sein.

Wenn ein Fehler aufgetreten ist, sollte diese mit möglichst geringem Aufwand behoben werden. Dieser Aufwand wird durch die Änderbarkeit beschrieben. Auch zusätzliche Erweiterungen sollen mit möglichst wenig Aufwand hinzugefügt werden können.

Durch eine geeignete Architektur kann auch die Stabilität des MM-Repositories erhöht werden, da dadurch die Wahrscheinlichkeit des Auftretens von unvorhergesehenem Verhalten des MM-Repositories minimiert wird. Des Weiteren soll das MM-Repository auch im Fehlerfall nicht abstürzen und eine entsprechende Fehlermeldung zurückliefern.

Die Testbarkeit des MM-Repositories ist ein weitere nicht-funktionale Anforderung. Dabei soll sowohl der Aufwand für die Definition als auch für die Ausführung von Testszenarien möglichst gering gehalten werden und eine automatisierte Durchführung der Tests möglich sein. Es muss ein angemessenes Maß zwischen Aufwand für Definition und Ausführung gefunden werden. Die Wiederholung von Testläufen mit unterschiedlichen Parametern ist ebenfalls notwendig.

Zusammenfassend kann man sagen, dass für die Wartbarkeit des MM-Repositories sowohl eine ausführliche Dokumentation als auch eine geeignete Architektur notwendig ist. Die Wartbarkeit muss ebenfalls bereits in den ersten Phasen der Entwicklung beachtet werden, da dadurch nachträgliche Änderungen einfacher zu bewältigen sind.

### 3.1.6 Portabilität

Die Portabilität beinhaltet Anforderungen wie Flexibilität, Erweiterbarkeit, Installierbarkeit und Wiederverwendbarkeit. Durch die Flexibilität soll das MM-Repository an die unterschiedlichen Benutzerbedürfnisse einfach angepasst werden können. Erweiterungen erhöhen den Funktionsumfang des MM-Repositories. Die Erweiterbarkeit wird durch eine ausführliche Dokumentation sowie durch die Einhaltung von Programmier- und Verhaltensrichtlinien erleichtert. Das MM-Repository soll einfach von geeigneten Benutzern installiert werden können. Dabei ist eine Dokumentation bzw. Benutzeranleitung wichtig, um die Installation unterstützen zu können. Dadurch kann ebenfalls die Wiederverwendung unterstützt werden.

Um diese Anforderungen erfüllen zu können, müssen bereits zu Beginn die Grundlagen getroffen werden. Diese sind vor allem Dokumentation und Einhaltung von Richtlinien.

## 3.2 Funktionale Anforderungen

Die funktionalen Anforderungen werden aus Benutzersicht beschrieben und beinhalten die Funktionalitäten des MM-Repositories. Daher werden in den nachfolgenden Kapitel die funktionalen Anforderungen genauer beschrieben, um im Anschluss die Funktionen des MM-Repositories ableiten zu können.

### 3.2.1 Verwaltung von Objekten

Die Verwaltung von Objekten ist in folgende Teile gegliedert: unterschiedliche Arten von Objekten und Operationen, die angewendet werden können.

#### Arten von Objekten

Die unterschiedlichen Arten von Objekten und deren Zusammenhänge wurden aus den Bestandteilen einer Modellierungsmethode (Karagiannis u. Kühn [2002]) abgeleitet. Dabei besteht eine Modellierungsmethode aus Modellierungssprache, Vorgehen und Mechanismen. Für die Beschreibung der Modellierungssprache benötigt man Syntax, Notation und Semantik, welche durch Modelltypen, Modellierungselemente und Beziehungen definiert werden. Das Vorgehen wird durch unterschiedliche Phasen, welche wiederum in Schritte unterteilt werden, beschrieben. Mechanismen sollen das Vorgehen unterstützen.

Des Weiteren werden Ressourcen benötigt, um diverse Objekte besser beschreiben zu können. Diese Ressourcen werden durch Metadaten beschrieben, die zumindest aus Namen, Beschreibung, Erstellungsdatum, Sprache und Speicherort bestehen. Die Metadaten sind an Dublin Core Metadata Initiative [2010] angelehnt. Des Weiteren besteht eine Ressource aus Multipurpose Internet Mail Extensions-Typ (MIME-Typ) (Freed u. Borenstein [1996]), Daten, Hashcode und Größe. Um die Anforderungen laut Karagiannis u. a. [2010] erfüllen zu können, muss ein Modell durch Name, Typ, Bild und Originaldatei beschrieben sein. Die verwendete Modellierungsmethode wird ebenfalls beim Modell benötigt. Zur besseren Verwaltung von Bildern und Tools werden dafür ebenfalls Objekte benötigt, welche dieselben Eigenschaften wie Ressourcen haben.

Für die Verwendung innerhalb der Open Models Initiative werden zusätzliche Objekte für Projekte, User und Communities benötigt.

Die Objekte mit deren Attributen und Zusammenhänge werden im nachfolgendem Kapitel 4.4 genauer beschrieben und grafisch dargestellt.

#### Operationen

Zu der Verwaltung von Objekten zählen einfache Operationen, wie Einfügen, Archivieren und Aktualisieren. Dabei ist wichtig, dass es zu jedem Zeitpunkt möglich ist, zu wissen, welcher Benutzer, wann Objekte eingefügt, archiviert oder verändert hat. Dabei ist zu beachten, dass auf das MM-Repository mehrere Benutzer Zugriff haben. Daher muss sichergestellt werden,

dass unterschiedliche Benutzer nicht gleichzeitig an Objekten arbeiten bzw. Änderungen von Benutzern nicht verloren gehen.

Beim Anlegen von neuen Objekten soll das MM-Repository prüfen, ob dieses Objekt bereits vorhanden ist oder nicht. Wenn es bereits vorhanden ist, soll eine neue Version des Objekts abgelegt werden, wenn nicht, soll ein neues Objekt eingefügt werden. Die vorangegangene Version des Objekts muss daher als obsolet markiert werden. Je nachdem welches Objekt gespeichert werden soll, gibt es unterschiedliche Attribute, die vom Benutzer entweder manuell eingegeben werden oder vom MM-Repository vorgeschlagen werden.

Eine wichtige Voraussetzung für das Archivieren von Objekten ist, dass keine Objekte aus dem MM-Repository entfernt werden, sondern nur als obsolet markiert werden können. So ist es nicht möglich, dass Modelle im MM-Repository vorhanden sind, die mit keiner Modellierungsmethode kompatibel sind und dadurch nicht mehr verwendet werden können.

Des Weiteren ist es möglich, dass Objekte durch eine neuere Version ersetzt werden sollen. Hier ist es ebenfalls wichtig, dass die älteren Versionen der Objekte nicht aus dem MM-Repository entfernt werden, sondern noch immer auffindbar sind, da es möglich sein soll, dass Benutzer die älteren Versionen verwenden, wenn die Benutzer dies ausdrücklich wollen. Für das Aktualisieren von Objekten gibt es zwei unterschiedliche Möglichkeiten: der Benutzer kennt das Objekt, welches auf den neuesten Stand gebracht werden soll, oder er kennt das Objekt nicht und möchte eigentlich ein neues Objekt anlegen. Im ersten Fall wird eine neue Version des alten Objekts angelegt. Der zweite Fall wird durch das Anlegen eines neuen Objekts ausgelöst. Dabei weiß der Benutzer nicht, dass es bereits ein ähnliches Objekt gibt und es wird nach einer Rückmeldung zum Benutzer eine neue Version des vorhandenen Objekts angelegt, falls der Benutzer damit einverstanden ist. Anderenfalls wird ein neues Objekt angelegt. Beim Aktualisieren von Objekten ist wichtig zu beachten, dass in jedem Fall eine neue Version des Objekts angelegt wird und keine Objekte überschrieben werden.

### 3.2.2 Repräsentation von Objekten

Es soll möglich sein, die Attribute der Objekte des MM-Repositories dem Benutzer zur Verfügung zu stellen. Dafür ist eine grafische Aufbereitung der Daten notwendig. Da die Objekte im Originalformat gespeichert werden, wird es nicht immer möglich sein, dieses Format zu interpretieren, jedoch kann bei den unterschiedlichen Dokumenten das Format angegeben werden und das Interpretieren des Originalformats wird dadurch vereinfacht. Für die Definition des Formats von Dokumenten kann MIME-Typ verwendet werden, da dies ein gebräuchlicher Standard für unterschiedliche Formate ist. Die unterschiedlichen Typen werden in Freed u. Borenstein [1996]

beschrieben, untergliedern sich grob in Text, Bild, Audio, Video und Applikation sowie in die zusammengesetzten Typen.

Bei der Repräsentation sollen ebenfalls Zusammenhänge zwischen Modellen, Modellierungsmethoden und Tools dem Benutzer dargestellt werden. Dadurch wird der Benutzer bei der Verwendung der unterschiedlichen Objekte unterstützt. Es soll auch möglich sein zwischen den einzelnen Objekten zu navigieren. Zum Beispiel soll es möglich sein, wenn man eine Modellierungsmethode auswählt, dass alle Modelle dieser Methode angezeigt werden.

Die Unterstützung von unterschiedlichen Sichten auf die Objekte ist ebenfalls ein wichtiger Punkt in der Repräsentation von Objekten. So können in unterschiedlichen Anwendungsbereichen unterschiedliche Attribute von Objekten mehr oder weniger Bedeutung haben. Diese Sichten sollen konfigurierbar sein.

### 3.2.3 Suchfunktionalität

Eine wichtige funktionale Anforderung bei der Verwendung eines MM-Repositories ist die Suche. Dabei soll es unterschiedliche Arten von Suchen geben, da die Bedürfnisse der Benutzer unterschiedlich sind.

Die einfache Suche soll in allen Objekten nach den eingegebenen Wörtern suchen. Dabei ist eine Suche wünschenswert, die auch ähnliche Resultate liefert. Daher ist es im MM-Repository möglich, zu den einzelnen Objekten Begriffe (Topics) zu definieren, welche bei einer Suche berücksichtigt werden. Dem Benutzer wird das Suchergebnis nach den Objekten gegliedert dargestellt, damit das Ergebnis übersichtlich aufbereitet ist. Hier ist zu beachten, dass auch als obsolet markierte Modellierungsmethoden gefunden werden können. Für den Benutzer muss jedoch ersichtlich sein, dass das Objekt veraltet ist und eventuell eine neuere Version verfügbar ist. Modelle, die veraltet sind, sollen in dieser Suche für den Benutzer nicht auffindbar sein.

Bei der erweiterten Suche oder Expertensuche kann der Benutzer mehrere Einstellungen vornehmen. Er kann auswählen, ob er nur nach Modellen oder Modellierungsmethoden suchen will, oder eine Suche auf Attributebene durchführen will. Des Weiteren kann er auch entscheiden, ob veraltete Objekte im Suchergebnis vorhanden sein sollen oder nicht. Dadurch kann der Benutzer bei dieser Suche auch nach veralteten Modellen suchen.

Wichtig bei der Suche ist, dass der Benutzer immer bei der weiteren Verwendung der Objekte unterstützt wird. Dies bedeutet, dass er wenn er ein Modell als Suchergebnis bekommt, sofort auch die dazugehörige Modellierungsmethode zur Verfügung stellt wird. Bei Modellierungsmethoden sollen dem Benutzer mögliche Modelle dargestellt werden.

### 3.2.4 Versionierung von Objekte

Wie bereits bei der Verwaltung von Objekten erwähnt wurde, kann ein Objekt mehrere Versionen haben. Informationen, die pro Version gespeichert werden sollen sind: Erstellungsdatum, Ersteller, Hauptrelease-Nummer, Subrelease-Nummer und Kommentar. Des Weiteren soll es möglich sein, Zweige in der Entwicklung von Objekten einzufügen. Für das Löschen von Objekten ist es weiters notwendig, dass Objekte als obsolet markiert werden können.

Dem Benutzer soll ebenfalls die Möglichkeit der Darstellung eines Versionsbaums für die unterschiedlichen Objekte geboten werden. Dabei erhält der Benutzer eine grafische Repräsentation der Versionsentwicklung eines Objekts.

Eine zusätzliche Möglichkeit bei der Versionierung von Objekten ist die Darstellung der Unterschiede. Dabei werden dem Benutzer die Attribute von zwei Versionen eines Objekts aufgelistet, die sich unterscheiden und jene die unverändert geblieben sind.

### 3.2.5 Zusammenstellung von Modellierungsmethoden

Dem Benutzer kann die unterschiedlichen Komponenten einer Modellierungsmethode individuell zusammenstellen und in einem weiteren Schritt die neu gestellte Konfiguration verwenden. Ein wichtiger Teil der Konfiguration ist die Auswahl der Modellierungssprache. Dabei können sowohl Modellierungsklassen und Beziehungsklassen mit unterschiedlicher Syntax und Notation ausgewählt werden, welche zuvor im MM-Repository hinzugefügt wurden. Die Syntax wird vor allem durch die diversen Attribute definiert. Zur Modellierungssprache gehören ebenfalls Modelltypen, Klassen und Relationen, die ebenfalls ausgewählt werden können. Zu einer Modellierungsmethode gehört weiters das Vorgehen, das in Phasen und Schritte unterteilt werden und diese ebenfalls einzeln ausgewählt werden können. Zuletzt kann der Benutzer Mechanismen zur Modellierungsmethode hinzufügen, die innerhalb der Schritte verwendet werden können und von der Modellierungssprache verwendet werden.

Durch diese Zusammenstellung einer Modellierungsmethode sind alle Komponenten frei konfigurierbar und können individuell für unterschiedliche Anwendungsbereiche verwendet werden. Durch die unterschiedlichen Versionen, die bei den einzelnen Objekten angelegt werden können, können unterschiedliche Interpretationen der diversen Objekte verglichen und verwendet werden.

### 3.2.6 Exportieren von Modellierungssprachen

Nachdem der Benutzer alle notwendigen Objekte im MM-Repository angelegt hat, kann daraus eine Modellierungssprache erzeugt werden, die anschließend in ADOxx® importiert werden kann. Diese kann danach als Grundlage für die Implementierung der Modellierungsmethode verwendet werden. Dafür muss der Benutzer alle benötigten Modelltypen, Modellierungsklassen und Beziehungsklassen auswählen, die er verwenden möchte. Des Weiteren kann er zu den unterschiedlichen Objekten, die gewünschte Syntax und Notation wählen. Anschließend werden alle Modelltypen, Modellierungsklassen und Beziehungsklassen mit der dazugehörigen Syntax und Notation exportiert.

### 3.2.7 Erstellen von Dokumentationen

Für eine zuvor zusammengestellte Konfiguration einer Modellierungsmethode kann eine Dokumentation in unterschiedlichen Formaten erstellt werden. Dabei kann der Benutzer zwischen HyperText Markup Language (HTML), DOCument (DOC)[12] und Portable Document Format (PDF) Dokumentation der Modellierungsmethode wählen. Der Benutzer bekommt alle Komponenten und Objekte der Konfiguration aufgelistet, die für die Dokumentation herangezogen werden. Danach werden diese Komponenten und Objekte im gewünschten Format zusammengefasst und dem Benutzer zur Verfügung gestellt.

### 3.2.8 Benutzerrollen

Das MM-Repository muss mit unterschiedlichen Benutzern und Benutzerrechten verwendet werden können. Es soll eine Benutzergruppe geben, die Objekte uneingeschränkt verwalten kann und eine Gruppe, die nur eingeschränkten Zugriff auf die Objekte erhält. Die Unterschiede dieser Benutzergruppen werden nun kurz dargestellt.

Ein uneingeschränkten Zugriff auf die Objekte des MM-Repositories bedeutet, dass der Benutzer alle vorher definierten Möglichkeiten des MM-Repositories nutzen kann. Die Hauptaufgabe dieser Benutzergruppe ist die Verwaltung von Objekten. Jedoch soll es dieser Benutzergruppe auch möglich sein, Objekte zu suchen und zu exportieren sowie die Repräsentationskomponente zu benutzen.

Die Benutzergruppe mit einem eingeschränkten Zugriff hat ausschließlich lesenden Zugriff. Dies bedeutet, dass diese Benutzer die Objekte suchen und exportieren können. Nach der Suche

[12] Format für Mircosoft® Office Dokumente

werden die Objekte in der bereits definierten Form repräsentiert, danach können sie die Objekte verwenden, jedoch nicht ändern oder löschen. Eine weitere notwendige Aktivität für Benutzer mit eingeschränktem Zugriff ist das Exportieren der Modellierungssprache.

# 4 Modellierungsmethoden-Repository: Konzeption

Dieses MM-Repositorykonzept basiert auf der Schlussfolgerung der Analyse von existierenden Repositories (siehe Kapitel 2.3 und 2.4) und beantwortet die offenen Fragen für die Entwicklung eines Repositorykonzepts. Daher werden in diesem Kapitel die Eigenschaften des MM-Repositories definiert, die Funktionalitäten des MM-Repositories und externen Schnittstellen beschrieben sowie die Architektur und Userinteraktion festgehalten. Abschluss dieses Kapitels ist ein Vergleich des beschriebenen MM-Repositorykonzepts mit datenbank-basierten Lösungen. Der wesentliche Vorteil dieses MM-Repositorykonzepts ist, dass eine Metamodellierungsplattform für Modellierungsmethoden verwendet wird. Daher wird das Repositorykonzept durch Modellierungsmethoden umgesetzt, wodurch der Metamodellierungsansatz angewendet wird.

## 4.1 Eigenschaften des MM-Repositories

Durch die Analyse von bestehenden Repositories (siehe Kapitel 2.3.3) wurden einige Eigenschaften identifiziert, welche für dieses MM-Repositorykonzept ebenfalls von Bedeutung sind. So soll das MM-Repository einen zentralen Zugriff auf alle Objekte und Inhalte des MM-Repositories liefern, der physische Speicherort ist jedoch nicht relevant, da er im MM-Repository hinterlegt wird. Daher können die Datenquellen auf unterschiedliche Orte verteilt werden. Der Zugriff aus das MM-Repository soll jederzeit über vordefinierte Schnittstellen möglich sein, wodurch unterschiedliche Services für die Benutzer zur Verfügung gestellt werden. Das MM-Repository soll sowohl Metadaten als auch Objekte verwalten können. Diese Metadaten und Objekte sind jedoch unterschiedlich aufgebaut, wodurch das MM-Repository einen heterogenen Charakter aufweist. Weitere relevante Eigenschaften sind Erweiterbarkeit des Datenmodells und Vertrauenswürdigkeit der Inhalte, wodurch ein Authentifizieren der Benutzer notwendig ist.

## 4.2 Funktionalitäten des MM-Repositories

Um die funktionalen Anforderungen der Benutzer aus Kapitel 3 sowie Funktionen existierender Repositories (siehe Kapitel 2.3.2) erfüllen zu können, muss ein MM-Repository für die Verwaltung

von Objekten, in diesem speziellen Fall von Modellen, Modellierungsmethoden und Projekte, diverse Funktionalitäten zur Verfügung stellen. In diesem Kapitel werden die Funktionen aus einer technischeren Sicht, jedoch noch plattform- und implementierungsunabhängig, beschrieben. Diese Funktionalitäten beschäftigen sich mit Content-Management, Zugriffsmanagement, Transaktionsmanagement, Locking-Management, Lebenszyklus-Management, Abfragenmanagement, Versionsmanagement und Konfigurationsmanagement. Die angeführten Funktionalitäten können jedoch nicht isoliert voneinander betrachtet werden, sondern stehen meist in engem Zusammenhang und Abhängigkeit mit anderen Funktionalitäten des MM-Repositories. In den nachfolgenden Kapiteln werden die unterschiedlichen Funktionalitäten, die für die Erfüllung der Anforderungen notwendig sind, genauer beschrieben. Für eine übersichtliche Darstellung der Funktionen wird eine Schablone (Tabelle 4.1) verwendet, welche aus folgenden Feldern besteht: Name, Beschreibung, Input, Output, Vorbedingung, Nachbedingung und Aufruf.

| **<Name der Funktion>** | |
|---|---|
| Beschreibung | <Beschreibung der Funktion> |
| Input | <Inputspezifikation der Funktion> |
| Output | <Outputspezifikation der Funktion> |
| Vorbedingung | <Vorbedingung der Funktion> |
| Nachbedingung | <Nachbedingung der Funktion> |
| Aufruf | <Outputtype> <Funktionsname> (<Inputtype> <Inputname>, ...) |

Tabelle 4.1: Schablone für die Spezifikation von Funktionen

Um ein möglichst einheitliches Verhalten der Funktionen zu gewährleisten, gibt es einige allgemein gültige Regeln, die hier definiert werden:

- Eine Funktion kann einen Output-Parameter und mehrere Input-Parameter haben. Der Output-Parameter ist durch den Typ definiert. Input-Parameter werden durch Typ und Name definiert.
- Einige Funktionen liefern den Status zurück, ob eine Funktion erfolgreich war oder nicht. Wenn der Rückgabewert kleiner oder gleich -1 ist, wurde die Funktion nicht erfolgreich ausgeführt. Wenn der Rückgabewert größer als -1 ist, wurde die Funktion erfolgreich ausgeführt.
- Objekttypen, die im MM-Repository gespeichert werden können, sind vom Typ T.
- Mögliche Input- und Outputtypen sind jene, die auch in Java (Version 1.6) vorhanden sind, sowie selbstdefinierte Typen, wie Attribute, Configuration, usw.

### 4.2.1 Content-Management

Zum Content-Management zählen die Funktionen Erzeugen und Aktualisieren von Objekten, welche im Metamodell in Kapitel 4.4 genauer beschrieben werden. Objekte können entweder nur aus Metadaten bestehen oder auch die Originaldaten des Objekts beinhalten. Aus diesem Grund gibt es die Funktion für das Hochladen von Dateien.

Da Objekte aus dem MM-Repository nicht gelöscht werden können, gibt es keine Lösch-Funktionalität. Jedoch gibt es die Möglichkeit Objekte zu archivieren (siehe Kapitel 4.2.4).

| **Objekt erzeugen** | |
|---|---|
| Beschreibung | Mithilfe dieser Funktion wird ein Objekt in das MM-Repository gespeichert. Es werden zu einem Objekt die beschreibenden Daten im MM-Repository abgelegt. |
| Input | Objekt mit Objektattributen und gewünschten Werten |
| Output | Status, ob das Erzeugen des Objekts erfolgreich war oder nicht |
| Vorbedingung | Objekt muss bekannt sein. |
| Nachbedingung | Wenn das Erzeugen erfolgreich war, ist das neue Objekt im MM-Repository vorhanden. |
| Aufruf | Integer createObject(T object) |

Tabelle 4.2: Funktionsspezifikation: Objekt erzeugen

| **Datei hochladen** | |
|---|---|
| Beschreibung | Mithilfe dieser Funktion wird ein Objekt mit Dateianhang in das MM-Repository gespeichert. Dabei wird die Datei und alle beschreibenden Daten im MM-Repository abgelegt und müssen miteinander verbunden werden. |
| Input | Objekt mit Objektattributen und gewünschten Werten, Datei |
| Output | Status, ob das Erzeugen des Objekts erfolgreich war oder nicht |
| Vorbedingung | Objekttyp und Objektattribute müssen bekannt sein. Der Objekttyp muss für die Speicherung von Dateien ausgelegt sein. |
| Nachbedingung | Wenn das Erzeugen erfolgreich war, ist das neue Objekt und die Datei im MM-Repository vorhanden und miteinander verbunden. |
| Aufruf | createObject(T object, File originalfile) |

Tabelle 4.3: Funktionsspezifikation: Datei hochladen

| **Objekt aktualisieren** | |
|---|---|
| Beschreibung | Mit dieser Funktion wird ein Objekt aktualisiert bzw. mit neuen Attributen befüllt. Dabei wird jedoch nicht das Objekt verändert, sondern eine neue Version des bereits vorhandenen Objekts mit den neuen Attributen erzeugt. Dies ist notwendig, um alle Änderungen nachvollziehen zu können. |
| Input | Objekt mit Objektattributen und gewünschten Werten |
| Output | Status, ob das Aktualisieren des Objekts erfolgreich war oder nicht |
| Vorbedingung | Objekt mit der eindeutigen Identifikation muss im MM-Repository vorhanden und die Attribute müssen bekannt sein. |
| Nachbedingung | Wenn das Aktualisieren erfolgreich war, ist eine neue Version des Objekts im MM-Repository verfügbar. |
| Aufruf | updateObject(T object) |

Tabelle 4.4: Funktionsspezifikation: Objekt aktualisieren

### 4.2.2 Zugriffsmanagement

Das Zugriffsmanagement regelt, welche Benutzer mit welchen Rechten es im MM-Repository gibt. Dabei wird ebenfalls im Zugriffsmanagement festgelegt, welche Funktionen von welchem Benutzer ausgeführt werden dürfen. So hat ein eingeschränkter Benutzer nur lesenden Zugriff auf das MM-Repository, ein Administrator jedoch hat lesenden und schreibenden Zugriff auf das MM-Repository. Für das Zugriffsmanagement muss es möglich sein, Benutzer und Rollen anlegen zu können und die entsprechenden Rechte zuteilen zu können. Bevor ein Benutzer Informationen aus dem MM-Repository erhält muss er sich authentifizieren.

| **Benutzer anlegen** | |
|---|---|
| Beschreibung | Ein Benutzer wird zum MM-Repository hinzugefügt, um anschließend Zugang zum MM-Repository zu bekommen. |
| Input | Benutzername, Passwort, optional Rechte, optional Benutzergruppe |
| Output | Status, ob das Erzeugen des Objekts erfolgreich war oder nicht |
| Vorbedingung | Der eindeutige Benutzername darf noch nicht im MM-Repository vorhanden sein. |
| Nachbedingung | Wenn das Anlegen erfolgreich war, ist der Benutzer im MM-Repository vorhanden und kann sich in weiterer Folge anmelden. |
| Aufruf | createUser(String name, String password)<br>createUser(String name, String password, Right[] rights)<br>createUser(String name, String password, Right[] rights, String usergroup)<br>createUser(String name, String password, String usergroup) |

Tabelle 4.5: Funktionsspezifikation: Benutzer anlegen

| **Benutzergruppe anlegen** | |
|---|---|
| Beschreibung | Eine Benutzergruppe wird zum MM-Repository hinzugefügt, um anschließend Benutzer dieser Gruppe zuordnen zu können. |
| Input | Eindeutiger Benutzergruppenname, optional Liste mit Benutzern, optional Rechte |
| Output | Status, ob das Anlegen der Benutzergruppe erfolgreich war oder nicht |
| Vorbedingung | Der eindeutige Benutzergruppenname darf noch nicht im MM-Repository vorhanden sein. |
| Nachbedingung | Wenn das Anlegen erfolgreich war, ist die Benutzergruppe im MM-Repository vorhanden. Dadurch können Benutzer und Rechte zugeordnet werden. |
| Aufruf | createUsergroup(String usergroup)<br>createUsergroup(String usergroup, User[] users, Right[] rights) |

Tabelle 4.6: Funktionsspezifikation: Benutzergruppe anlegen

| **Benutzer löschen** | |
|---|---|
| Beschreibung | Es kann ein Benutzer aus dem MM-Repository wieder entfernt werden. Dadurch erlischt sein Zugriff auf das MM-Repository. |
| Input | Eindeutiger Benutzername |
| Output | Status, ob das Löschen des Benutzers erfolgreich war oder nicht |
| Vorbedingung | Der Benutzer mit der eindeutigen Identifikation muss im MM-Repository vorhanden sein. |
| Nachbedingung | Wenn das Löschen erfolgreich war, ist der Benutzer aus dem MM-Repository entfernt und hat keinen Zugriff auf das MM-Repository. |
| Aufruf | deleteUser(String username) |

Tabelle 4.7: Funktionsspezifikation: Benutzer löschen

| **Benutzergruppe löschen** | |
|---|---|
| Beschreibung | Es kann eine Benutzergruppe aus dem MM-Repository wieder entfernt werden. Dadurch verlieren die Benutzer dieser Gruppe die Gruppenrechte und müssen einer anderen Gruppe zugeordnet werden. |
| Input | Eindeutiger Benutzergruppenname |
| Output | Status, ob das Löschen der Benutzergruppe erfolgreich war oder nicht |
| Vorbedingung | Die Benutzergruppe mit der eindeutigen Identifikation muss im MM-Repository vorhanden sein. |
| Nachbedingung | Wenn das Löschen erfolgreich war, ist die Benutzergruppe aus dem MM-Repository entfernt. Die Benutzer dieser Gruppe müssen einer anderen Gruppe zugeordnet werden. |
| Aufruf | deleteUsergroup(String usergroup) |

Tabelle 4.8: Funktionsspezifikation: Benutzergruppe löschen

| **Rechte zuordnen** | |
|---|---|
| Beschreibung | Einer Benutzergruppe oder einem Benutzer können unterschiedliche Rechte zugeordnet werden. Dabei können Rechte auf das gesamte MM-Repository oder auf einzelne Objekte festgelegt werden. Die unterschiedlichen Zugriffsrechte sind Lesen und Schreiben. |
| Input | Benutzer oder Benutzergruppe, Objekt oder MM-Repository, Lesender oder schreibender Zugriff |
| Output | Status, ob das Zuordnen erfolgreich war oder nicht |
| Vorbedingung | Der Benutzer oder die Benutzergruppe muss im MM-Repository vorhanden sein. Falls die Rechte auf Objektebene zugeordnet werden soll, muss das Objekt vorhanden sein. |
| Nachbedingung | Wenn das Zuordnen erfolgreich war, haben der Benutzer oder die Benutzer der Benutzergruppe die entsprechenden Rechte und können damit auf das MM-Repository zugreifen. |
| Aufruf | assignRights(User object, T object, Right rights)<br>assignRights(String usergroup, T object, Right rights)<br>assignRights(User object, Right rights)<br>assignRights(String usergroup, Right rights) |

Tabelle 4.9: Funktionsspezifikation: Rechte zuordnen

| **Benutzer anmelden** | |
|---|---|
| Beschreibung | Ein Benutzer kann sich im MM-Repository anmelden und mit den zugeordneten Rechten auf das MM-Repository zugreifen. Diese Rechte können Benutzerrechte oder Gruppenrechte sein. |
| Input | Benutzername und Passwort |
| Output | Status, ob das Anmelden erfolgreich war oder nicht |
| Vorbedingung | Der Benutzer muss im MM-Repository vorhanden sein und das Passwort muss gültig sein. |
| Nachbedingung | Wenn das Anmelden erfolgreich war, hat der Benutzer Zugriff auf die Objekte. |
| Aufruf | login(String username, String password) |

Tabelle 4.10: Funktionsspezifikation: Benutzer anmelden

| **Benutzer abmelden** | |
|---|---|
| Beschreibung | Ein Benutzer kann sich vom MM-Repository wieder abmelden. |
| Input | Benutzername |
| Output | Status, ob das Abmelden erfolgreich war oder nicht |
| Vorbedingung | Der Benutzer muss im MM-Repository angemeldet sein. |
| Nachbedingung | Wenn das Abmelden erfolgreich war, hat der Benutzer keinen Zugriff auf die Objekte. |
| Aufruf | logout(String username) |

Tabelle 4.11: Funktionsspezifikation: Benutzer abmelden

### 4.2.3 Transaktions- und Locking-Management

Im Zusammenhang mit diesem MM-Repositorykonzept ist es im Transaktionsmanagement besonders wichtig, dass bei der Speicherung von Objekten immer die vollständigen Informationen abgelegt werden oder der gesamte Vorgang rückgängig gemacht werden muss. In Datenbanksystemen haben Transaktionen laut Haerder u. Reuter [1983] folgende ACID-Eigenschaften: (A) atomar, (C) konsistent, (I) isoliert und (D) dauerhaft. Für dieses MM-Repositorykonzept sind diese Eigenschaften ebenfalls notwendig. Atomar bedeutet, dass entweder alles ausgeführt wird oder nichts. Dies bedeutet, dass etwa bei der zusätzlichen Speicherung der Dateien sowohl die beschreibenden Informationen als auch die Dateien abgelegt werden müssen. Ebenfalls muss bei den zusätzlichen Informationen gewährleistet werden, dass wenn das Objekt im MM-Repository angelegt wurde alle zugehörigen Informationen gespeichert werden. Konsistent bedeutet, dass die Inhalte des MM-Repositories keine Widersprüche beinhalten dürfen. Isoliert im Zusammenhang mit Transaktionen heißt, dass die Durchführung der Funktionen sich nicht gegenseitig behindern dürfen. Dafür wird vor allem das Locking-Management benötigt. Nach der erfolgreichen Durchführung einer Transaktion sind die Informationen dauerhaft im MM-Repository vorhanden. Weiters muss es möglich sein beliebige Funktionsaufrufe zu einer Transaktion zusammenzufassen und auszuführen.

Mit Hilfe des Locking-Managements wird gewährleistet, dass nicht zwei oder mehr Benutzer gleichzeitig auf ein gewünschtes Objekt zugreifen. Daher ist Locking nur in Mehrbenutzer-Systemen notwendig. Die Aufgaben dabei sind das Setzen und Freigeben von Sperren, Verhinderung von dauerndem Blockieren und Verhinderung von Deadlocks (Schlageter u. Stucky [1983]). Beim Sperren muss zwischen lesendem und schreibendem Zugriff unterschieden werden, da es möglich sein soll, dass mehrere Benutzer dasselbe Objekt lesen, jedoch nicht schreibend darauf zugreifen können. Daher darf kein anderer Benutzer darauf zugreifen, wenn ein schreibender Zugriff erfolgt und es ergeben sich die Funktionen Sperren und Freigeben eines Objekts in den

| | | | | |
|---|---|---|---|---|
| T1 | Sperren $O_1$ | | Sperren $O_2$ (warten) | |
| T2 | | Sperren $O_2$ | | Sperren $O_1$ (warten) |

Abbildung 4.1: Beispiel eines Deadlock

unterschiedlichen Sperrmodi.

Deadlocks entstehen, wenn ein Benutzer auf die Freigabe eines Benutzers wartet, welcher wiederum auf die Freigabe des anderen Benutzers wartet (vgl. Abbildung 4.1). Dies kann man lösen, indem man entweder alle benötigten Objekte auf einmal sperrt oder in derselben Reihenfolge sperrt oder regelmäßig überprüft, ob ein Deadlock entstanden ist und gegebenenfalls die Transaktionen rückgängig macht und danach neu startet. Um Deadlocks hier zu vermeiden, werden alle Objekte, die innerhalb einer Transaktion benötigt werden, auf einmal gesperrt.

| **Transaktion anlegen** | |
|---|---|
| Beschreibung | Es wird eine Sammlung von Funktionen zu einer Transaktion zusammengefasst. Die darin befindlichen Funktionen müssen erfolgreich ausgeführt werden. |
| Input | Eindeutige Identifikation, Liste von Funktionen |
| Output | Status, ob das Anlegen erfolgreich war oder nicht |
| Vorbedingung | Die Identifikation darf nicht vorhanden sein. |
| Nachbedingung | Wenn das Anlegen erfolgreich war, kann die Transaktion gestartet werden. |
| Aufruf | createTransaction(String id, Map<String, Function> functionmap) |

Tabelle 4.12: Funktionsspezifikation: Transaktion anlegen

| **Transaktion starten** | |
|---|---|
| Beschreibung | Die Funktion startet eine angelegte Transaktion. Die beinhalteten Funktionen werden aufgerufen. |
| Input | Eindeutige Identifikation der Transaktion |
| Output | Status, ob das Ausführen der Transaktion erfolgreich war oder nicht |
| Vorbedingung | Die Transaktion muss vorhanden sein. |
| Nachbedingung | Wenn nur eine Funktion der Transaktion nicht erfolgreich durchgeführt werden konnte, muss die gesamte Transaktion rückgängig gemacht werden. Wenn alle Funktionen erfolgreich durchgeführt wurden, war die Transaktion erfolgreich. |
| Aufruf | startTransaction(String id) |

Tabelle 4.13: Funktionsspezifikation: Transaktion starten

| **Transaktion rückgängig machen** | |
|---|---|
| Beschreibung | Alle durchgeführten Funktionen einer Transaktion müssen rückgängig gemacht werden. Nicht durchgeführte Funktionen dürfen nicht rückgängig gemacht werden. |
| Input | Eindeutige Identifikation der Transaktion, Liste der Funktionen mit Status |
| Output | Status, ob das Sperren alle Objekte erfolgreich war oder nicht |
| Vorbedingung | Innerhalb einer Transaktion wurde zumindest eine Funktion nicht erfolgreich ausgeführt. |
| Nachbedingung | Andere Transaktionen und Funktionen können wieder ausgeführt werden. Das MM-Repository befindet sich in einem konsistenten Zustand. |
| Aufruf | rollbackTransaction(String id, Map<String, State> functionstatemap) |

Tabelle 4.14: Funktionsspezifikation: Transaktion rückgängig machen

| **Objekte einer Transaktion sperren** | |
|---|---|
| Beschreibung | Alle Objekte, die innerhalb einer Transaktion verwendet werden, sollen zu Beginn der Transaktion gesperrt werden. Dabei wird je nachdem welcher Sperrmodus angeben wird entweder *readlockObject(String id, T object)* oder *writelockObject(String id, T object)* aufgerufen. |
| Input | Eindeutige Identifikation der Transaktion, Liste der Objekte mit Sperrmodus |
| Output | Status, ob das Sperren alle Objekte erfolgreich war oder nicht |
| Vorbedingung | Die einzelnen Objekte dürfen nicht mit schreibendem Zugriff gesperrt sein. |
| Nachbedingung | Falls das Sperren erfolgreich war, müssen die Objekte mit dem Sperrstatus solange gespeichert bleiben bis die Objekte wieder freigegeben werden. Falls das Sperren nicht erfolgreich war, wird eine Fehlermeldung zurückgeliefert und die Transaktion muss zu einem späteren Zeitpunkt wiederholt werden. |
| Aufruf | lockObjects(String id, Map<String, Modi> objectmodimap) |

Tabelle 4.15: Funktionsspezifikation: Objekte einer Transaktion sperren

| **Objekte einer Transaktion freigeben** | |
|---|---|
| Beschreibung | Alle Objekte einer Transaktion werden am Schluss der Transaktion wieder freigegeben und können anschließend von anderen verwendet werden. Dabei wird für jedes Objekt die Funktion *unlockObject(String id, T object)* aufgerufen. |
| Input | Eindeutige Identifikation der Transaktion |
| Output | Status, ob das Freigeben alle Objekte erfolgreich war oder nicht |
| Vorbedingung | Die Objekte waren mit lesendem oder schreibendem Zugriff gesperrt. |
| Nachbedingung | Falls das Freigeben erfolgreich war, werden die Zustände der Objekte wieder geändert und andere dürfen das Objekt verwenden. |
| Aufruf | unlockObjects(String id) |

Tabelle 4.16: Funktionsspezifikation: Objekte einer Transaktion sperren

| **Lesender Zugriff auf ein Objekt** | |
|---|---|
| Beschreibung | Ein Objekt wird mit lesendem Zugriff gesperrt. Dies bedeutet, dass andere lesend auf das Objekt zugreifen können, jedoch nicht schreibend. Falls ein Objekt bereits schreibenden Zugriff auf das Objekt hat, ist der lesende Zugriff nicht möglich. Wenn das Objekt jedoch mit lesendem Zugriff gesperrt wurde, ist ein weiterer lesender Zugriff möglich. |
| Input | Eindeutige Identifikation der Transaktion, Objekt |
| Output | Status, ob das Sperren mit lesendem Zugriff erfolgreich war oder nicht |
| Vorbedingung | Das Objekt darf nicht mit schreibendem Zugriff gesperrt sein. |
| Nachbedingung | Falls das Sperren erfolgreich war, muss der Zustand solange gespeichert bleiben bis das Objekt wieder freigegeben wird. Falls das Sperren nicht erfolgreich war, wird eine Fehlermeldung zurückgeliefert und das Sperren muss zu einem späteren Zeitpunkt wiederholt werden. |
| Aufruf | readlockObject(String id, T object) |

Tabelle 4.17: Funktionsspezifikation: Lesender Zugriff auf ein Objekt

| **Schreibender Zugriff auf ein Objekt** | |
|---|---|
| Beschreibung | Ein Objekt wird mit schreibendem Zugriff gesperrt. Dies bedeutet, dass andere weder lesend noch schreibend auf das Objekt zugreifen können. Falls ein Objekt bereits schreibenden oder lesenden Zugriff auf das Objekt hat, ist der schreibende Zugriff nicht möglich. |
| Input | Eindeutige Identifikation der Transaktion, Objekt |
| Output | Status, ob das Sperren mit schreibendem Zugriff erfolgreich war oder nicht |
| Vorbedingung | Das Objekt darf nicht mit lesendem oder schreibendem Zugriff gesperrt sein. |
| Nachbedingung | Falls das Sperren erfolgreich war, muss der Zustand solange gespeichert werden, bis das Objekt wieder freigegeben wird. Falls das Sperren nicht erfolgreich war, wird eine Fehlermeldung zurückgeliefert und das Sperren muss zu einem späteren Zeitpunkt wiederholt werden. |
| Aufruf | writelockObject(String id, T object) |

Tabelle 4.18: Funktionsspezifikation: Schreibender Zugriff auf ein Objekt

| **Objekt freigeben** | |
|---|---|
| Beschreibung | Ein gesperrtes Objekt wird wieder freigegeben. Dies bedeutet, dass danach wieder jeglicher Zugriff auf das Objekt möglich ist. |
| Input | Eindeutige Identifikation der Transaktion, Objekt |
| Output | Status, ob das Freigeben erfolgreich war oder nicht |
| Vorbedingung | Das Objekt war mit lesendem oder schreibendem Zugriff gesperrt. |
| Nachbedingung | Falls das Freigeben erfolgreich war, wird der Zustand des Objekts wieder geändert und andere können das Objekt wieder uneingeschränkt verwenden. |
| Aufruf | unlockObject(String id, T object) |

Tabelle 4.19: Funktionsspezifikation: Objekt freigeben

### 4.2.4 Lebenszyklus-Management

Ein Objekt innerhalb des MM-Repositories kann unterschiedliche Zustände durchlaufen. Diese werden in diesem MM-Repository im ersten Schritt sehr gering gehalten, da Objekte momentan ins MM-Repository gespeichert und anschließend archiviert werden können. Daher kann ein Objekt lediglich zwei Zustände einnehmen: aktuell und archiviert. Archivierte Objekte sind im MM-Repository noch vorhanden und können gefunden werden, jedoch sollten diese Objekte nicht mehr verwendet werden.

| **Objekt archivieren** | |
|---|---|
| Beschreibung | Ein Objekt wird archiviert, wenn eine aktuellere Version vorhanden ist. |
| Input | Objekt |
| Output | Status, ob das Archivieren erfolgreich war oder nicht |
| Vorbedingung | Das Objekt muss im MM-Repository vorhanden sein. |
| Nachbedingung | Nach dem erfolgreichen Archivieren ist das Objekt als obsolet markiert und ist daher gekennzeichnet, dass es nicht mehr verwendet werden soll. |
| Aufruf | archiveObject(T Object) |

Tabelle 4.20: Funktionsspezifikation: Objekt archivieren

| **Aktuelle Version des Objekt festlegen** | |
|---|---|
| Beschreibung | Eine bereits archivierte Version eines Objekt wird wieder als aktuelle Version des Objekt markiert. Die aktuelle Version des Objekts muss wiederum archiviert werden. |
| Input | Objekt |
| Output | Status, ob das Ändern des Zustands erfolgreich war oder nicht. |
| Vorbedingung | Das Objekt muss im MM-Repository vorhanden sein und bereits als archiviert markiert sein. |
| Nachbedingung | Nach dem erfolgreichen Ändern des Zustands ist diese Version des Objekt wieder das aktuelle Objekt. |
| Aufruf | currentVersion(T Object) |

Tabelle 4.21: Funktionsspezifikation: Aktuelle Version des Objekt festlegen

### 4.2.5 Abfragenmanagement

Die verwalteten Objekte des MM-Repositories können mit Hilfe des Abfragenmanagements abgefragt werden. Daher sind unterschiedliche Funktionen für die Abfrage der Objekte notwendig. Die Daten können entweder durch vorgefertigte Abfragen abgerufen werden oder es können auch eigene Abfragen definiert werden.

| **Abfrage erstellen** | |
|---|---|
| Beschreibung | Es können eigene Abfragen definiert werden, um die gewünschten Objekte geliefert zu bekommen. |
| Input | Objekttyp, optional Liste mit gewünschten Attributen, optional Attributtyp mit Attributwert und Vergleichstyp (gleich, verschieden, beinhaltet, beinhaltet nicht, größer, größer oder gleich, kleiner, kleiner oder gleich) |
| Output | Abfragenidentifikation |
| Vorbedingung | Falls Attribut angegeben wird, muss das Attribut im Objekt vorhanden sein. |
| Nachbedingung | Nach dem erfolgreichen Erstellen der Abfrage kann diese zu einem späteren Zeitpunkt verwendet werden. |
| Aufruf | createQuery(String objecttype)<br>createQuery(String objecttype, List<Attribute> attributelist)<br>createQuery(String objecttype, Map<Attribute, String> comparemap)<br>createQuery(String objecttype, List<Attribute> attributelist, Map<String, String> comparemap) |

Tabelle 4.22: Funktionsspezifikation: Abfrage erstellen

| **Abfrage aufrufen** | |
|---|---|
| Beschreibung | Vordefinierte oder eigene Abfragen werden mit dieser Funktion aufgerufen und liefern die gewünschten Objekte. |
| Input | Abfragenidentifikation |
| Output | Liste mit abgefragten Objekten |
| Vorbedingung | Die Abfrage mit der eindeutigen Identifikation muss im MM-Repository vorhanden sein. |
| Nachbedingung | Nach dem erfolgreichen Ausführen der Abfrage werden die Objekte zurückgeliefert, welche anschließend für andere Funktionen verwendet werden können. |
| Aufruf | getQuery(String id) |

Tabelle 4.23: Funktionsspezifikation: Abfrage aufrufen

| **Abfrage löschen** | |
|---|---|
| Beschreibung | Eigene Abfragen werden mit dieser Funktion gelöscht und können anschließend nicht mehr aufgerufen werden. |
| Input | Abfragenidentifikation |
| Output | Status, ob das Löschen der Abfrage erfolgreich war oder nicht |
| Vorbedingung | Die Abfrage mit der eindeutigen Identifikation muss im MM-Repository vorhanden sein. |
| Nachbedingung | Nach dem erfolgreichen Löschen der Abfrage ist sie aus dem MM-Repository entfernt und kann nicht mehr verwendet werden. |
| Aufruf | deleteQuery(String id) |

Tabelle 4.24: Funktionsspezifikation: Abfrage löschen

### 4.2.6 Versionsmanagement

Aufgaben des Versionsmanagements sind vor allem die Unterstützung des Benutzers bei der Vergabe von Versionsnummern, die Repräsentation der unterschiedlichen Versionen eines Objekts, Zusammenhänge von unterschiedlichen Versionen eines Objekts zu anderen Objekten mit einer bestimmten Version, Erzeugen von neuen Versionen anhand einer bereits vorhanden Version sowie Erzeugen von neuen Entwicklungszweigen (Branches).

| **Versionsnummer automatisch vergeben** | |
|---|---|
| Beschreibung | Mit Hilfe dieser Funktion wird die Versionsnummer eines Objekts automatisch vergeben. |
| Input | Objekt, Release-Ebene, die erhöht werden soll (Main-Release und/oder Subrelease) |
| Output | Automatisch erzeugte Versionsnummer |
| Vorbedingung | Das Objekt muss im Repository vorhanden sein. |
| Nachbedingung | Die neue Version des Objekts kann mit der automatisch erzeugten Versionsnummer erstellt werden. |
| Aufruf | getVersionnumber(T object, Boolean mainrelease, Boolean subrelease) |

Tabelle 4.25: Funktionsspezifikation: Versionsnummer automatisch vergeben

| **Versionen eines Objekts zurückgeben** | |
|---|---|
| Beschreibung | Alle Versionen eines Objekts werden mit Hilfe dieser Funktion zurückgeliefert. |
| Input | Objektname |
| Output | Alle Versionen mit übergebenen Objektnamen |
| Vorbedingung | Das Objekt muss im MM-Repository vorhanden sein. |
| Nachbedingung | Die gelieferten Versionen können zu einem Versionsbaum weiterverarbeitet werden. |
| Aufruf | getVersions(String name) |

Tabelle 4.26: Funktionsspezifikation: Versionen eines Objekts zurückgeben

| **Abhängigkeiten eines Objekts zurückgeben** | |
|---|---|
| Beschreibung | Falls das Objekt eine Abhängigkeit zu einem anderen Objekt einer bestimmten Version aufweist, werden diese Abhängigkeiten mit Hilfe dieser Version zurückgeliefert. |
| Input | Objekt |
| Output | Alle Objekte, die in Beziehung mit dem übergebenen Objekt stehen |
| Vorbedingung | Das Objekt muss im MM-Repository vorhanden sein. |
| Nachbedingung | Die gelieferten Versionen können für die grafische Darstellung der Abhängigkeiten weiterverwendet werden. |
| Aufruf | getDependencies(T object) |

Tabelle 4.27: Funktionsspezifikation: Abhängigkeiten eines Objekts zurückgeben

| **Branch erzeugen** | |
|---|---|
| Beschreibung | Mit Hilfe dieser Funktion wird ein neuer Entwicklungszweig (Branch) erstellt. Dabei wird der die gewünschte Version eines Objekts kopiert und mit einem neuen Branch versehen. |
| Input | Objekt, Branchbezeichnung |
| Output | Status, ob das Erzeugen des Branches erfolgreich war oder nicht |
| Vorbedingung | Das Objekt muss im MM-Repository vorhanden sein und der Branch darf noch nicht vorhanden sein. |
| Nachbedingung | Es können neue Versionen im erzeugten Branch erstellt werden. |
| Aufruf | createBranch(T object, String branchname) |

Tabelle 4.28: Funktionsspezifikation: Branch erzeugen

### 4.2.7 Konfigurationsmanagement

Es soll die Möglichkeit geben, Modellierungsmethoden und deren Komponenten individuell zusammenzustellen. Daher wird ein Konfigurationsmanagement benötigt, welches sich mit der Verwaltung dieser Konfigurationen beschäftigt. Funktionen des Konfigurationsmanagements sind Erstellen, Aktualisieren und Löschen von Konfigurationen.

| **Konfiguration erstellen** | |
|---|---|
| Beschreibung | Mit Hilfe dieser Funktion kann eine Konfiguration mit unterschiedlichen Objekten erstellt werden. |
| Input | Eindeutiger Konfigurationsname sowie alle Objekte, die in der Konfiguration enthalten sein sollen |
| Output | Status, ob das Erzeugen der Konfiguration erfolgreich war oder nicht |
| Vorbedingung | Objekte müssen im MM-Repository vorhanden sein. |
| Nachbedingung | Wenn das Erzeugen erfolgreich war, ist die neue Konfiguration im MM-Repository vorhanden und kann weiterverwendet werden. |
| Aufruf | createConfiguration(String id, List<T> objects) |

Tabelle 4.29: Funktionsspezifikation: Konfiguration erstellen

| **Objekte von einer Konfiguration entfernen** | |
|---|---|
| Beschreibung | Mit Hilfe dieser Funktion kann eine vorhandene Konfiguration verändert werden. Es können vorhandene Objekte von einer Konfiguration entfernt werden. |
| Input | Konfiguration sowie alle Objekte, die von der Konfiguration entfernt werden sollen |
| Output | Status, ob das Aktualisieren der Konfiguration erfolgreich war oder nicht |
| Vorbedingung | Objekte müssen bei der Konfiguration vorhanden sein. |
| Nachbedingung | Wenn das Aktualisieren erfolgreich war, sind die Objekte bei der Konfiguration nicht mehr vorhanden und können nicht mehr verwendet werden. |
| Aufruf | removeObjectsFromConfiguration(String id, List<T> removeobjects) |

Tabelle 4.30: Funktionsspezifikation: Objekte von einer Konfiguration entfernen

| **Objekte zu einer Konfiguration hinzufügen** | |
|---|---|
| Beschreibung | Mit Hilfe dieser Funktion kann eine vorhandene Konfiguration verändert werden. Es können zusätzliche Objekte zu einer Konfiguration hinzugefügt werden. |
| Input | Konfiguration sowie alle Objekte, die zu der Konfiguration hinzugefügt werden sollen |
| Output | Status, ob das Aktualisieren der Konfiguration erfolgreich war oder nicht |
| Vorbedingung | Objekte müssen im MM-Repository vorhanden sein. |
| Nachbedingung | Wenn das Aktualisieren erfolgreich war, sind die neuen Objekte in der Konfiguration vorhanden und können in weiterer Folge verwendet werden. |
| Aufruf | addObjectsToConfiguration(String id, List<T> addobjects) |

Tabelle 4.31: Funktionsspezifikation: Objekte zu einer Konfiguration hinzufügen

| **Konfiguration löschen** | |
|---|---|
| Beschreibung | Mit Hilfe dieser Funktion kann eine vorhandene Konfiguration aus dem MM-Repository entfernt werden. |
| Input | Konfigurationsidentifikation |
| Output | Status, ob das Löschen der Konfiguration erfolgreich war oder nicht |
| Vorbedingung | Konfiguration muss im MM-Repository vorhanden sein. |
| Nachbedingung | Wenn das Löschen erfolgreich war, ist die Konfiguration nicht mehr im MM-Repository vorhanden und kann nicht mehr verwendet werden. |
| Aufruf | deleteConfiguration(String id) |

Tabelle 4.32: Funktionsspezifikation: Konfiguration erstellen

## 4.3 Externe Schnittstellen

In diesem Kapitel werden alle Benutzerschnittstellen erläutert, welche dem Benutzer über Web-Interfaces zur Verfügung gestellt werden. Da es unterschiedliche Benutzergruppen des MM-Repositories gibt, werden ebenfalls für die unterschiedlichen Schnittstellen die Benutzer definiert.

### 4.3.1 Benutzerverwaltung

Die Benutzerverwaltung wird in dieser Arbeit durch das Metamodellierungswerkzeug ADOxx® realisiert, wodurch die Beschreibung der Benutzerschnittstelle dem Werkzeug entnommen werden kann. Das Werkzeug deckt alle notwendigen Funktionen, wie Anlegen und Löschen von Benutzern und Benutzergruppen sowie die Rechtezuteilung ab. Diese sind bereits im

Administration-Toolkit vorhanden und können daraus entnommen werden. Für die weitere Umsetzung des MM-Repositories sind daher Benutzerschnittstellen für die An- und Abmeldung notwendig.

#### Anmeldung

Der Benutzer muss sich am MM-Repository anmelden, um die unterschiedlichen Anwendungen ausführen zu können. Dabei muss er seinen Benutzernamen und das dazugehörige Passwort angeben. Danach werden die Eingaben überprüft, um sicher zu gehen, dass der Benutzer für die Anwendung autorisiert ist.

Die Eingabemaske benötigt ein Textfeld für den Namen und ein Textfeld für das Passwort, wobei die Zeichen des Passworts durch * angeben werden sollen. Des Weiteren wird ein Login-Button benötigt, um die Anmeldung durchzuführen.

#### Abmeldung

Nachdem der Benutzer sich am MM-Repository angemeldet hat, kann er sich vom MM-Repository wieder abmelden. Nach der Abmeldung können die Anwendungen des MM-Repositories nicht mehr durchgeführt werden.

Für die Abmeldung wird ein Logout-Button benötigt. Dieser soll in allen Interfaces vorhanden sein.

### 4.3.2 Verwaltung von Modellierungsmethoden

#### Anlegen von Objekten

Ein angemeldeter Benutzer mit Administrator-Rechten kann eine Modellierungsmethode oder einzelne Komponenten einer Modellierungsmethode erstellen. Für jedes Objekt, welches im MM-Repository gespeichert werden kann, wird dem Benutzer ein Formular mit allen möglichen Attributen zur Verfügung gestellt. Des Weiteren bekommt er nachdem er den Namen und den Typ des Objekts angegeben hat einen Vorschlag für die Versionsnummer. Nachdem eine neue Version eines Objekts angelegt wurde, muss die aktuelle Version des Objekts als obsolet markiert werden, damit Benutzer, die nach Objekten suchen, die aktuellste Version des Objekts erhalten.

Die Eingabemasken des Formulars sind für jedes Objekt spezifisch. Der Benutzer wird durch die einzelnen Schritte bei der Erstellung eines Modellierungsmethode geführt. Zuerst wird die Modellierungsmethode angelegt, danach alle dazugehörigen Komponenten, wie Modellierungssprache, Vorgehen und Mechanismen. Die Modellierungssprache beinhaltet wiederum Modelltypen, Modellierungsklassen sowie Beziehungsklassen. Modellierungsklassen und Beziehungsklassen besitzen sowohl Attribute für die Beschreibung der Syntax sowie einer Notation. Das Vorgehen ist in Phasen unterteilt, welche wiederum in Schritte untergliedert werden können. Mechanismen einer Modellierungsmethode bestehen aus einer Beschreibung, zusätzlichen Spezifikationsressourcen sowie der Implementierung. Darüber hinaus können die einzelnen Objekte mit zusätzlichen Begriffen annotiert werden, um die Suchergebnisse zu verbessern.

#### Archivieren von Objekten

Objekte, die im MM-Repository vorhanden sind, können von einem angemeldeten Benutzer mit Administrator-Rechten als obsolet markiert werden. Diese Objekte können danach zwar noch gefunden werden, sollten jedoch nicht mehr verwendet werden, da es möglicherweise eine aktuellere Version des Objekts gibt.

Der Benutzer bekommt eine Liste aller Objekte des gewünschten Typs und kann die Objekte auswählen, die er archivieren möchte. Um die Archivierung abzuschließen benötigt er einen Button zur Ausführung der Archivierung, wodurch alle ausgewählten Objekte obsolet gesetzt werden.

### 4.3.3 Suchen

Alle Objekte, die im MM-Repository vorhanden sind, können von angemeldeten Benutzern gesucht bzw. gefunden werden. Dabei gibt es wie bereits in den Anforderungen beschrieben zwei Möglichkeiten der Suche, eine einfache Suche sowie erweiterte Suche.

Für die einfache Suche benötigt der Benutzer lediglich ein Textfeld für die Eingabe des gewünschten Suchbegriffs sowie einen Such-Button, um die Suche zu starten. Bei der erweiterten Suche hat der Benutzer mehrere Einstellungsmöglichkeiten. So kann er den Typ des Objekts wählen und auch entscheiden, ob obsolete Objekte angezeigt bekommen will oder nicht.

Die Ergebnisse werden dem Benutzer übersichtlich dargestellt. Dies bedeutet, dass die Objekte für eine weitere Verwendung, wie Exportieren oder Zusammenstellen einer neuer Konfiguration, aufbereitet werden. Des Weiteren muss es für den Benutzer klar ersichtlich sein, wenn ein archiviertes Objekt im Suchergebnis vorhanden ist.

### 4.3.4 Konfiguration einer neuen Modellierungsmethode

#### Erstellen

Ein angemeldeter Benutzer kann eine neue Zusammenstellung der Komponenten einer Modellierungsmethode erzeugen. Dabei wählt er die notwendigen Komponenten aus und speichert die neue Konfiguration. Bei der Zusammenstellung einer neuen Modellierungsmethode wird der Benutzer ebenfalls vom System unterstützt, da er durch die einzelnen Schritte geleitet wird. Zuerst wählt er die Syntax und Notation der Modellierungssprache aus. Dies bedeutet er wählt Modelltypen, Modellierungsklassen und Beziehungsklassen sowie die gewünschten Attribute und Notationen. Im nächsten Schritt kann der Benutzer das Vorgehen definieren, indem er Phasen und Schritte auswählt. Im letzten Schritt wählt er die benötigten Mechanismen der Modellierungsmethode aus und kann diese den Schritten zuordnen, in welchen diese verwendet werden sollen. Nachdem die neue Modellierungsmethode konfiguriert wurde, kann diese exportiert werden (siehe Kapitel 4.3.5).

In den Formularen für die Konfiguration einer neuen Modellierungsmethode werden dem Benutzer alle vorhandenen Komponenten in allen Versionen dargestellt. Daraus kann er die gewünschten Komponenten und Objekte wählen und diese zu einer Konfiguration zusammenfügen.

#### Verändern

Eine bereits vorhandene Konfiguration kann verändert werden, indem Objekte hinzugefügt oder entfernt werden.

Für das Entfernen benötigt der angemeldete Benutzer eine Liste alle Objekte der Konfiguration. Die zu löschenden Objekte können ausgewählt werden und nach dem betätigen des Update-Buttons werden die Objekte von der Konfiguration entfernt.

Es können jedoch auch Komponenten und Objekte zu einer vorhanden Konfiguration hinzugefügt werden. Dabei wählt der Benutzer die gewünschten Objekte und gibt weiters die Konfiguration an, welche erweitert werden soll. Durch den Update-Button wird die Änderung vorgenommen.

### 4.3.5 Erstellung von Dokumentationen

Der angemeldete Benutzer wählt für die Erstellung einer Dokumentation die gewünschte Konfiguration. Er kann jedoch auch eine vorhandene Modellierungsmethode auswählen. Danach

wählt der Benutzer das gewünschte Format, wobei er zwischen HTML, DOC oder PDF auswählen kann. Der Benutzer hat weiters die Möglichkeit nur für einen Teil der Konfiguration oder Modellierungsmethode eine Dokumentation zu erstellen. Dafür muss der Benutzer die gewünschten Objekte aus einer Liste auswählen. Ein Assistent unterstützt den Benutzer bei der Erstellung der Dokumentation.

Die Eingabemaske für die Erstellung der Dokumentation benötigt eine Liste alle vorhandenen Konfigurationen und Modellierungsmethoden mit allen vorhandenen Objekten, eine Auswahl des gewünschten Formats, die Eingabe des Speicherorts für die Dokumentation sowie einem Create-Button für die Erstellung der gewünschten Dokumentation.

## 4.4 Architektur des MM-Repositories

Für das MM-Repository wurde ein spezielles Metamodell entwickelt, welches die Komponenten einer Modellierungsmethode (siehe Abbildung 2.2) beinhaltet sowie die Realisierung innerhalb der Open Models Initiative berücksichtigt. In diesem Kapitel werden sowohl die Klassenhierarchie (siehe Abbildung 4.2) als auch die Relationen zwischen den Klassen (siehe Abbildung 4.3) dargestellt und beschrieben. Dabei gibt es drei grundsätzliche Kategorien von Objekten, Communitydaten (*OM Project*, *OM Community*, *OM User*, *Location*), Metadaten (*Metadata*, *Modeldata*, *Resource*, *Tool*, *Image*) und Modellierungsmethoden (*Modellingmethod*, *Mechanism*, *Procedure*, *Phase*, *Step*, *Modellinglanguage*, *Modeltype*, *Class*, *Relation*, *Attribute*, *Notation*). Für das MM-Repository werden zusätzlich noch *MM-Repository Object*, *Annotateable Object*, *Versioned Object*, *Configuration* und *Topic* benötigt. Um die Architektur übersichtlicher zu gestalten, sind die Attribute der Klassen nur in der Klassenhierarchie (Abbildung 4.2) abgebildet.

### 4.4.1 Klassenspezifikationen

Um eine einheitliche Darstellung der Klassen zu erzielen, wird eine Schablone (Tabelle 4.33) für die Definition der Klassen verwendet. Diese Schablone beinhaltet Information wie Name, Beschreibung, Oberklasse, Eigenschaften und beteiligte Relationen der Klassen.

| **<Name der Klasse>** | |
|---|---|
| Beschreibung | <Beschreibung der Klasse> |
| Oberklasse | <Name der Oberklasse> |
| Eigenschaften | <Liste der Attribute> |
| Beteiligte Relationen | <Liste der beteiligten Relationen> |

Tabelle 4.33: Schablone für die Spezifikation von Klassen

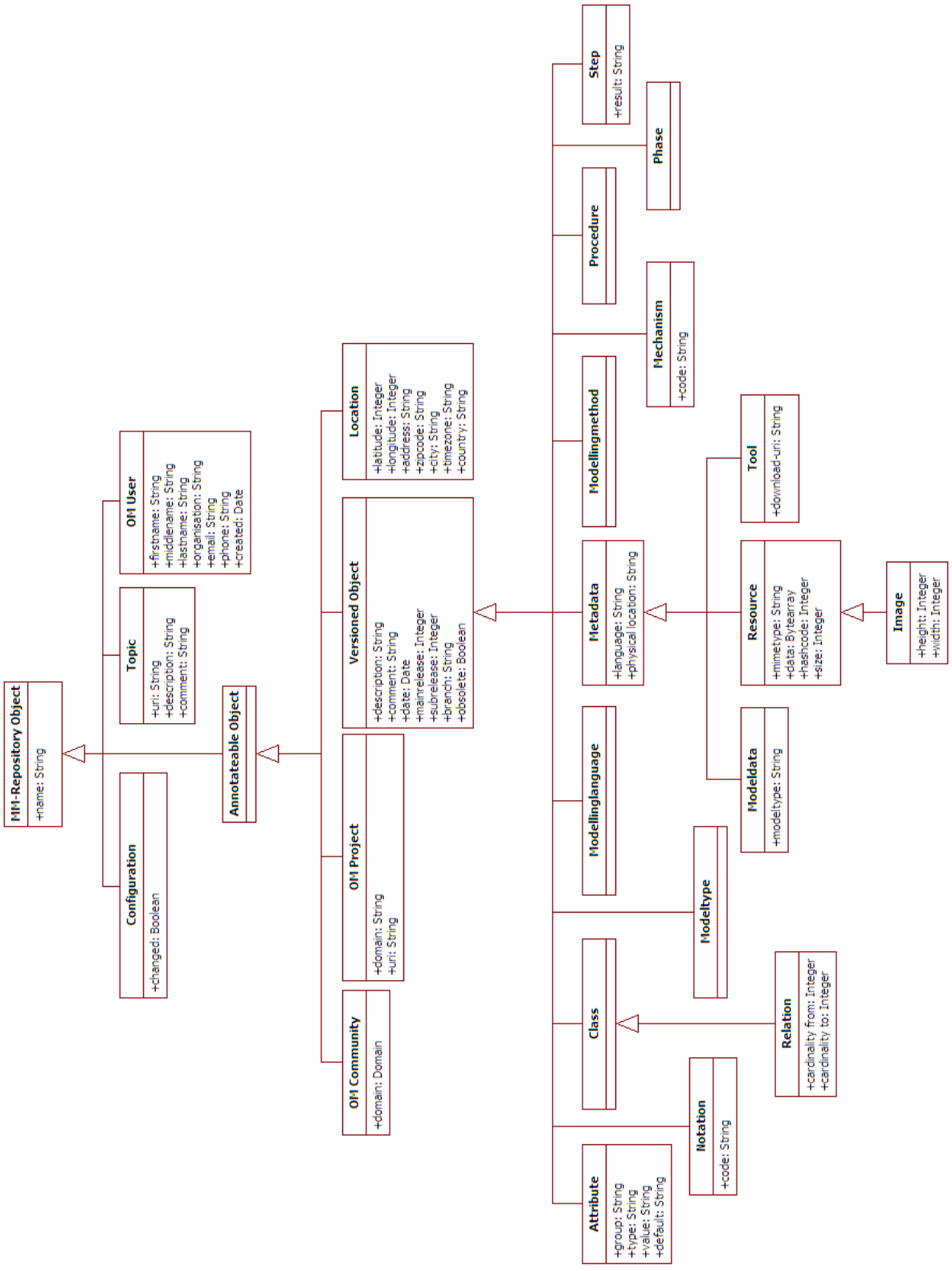



Abbildung 4.2: Klassenhierarchie des MM-Repository-Metamodells

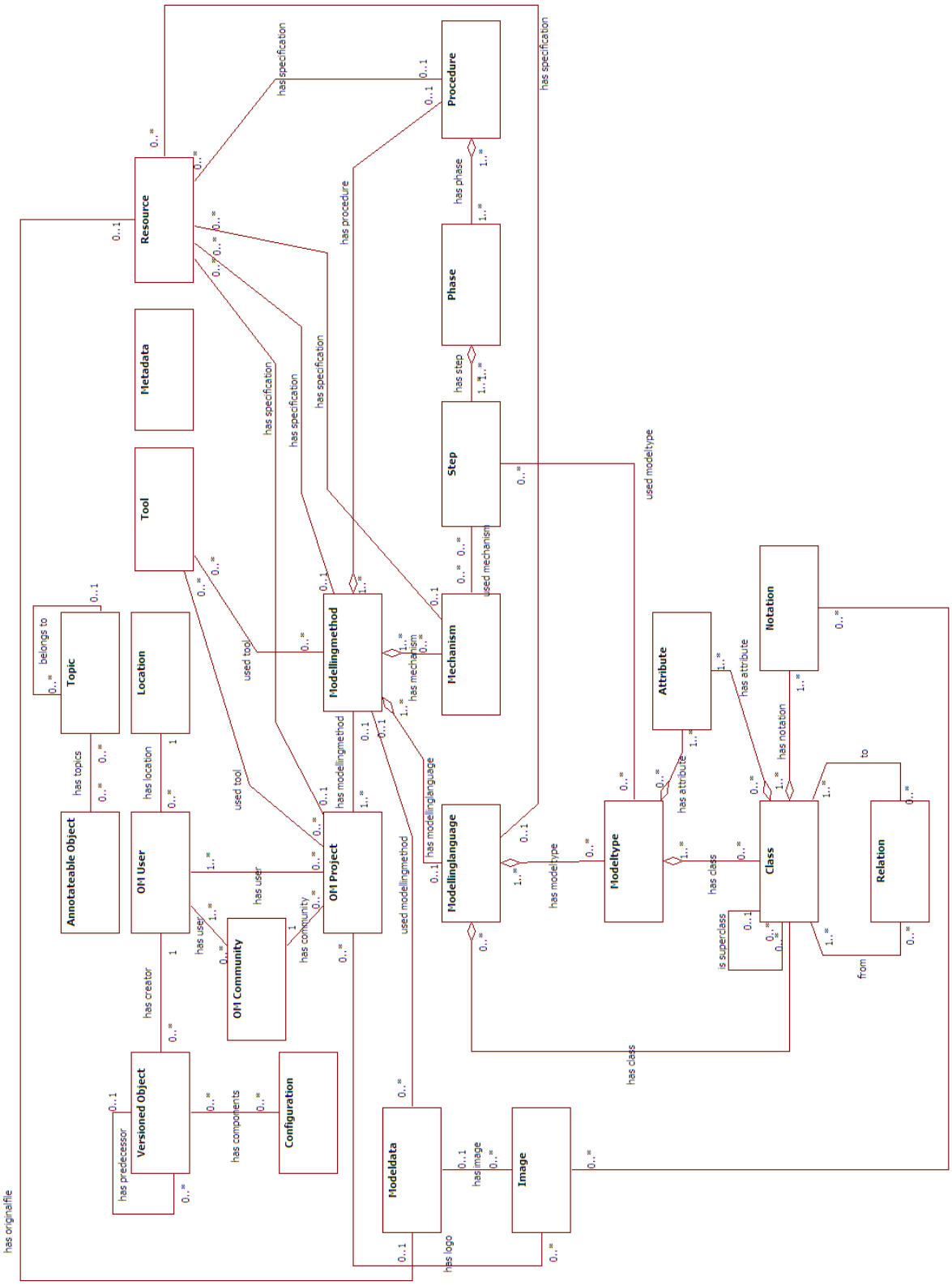



Abbildung 4.3: Relationen innerhalb des MM-Repository-Metamodells

| **MM-Repository Object** | |
|---|---|
| Beschreibung | Ein *MM-Repository Object* beschreibt alle Objekte des MM-Repositories mit einem Namen. |
| Oberklasse | - |
| Eigenschaften | Name des Objekts |
| Beteiligte Relationen | - |

Tabelle 4.34: Klassenspezifikation: MM-Repository Object

| **Annotateable Object** | |
|---|---|
| Beschreibung | Ein *Annotateable Object* kann mit unterschiedlichen Begriffen annotiert werden, um bessere Suchergebnisse erzielen zu können. |
| Oberklasse | *MM-Repository Object* |
| Eigenschaften | Name des Objekts |
| Beteiligte Relationen | *has topic* |

Tabelle 4.35: Klassenspezifikation: Annotateable Object

| **Versioned Object** | |
|---|---|
| Beschreibung | Ein *Versioned Object* beinhaltet weitere Informationen für die Versionierung der Objekte. |
| Oberklasse | *Annotateable Object* |
| Eigenschaften | Name des Objekts<br>Beschreibung des Objekts<br>Kommentar zum Objekt bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components* |

Tabelle 4.36: Klassenspezifikation: Versioned Object

| **Configuration** | |
|---|---|
| Beschreibung | Eine *Configuration* fasst unterschiedliche versionierbare Objekte zusammen. |
| Oberklasse | *MM-Repository Object* |
| Eigenschaften | Name der Konfiguration<br>Markierung, ob sich Objekte der Konfiguration geändert haben |
| Beteiligte Relationen | *has components* |

Tabelle 4.37: Klassenspezifikation: Configuration

| **Topic** | |
|---|---|
| Beschreibung | Ein *Topic* wird für die Annotation von Objekten verwendet und beschreibt diese genauer. |
| Oberklasse | *MM-Repository Object* |
| Eigenschaften | Name des Topics<br>Eindeutige Uniform Resource Identifier (URI) des Topics<br>Beschreibung des Topics<br>Kommentar für das Topic |
| Beteiligte Relationen | *has topic*, *belongs to* |

Tabelle 4.38: Klassenspezifikation: Topic

| **OM Project** | |
|---|---|
| Beschreibung | Ein *OM Project* beschreibt ein Projekt innerhalb der Open Models Initiative. |
| Oberklasse | *Annotateable Object* |
| Eigenschaften | Name des Projekts<br>Domäne, in welcher das Projekt angesiedelt ist<br>Link zur Homepage des Projekts |
| Beteiligte Relationen | *has topic*, *has community*, *has logo*, *has modellingmethod*, *has user* |

Tabelle 4.39: Klassenspezifikation: OM Project

| **OM Community** | |
|---|---|
| Beschreibung | Eine *OM Community* beschreibt eine Community innerhalb der Open Models Initiative. |
| Oberklasse | *Annotateable Object* |
| Eigenschaften | Name der Community<br>Domäne, in welcher die Community angesiedelt ist |
| Beteiligte Relationen | *has topic*, *has community*, *has user* |

Tabelle 4.40: Klassenspezifikation: OM Community

| **OM User** | |
|---|---|
| Beschreibung | Ein *OM User* beschreibt einen Benutzer innerhalb der Open Models Initiative. |
| Oberklasse | *MM-Repository Object* |
| Eigenschaften | Vollständiger Name des Benutzers<br>Vorname des Benutzers<br>Zweiter Vorname des Benutzers<br>Familienname des Benutzers<br>Organisation, in welcher der Benutzer beschäftigt ist<br>E-Mail-Adresse des Benutzers<br>Telefonnummer des Benutzers<br>Datum der Erstellung des Benutzers |
| Beteiligte Relationen | *has location*, *has creator*, *has user* |

Tabelle 4.41: Klassenspezifikation: OM User

| **Location** | |
|---|---|
| Beschreibung | Eine *Location* beschreibt eine bestimmte Position für die Lokalisierung eines Benutzers innerhalb der Open Models Initiative. |
| Oberklasse | *Annotateable Object* |
| Eigenschaften | Name der Position<br>Breitengrad der Position<br>Längengrad der Position<br>Adresse der Position<br>Postleitzahl der Position<br>Stadt der Position<br>Land der Position<br>Zeitzone der Position |
| Beteiligte Relationen | *has topic*, *has location*, *has user* |

Tabelle 4.42: Klassenspezifikation: Location

| **Metadata** | |
|---|---|
| Beschreibung | *Metadata* beinhalten beschreibende Informationen für unterschiedliche Objekte. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name des Objekts<br>Beschreibung des Objekts<br>Kommentar zum Objekt bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Sprache des Objekts<br>Physischer Speicherort des Objekts |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components* |

Tabelle 4.43: Klassenspezifikation: Metadata

| **Modeldata** | |
|---|---|
| Beschreibung | *Modeldata* beschreiben ein Modell innerhalb der Open Models Initiative genauer. |
| Oberklasse | *Metadata* |
| Eigenschaften | Name des Modells<br>Beschreibung des Modells<br>Kommentar zum Modell bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Sprache des Modells<br>Physischer Speicherort des Modells<br>Typ des Modells |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has image*, *used modellingmethod*, *has originalfile* |

Tabelle 4.44: Klassenspezifikation: Modeldata

| **Resource** | |
|---|---|
| Beschreibung | *Resource* beschreibt ein Objekt mit einem MIME-Typ innerhalb der Open Models Initiative genauer. |
| Oberklasse | *Metadata* |
| Eigenschaften | Name der Ressource<br>Beschreibung der Ressource<br>Kommentar zur Ressource bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Sprache der Ressource<br>Physischer Speicherort der Ressource<br>MIME-Typ der Ressource<br>Daten für Speicherung der Ressource als Bytearray<br>Hashcode für die Überprüfung der Daten<br>Größe für die Überprüfung der Daten |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has originalfile* |

Tabelle 4.45: Klassenspezifikation: Resource

| **Tool** | |
|---|---|
| Beschreibung | Ein *Tool* beschreibt ein Werkzeug im Zusammenhang mit Modellierungsmethoden genauer. |
| Oberklasse | *Metadata* |
| Eigenschaften | Name des Tools<br>Beschreibung des Tools<br>Kommentar zum Tool bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Sprache des Tools<br>Physischer Speicherort des Tools<br>Link zum Download des Tools |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *used tool* |

Tabelle 4.46: Klassenspezifikation: Tool

| **Image** | |
|---|---|
| Beschreibung | *Image* beschreibt ein Bild genauer. |
| Oberklasse | *Metadata* |
| Eigenschaften | Name des Bilds<br>Beschreibung des Bilds<br>Kommentar zum Bild bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Sprache des Objekts<br>Physischer Speicherort des Bilds<br>MIME-Typ des Bilds<br>Daten für Speicherung des Bilds als Bytearray<br>Hashcode für die Überprüfung des Bilds<br>Größe für die Überprüfung des Bilds<br>Höhe des Bilds<br>Breite des Bilds |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has originalfile* |

Tabelle 4.47: Klassenspezifikation: Image

| **Modellingmethod** | |
|---|---|
| Beschreibung | Eine *Modellingmethod* beinhaltet Informationen zu einer Modellierungsmethode. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name der Modellierungsmethode<br>Beschreibung der Modellierungsmethode<br>Kommentar zur Modellierungsmethode bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has mechanism*, *has modellinglanguage*, *has procedure*, *has specification*, *used tool*, *has modellingmethod*, *used modellingmethod* |

Tabelle 4.48: Klassenspezifikation: Modellingmethod

| **Mechanism** | |
|---|---|
| Beschreibung | Mit *Mechanism* werden die unterschiedlichen Mechanismen einer Modellierungsmethode beschrieben. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name des Mechanismus<br>Beschreibung des Mechanismus<br>Kommentar zum Mechanismus bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Implementierungscode des Mechanismus |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has specification*, *used mechanism*, *has mechanism* |

Tabelle 4.49: Klassenspezifikation: Mechanism

| **Procedure** | |
|---|---|
| Beschreibung | Eine *Procedure* beschreibt das Vorgehen einer Modellierungsmethode genauer. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name des Vorgehens<br>Beschreibung des Vorgehens<br>Kommentar zum Vorgehen bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has specification*, *has phase*, *used mechanism*, *has procedure* |

Tabelle 4.50: Klassenspezifikation: Procedure

| **Phase** | |
|---|---|
| Beschreibung | Eine *Phase* beschreibt die Phasen eines Vorgehens genauer. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name der Phase<br>Beschreibung der Phase<br>Kommentar zur Phase bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has phase*, *has step* |

Tabelle 4.51: Klassenspezifikation: Phase

| **Step** | |
|---|---|
| Beschreibung | Ein *Step* beschreibt die Schritte einer Phase genauer. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name des Steps<br>Beschreibung des Steps<br>Kommentar zum Step bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has specification*, *used mechanism*, *used modeltype*, *has step* |

Tabelle 4.52: Klassenspezifikation: Step

| **Modellinglanguage** | |
|---|---|
| Beschreibung | Eine *Modellinglanguage* beinhaltet Informationen zu einer Modellierungssprache einer Modellierungsmethode. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name der Modellierungssprache<br>Beschreibung der Modellierungssprache<br>Kommentar zur Modellierungssprache bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has specification*, *has modellinglanguage*, *has class*, *has modeltype* |

Tabelle 4.53: Klassenspezifikation: Modellinglanguage

| **Modeltype** | |
|---|---|
| Beschreibung | Ein *Modeltype* fasst Informationen eines Modelltypes zusammen. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name des Modelltyps<br>Beschreibung des Modelltyps<br>Kommentar zum Modelltyp bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has attribute*, *has modeltype*, *has class*, *used modeltype* |

Tabelle 4.54: Klassenspezifikation: Modeltype

| **Class** | |
|---|---|
| Beschreibung | Eine *Class* beschreibt eine Klasse mit den notwendigen Informationen. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name der Klasse<br>Beschreibung der Klasse<br>Kommentar zur Klasse bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has attribute*, *has notation*, *has class*, *has superclass*, *used modeltype*, *from*, *to* |

Tabelle 4.55: Klassenspezifikation: Class

| **Relation** | |
|---|---|
| Beschreibung | Eine *Relation* beschreibt eine Relation zwischen Klassen mit den notwendigen Informationen. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name der Relation<br>Beschreibung der Relation<br>Kommentar zur Relation bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Kardinalität des Ausgangspunkts<br>Kardinalität des Endpunkts |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has attribute*, *has notation*, *has class*, *has superclass*, *used modeltype*, *from*, *to* |

Tabelle 4.56: Klassenspezifikation: Relation

| **Attribute** | |
|---|---|
| Beschreibung | Ein *Attribute* beschreibt die Syntax einer Klasse. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name des Attributs<br>Beschreibung des Attributs<br>Kommentar zum Attribute bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Gruppierung des Attributs<br>Typ des Attributs<br>Werte des Attributs<br>Standardwert des Attributs |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has attribute* |

Tabelle 4.57: Klassenspezifikation: Attribute

| **Notation** | |
|---|---|
| Beschreibung | Eine *Notation* beschreibt die grafische Repräsentation einer Klasse. |
| Oberklasse | *Versioned Object* |
| Eigenschaften | Name der Notation<br>Beschreibung der Notation<br>Kommentar zur Notation bzw. der Version<br>Erstellungsdatum der Version<br>Mainrelease-Nummer der Version<br>Subrelease-Nummer der Version<br>Bezeichnung des Entwicklungszweiges<br>Markierung für eine obsolete Version<br>Code der Notation |
| Beteiligte Relationen | *has topic*, *has creator*, *has predecessor*, *has components*, *has image*, *has notation* |

Tabelle 4.58: Klassenspezifikation: Notation

### 4.4.2 Relationsspezifikationen

In Abbildung 4.3 sind alle Relationen abgebildet. Für eine einheitliche Spezifikation der Relationen werden in diesem Kapitel die vorhanden Relationen in tabellarischer Form dargestellt

und verwenden die Schablone, welche in Tabelle 4.59 definiert wird. Die beteiligten Klassen werden in Ausgangs- und Endpunkt unterteilt, welche auch die Kardinalität beinhalten.

| **<Name der Relation>** | |
|---|---|
| Beschreibung | <Beschreibung der Klasse> |
| Ausgangspunkt | <Liste der ausgehenden Klassen mit deren Kardinalität> |
| Endpunkt | <Liste der eingehenden Klassen mit deren Kardinalität> |

Tabelle 4.59: Schablone für die Spezifikation von Relationen

Nachfolgend werden alle Relationen des MM-Repositories einzeln laut Schablone (siehe Tabelle 4.59) spezifiziert.

| **belongs to** | |
|---|---|
| Beschreibung | Mithilfe der Relation *belongs to* wird eine Hierarchie innerhalb der *Topics* erstellt. |
| Ausgangspunkt | *Topic* (0..*) |
| Endpunkt | *Topic* (0..1) |

Tabelle 4.60: Relationsspezifikation: belongs to

| **has topic** | |
|---|---|
| Beschreibung | Ein *Annotateable Object* kann mit ein oder mehreren *Topics* annotiert werden, wobei ein *Topic* auch für mehrere *Annotateable Objects* verwendet werden kann. |
| Ausgangspunkt | *Annotateable Object* (0..*) |
| Endpunkt | *Topic* (0..*) |

Tabelle 4.61: Relationsspezifikation: has topic

| **has location** | |
|---|---|
| Beschreibung | Zu einem *OM User* wird die *Location*, wo er sich gerade befindet, gespeichert. Dabei kann eine *Location* für mehrere, unterschiedliche *OM User* verwendet werden. |
| Ausgangspunkt | *OM User* (0..*) |
| Endpunkt | *Location* (1) |

Tabelle 4.62: Relationsspezifikation: has location

| **has user** | |
|---|---|
| Beschreibung | Ein *OM Project* sowie eine *OM Community* hat mehrere *OM User*, die innerhalb des Projekts oder der Community tätig sind. Ein *OM User* kann wiederum in mehreren *OM Projects* und *OM Communities* tätig sein. |
| Ausgangspunkt | *OM Project* (0..*)<br>*OM Community* (0..*) |
| Endpunkt | *OM User* (1..*) |

Tabelle 4.63: Relationsspezifikation: has user

| **has community** | |
|---|---|
| Beschreibung | Ein *OM Project* ist innerhalb genau einer *OM Community* tätig. Ein *OM Commmunity* kann mehrere *OM Projects* beinhalten. |
| Ausgangspunkt | *OM Project* (0..*) |
| Endpunkt | *OM Community* (1) |

Tabelle 4.64: Relationsspezifikation: has community

| **has logo** | |
|---|---|
| Beschreibung | Ein *OM Project* kann mehrere *Images* als Logo beinhalten, wobei ein *Image* in mehreren *OM Projects* verwendet werden kann. |
| Ausgangspunkt | *OM Project* (0..*) |
| Endpunkt | *Image* (0..*) |

Tabelle 4.65: Relationsspezifikation: has logo

| **has creator** | |
|---|---|
| Beschreibung | Ein *Versioned Object* wird von genau einem *OM User* erzeugt. Ein *OM User* kann mehrere *Versioned Objects* erzeugen. |
| Ausgangspunkt | *Versioned Object* (0..*) |
| Endpunkt | *OM User* (1) |

Tabelle 4.66: Relationsspezifikation: has creator

| **has predecessor** | |
|---|---|
| Beschreibung | Ein *Versioned Object* kann eine Vorgängerversion vom Typ *Versioned Object* haben, wobei eine Vorgängerversion mehrere *Versioned Objects* als Nachfolgerversionen haben kann. |
| Ausgangspunkt | *Versioned Object* (0..*) |
| Endpunkt | *Versioned Object* (0..1) |

Tabelle 4.67: Relationsspezifikation: has predecessor

| **has components** | |
|---|---|
| Beschreibung | Eine *Configuration* kann aus mehreren *Versioned Objects* bestehen, welche wiederum in mehreren *Configurations* sein können. |
| Ausgangspunkt | *Configuration* (0..*) |
| Endpunkt | *Versioned Object* (0..*) |

Tabelle 4.68: Relationsspezifikation: has components

| **used tool** | |
|---|---|
| Beschreibung | Ein *OM Project* und eine *Modellingmethod* können *Tools* vorschlagen, die verwendet werden können. Ein *Tool* kann wiederum in mehreren *OM Projects* und *Modellingmethods* verwendet werden. |
| Ausgangspunkt | *OM Project* (0..*)<br>*Modellingmethod* (0..*) |
| Endpunkt | *Tool* (0..*) |

Tabelle 4.69: Relationsspezifikation: used tool

| **has originalfile** | |
|---|---|
| Beschreibung | Die Originaldatei eines *Modeldata* kann auch als *Resource* abgespeichert werden, wobei eine *Resource* zu einem *Modeldata* zugeordnet werden kann. |
| Ausgangspunkt | *Modeldata* (0..1) |
| Endpunkt | *Resource* (0..1) |

Tabelle 4.70: Relationsspezifikation: has originalfile

| **has specification** | |
|---|---|
| Beschreibung | *OM Project*, *Modellingmethod*, *Mechanism*, *Procedure* und *Modellinglanguage* können *Resources* als Spezifikation beinhalten. Eine *Resource* kann zu einem *OM Project*, *Modellingmethod*, *Mechanism*, *Procedure* oder *Modellinglanguage* zugeordnet werden. |
| Ausgangspunkt | *OM Project* (0..*)<br>*Modellingmethod* (0..*)<br>*Mechanism* (0..*)<br>*Procedure* (0..*)<br>*Modellinglanguage* (0..*) |
| Endpunkt | *Resource* (0..1) |

Tabelle 4.71: Relationsspezifikation: has specification

| **has modellingmethod** | |
|---|---|
| Beschreibung | Ein *OM Project* kann mehrere *Modellingmethods* verwalten. Eine *Modellingmethod* kann wiederum in mehreren *OM Projects* verwendet werden, muss jedoch mindestens einem *OM Project* zugeteilt sein. |
| Ausgangspunkt | *OM Project* (1..*) |
| Endpunkt | *Modellingmethod* (0..*) |

Tabelle 4.72: Relationsspezifikation: has modellingmethod

| **used modellingmethod** | |
|---|---|
| Beschreibung | Für die Erstellung eine Modells kann eine *Modellingmethod* verwendet werden, die im MM-Repository vorhanden ist. Eine *Modellingmethod* kann für mehrere Modelle verwendet werden. |
| Ausgangspunkt | *Modeldata* (0..*) |
| Endpunkt | *Modellingmethod* (0..1) |

Tabelle 4.73: Relationsspezifikation: used modellingmethod

| **has image** | |
|---|---|
| Beschreibung | *Modeldata* und *Notations* können *Images* als zusätzliche Information haben. Ein *Image* kann wiederum zu mehreren *Modeldata* und *Notations* zugeordnet werden. |
| Ausgangspunkt | *Modeldata* (0..1)<br>*Notation* (0..*) |
| Endpunkt | *Image* (0..*) |

Tabelle 4.74: Relationsspezifikation: has image

| **has procedure** | |
|---|---|
| Beschreibung | Eine *Modellingmethod* kann eine *Procedure* definiert haben. Eine *Procedure* kann für mehrere *Modellingmethods* verwendet werden, muss jedoch mindestens einer *Modellingmethod* zugeteilt sein. |
| Ausgangspunkt | *Modellingmethod* (1..*) |
| Endpunkt | *Procedure* (0..1) |

Tabelle 4.75: Relationsspezifikation: has procedure

| **has phase** | |
|---|---|
| Beschreibung | Eine *Procedure* besteht aus ein oder mehreren *Phases*. Eine *Phase* kann für mehrere *Procedures* verwendet werden, muss jedoch mindestens einer *Procedure* zugeteilt sein. |
| Ausgangspunkt | *Procedure* (1..*) |
| Endpunkt | *Phase* (1..*) |

Tabelle 4.76: Relationsspezifikation: has phase

| **has step** | |
|---|---|
| Beschreibung | Eine *Phase* besteht aus ein oder mehreren *Steps*. Eine *Step* kann für mehrere *Phases* verwendet werden, muss jedoch mindestens einer *Phse* zugeteilt sein. |
| Ausgangspunkt | *Phase* (1..*) |
| Endpunkt | *Step* (1..*) |

Tabelle 4.77: Relationsspezifikation: has step

| **used mechanism** | |
|---|---|
| Beschreibung | Innerhalb eines *Steps* können mehrere *Mechanism* verwendet werden. Eine *Mechanism* kann von mehreren *Steps* verwendet werden. |
| Ausgangspunkt | *Steps* (0..*) |
| Endpunkt | *Mechanism* (0..*) |

Tabelle 4.78: Relationsspezifikation: used mechanism

| **used modeltype** | |
|---|---|
| Beschreibung | Innerhalb eines *Steps* können mehrere *Modeltypes* verwendet werden. Ein *Modeltyp* kann von mehreren *Steps* verwendet werden. |
| Ausgangspunkt | *Steps* (0..*) |
| Endpunkt | *Modeltype* (0..*) |

Tabelle 4.79: Relationsspezifikation: used modeltype

| **has mechanism** | |
|---|---|
| Beschreibung | Eine *Modellingmethod* kann mehrere *Mechanism* definiert haben. Eine *Mechanism* kann für mehrere *Modellingmethods* verwendet werden, muss jedoch mindestens einer *Modellingmethod* zugeteilt sein. |
| Ausgangspunkt | *Modellingmethod* (1..*) |
| Endpunkt | *Mechanism* (0..*) |

Tabelle 4.80: Relationsspezifikation: has mechanism

| **has modellinglanguage** | |
|---|---|
| Beschreibung | Eine *Modellingmethod* kann eine *Modellinglanguage* definiert haben. Eine *Modellinglanguage* kann für mehrere *Modellingmethods* verwendet werden, muss jedoch mindestens einer *Modellingmethod* zugeteilt sein. |
| Ausgangspunkt | *Modellingmethod* (1..*) |
| Endpunkt | *Modellinglanguage* (0..1) |

Tabelle 4.81: Relationsspezifikation: has modellinglanguage

| **has modeltype** | |
|---|---|
| Beschreibung | Eine *Modellinglanguage* kann mehrere *Modeltypes* definiert haben. Ein *Modeltype* kann für mehrere *Modellinglanguages* verwendet werden, muss jedoch mindestens einer *Modellinglanguage* zugeteilt sein. |
| Ausgangspunkt | *Modellinglanguage* (1..*) |
| Endpunkt | *Modeltype* (0..*) |

Tabelle 4.82: Relationsspezifikation: has modeltype

| **has class** | |
|---|---|
| Beschreibung | Eine *Modellinglanguage* und ein *Modeltype* können mehrere *Classes* definiert haben. Eine *Class* kann für mehrere *Modellinglanguages* oder *Modeltypes* verwendet werden. |
| Ausgangspunkt | *Modellinglanguage* (0..*)<br>*Modeltype* (0..*) |
| Endpunkt | *Class* (0..*) |

Tabelle 4.83: Relationsspezifikation: has class

| **is superclass** | |
|---|---|
| Beschreibung | Um Vererbung zu realisieren, können Superklassen definiert werden. Eine *Class* kann eine *Class* als Superklasse haben. Eine *Class* kann Superklasse von mehreren *Classes* sein. |
| Ausgangspunkt | *Class* (0..1) |
| Endpunkt | *Class* (0..*) |

Tabelle 4.84: Relationsspezifikation: is superclass

| **has attribute** | |
|---|---|
| Beschreibung | Ein *Modeltype* und eine *Class* bestehen aus ein oder mehreren *Attributes*. Ein *Attribute* kann für mehrere *Modeltypes* oder *Classes* verwendet werden. |
| Ausgangspunkt | *Class* (0..*)<br>*Modeltype* (0..*) |
| Endpunkt | *Attribute* (1..*) |

Tabelle 4.85: Relationsspezifikation: has attribute

| **has notation** | |
|---|---|
| Beschreibung | Eine *Class* bestehen aus ein oder mehreren *Notations*. Ein *Notation* kann für mehrere *Modeltypes* oder *Classes* verwendet werden, muss jedoch zumindest in einer *Class* verwendet werden. |
| Ausgangspunkt | *Class* (1..*) |
| Endpunkt | *Notation* (1..*) |

Tabelle 4.86: Relationsspezifikation: has notation

| **from** | |
|---|---|
| Beschreibung | Eine *Relation* bestehen aus ein oder mehreren *Classes*, welche die Ausgangspunkte darstellen. Eine *Class* kann für mehrere *Relations* verwendet werden. |
| Ausgangspunkt | *Relation* (0..*) |
| Endpunkt | *Class* (1..*) |

Tabelle 4.87: Relationsspezifikation: from

| **to** | |
|---|---|
| Beschreibung | Eine *Relation* bestehen aus ein oder mehreren *Classes*, welche die Endpunkte darstellen. Eine *Class* kann für mehrere *Relations* verwendet werden. |
| Ausgangspunkt | *Relation* (0..*) |
| Endpunkt | *Class* (1..*) |

Tabelle 4.88: Relationsspezifikation: to

## 4.5 Benutzerinteraktion

In Abbildung 4.4 werden alle Möglichkeiten für die Interaktion zwischen Benutzer und MM-Repository dargestellt. Das MM-Repository unterscheidet zwischen zwei unterschiedlichen Benutzergruppen, wobei der Administrator ebenfalls alle Funktionen des eingeschränkten Benutzers aufrufen darf. Des Weiteren ist zu beachten, dass der Benutzer angemeldet sein muss, um die Funktionen des MM-Repositories nutzen zu können. Das MM-Repository wird in die vier unterschiedlichen Bereiche Zugriffsmanagement, Versionsmanagement, Content-Management sowie Abfragenmanagement und Publisching unterteilt.

Interaktionsmöglichkeiten für den eingeschränkten Benutzer im Bereich Zugriffsmanagement sind vor allem Anmelden und Abmelden. Der Administrator darf zusätzlich Benutzer anlegen und löschen, Benutzergruppen anlegen und löschen sowie Rechte zu den Benutzern und Benutzergruppen zuweisen.

Innerhalb des Abfragenmanagements und Publishing kann der eingeschränkte Benutzer die beiden Suchfunktionen verwenden. Des Weiteren hat er hier die Möglichkeit eine Modellierungsmethode zu erstellen, eine neue Konfiguration zusammenzustellen oder die Dokumentation zu einer Modellierungsmethode oder Konfiguration zu erzeugen. Die Suche wird durch das Aufrufen von Abfragen realisiert. Abfragen können vom Administrator erstellt und gelöscht werden.

Das Versionsmanagement beinhaltet für die eingeschränkten Benutzer die Möglichkeiten Abhängigkeiten der unterschiedlichen Objekte präsentiert zu bekommen und Versionen zu vergleichen. Der Administrator kann im Versionsmanagement die aktuelle Version eines Objekts festlegen und Verzweigungen (Branches) erzeugen. Die Erstellung eines neuen Zweiges beinhaltet das Erzeugen eines neuen Objekts.

Im Content-Management stehen dem eingeschränkten Benutzer keine Interaktionsmöglichkeiten zur Verfügung. Der Administrator kann jedoch Objekte erzeugen, archivieren und aktualisieren.

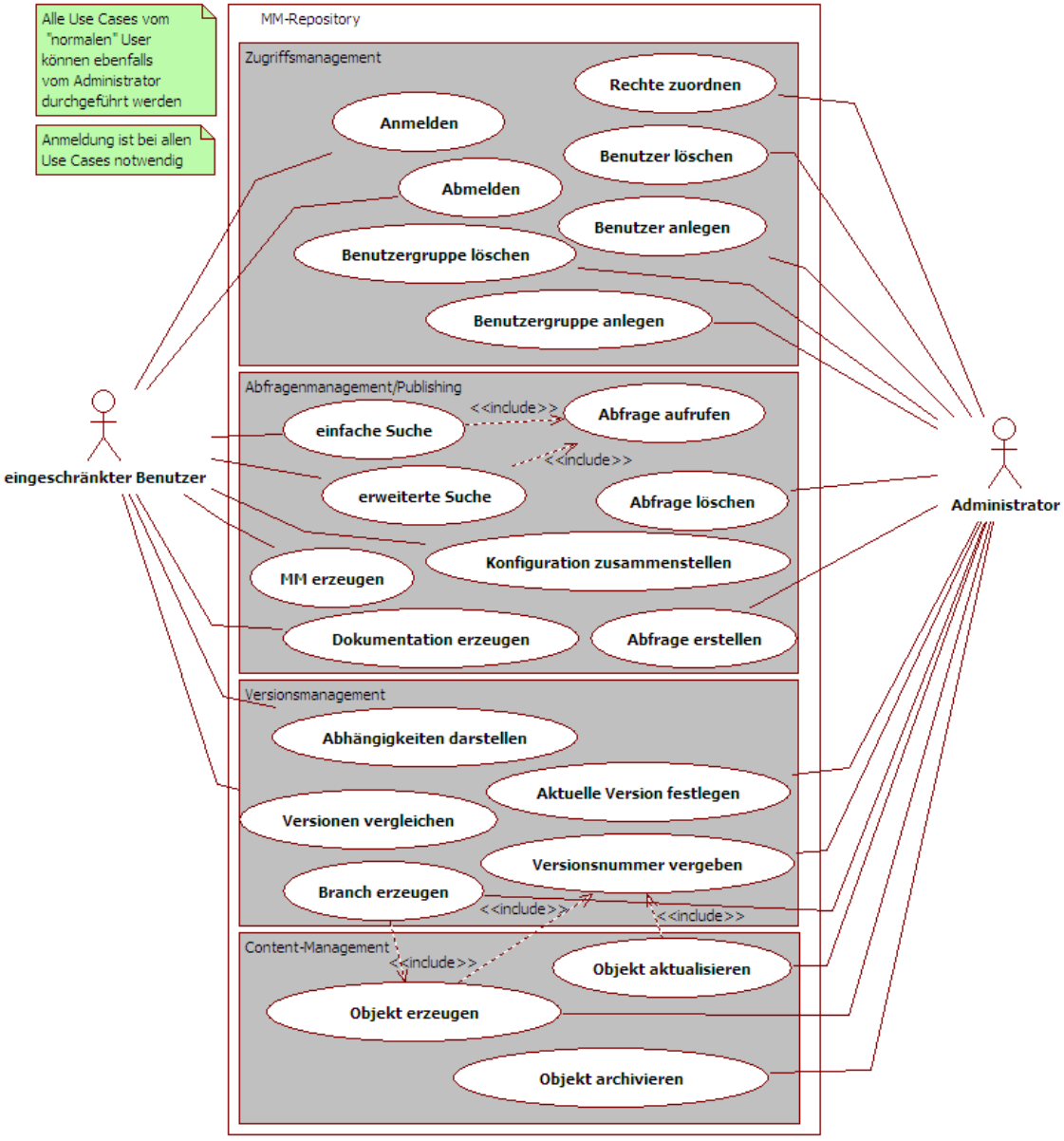



Abbildung 4.4: Use-Case Diagramm

Beim Erstellen und Aktualisieren von Objekten werden vom System oder vom Administrator Versionsnummern vergeben.

## 4.6 Vergleich dieses Konzepts mit datenbank-basierten Konzepten

In diesem Kapitel werden sowohl die Vorteile als auch die Nachteile eines Konzepts auf Basis einer Metamodellierungsplattform aufgelistet und anschließend eine Schlussfolgerung für die Umsetzung des MM-Repositories diskutiert.

### 4.6.1 Vorteile

Durch die Verwendung einer Metamodellierungsplattform wie ADOxx® werden diverse Funktionalitäten, die für ein MM-Repository benötigt werden, bereits mitgeliefert. Zu diesen Funktionalitäten und Mechanismen zählen unter anderem lose Kopplung zwischen Metamodell und Objekten des MM-Repositories, einfache Archivierung, einfache Erstellung von Analysen, einfache Erstellung von Sichten, User- und Rechtemanagement, Mehrsprachigkeit, vorgefertigte Lösungen für Publishing-Möglichkeiten, grafische Toolunterstützung bei der Erstellung des Metamodells (Administration-Toolkit) und eine integrierte Modellierungsumgebung für grafische Repräsentation und Manipulation der Objekte.

#### Lose Kopplung zwischen Metamodell und Objekten

Bei der Entwicklung von Datenbank-Schemata ist es wichtig, dass sich die Struktur der Daten nach dem Umsetzen nicht mehr ändert, da eine Änderung des Datenschemas zu Problemen führt und eine Anpassung des Datenschemas nur bedingt möglich ist. Durch den Metamodellierungsansatz besteht eine geringere Bindung zwischen der Datenstruktur und den Objekten des MM-Repositories. So können Erweiterungen im Metamodell problemlos durchgeführt werden, ohne die bereits gespeicherten Daten ändern zu müssen. Beim Entfernen muss beachtet werden, dass bereits erstellte Daten ebenfalls entfernt werden, jedoch ist es problemlos möglich.

#### Archivieren von Objekten

In ADOxx® gibt es die Möglichkeit, Objekte nicht endgültig zu löschen, sondern in einen Papierkorb verschieben zu können. Diese sind danach nicht mehr verwendbar, können aber wiederhergestellt werden. Dies ist ein Vorteil für dieses MM-Repository, da alle Objekte zu jedem Zeitpunkt vorhanden sein sollen.

### Einfache Erstellung von Analysen und Sichten

Da ADOxx® nicht nur eine Metamodellierungsumgebung ist sondern ebenfalls eine Modellierungsumgebung mitliefert, ist es möglich die Inhalte des MM-Repositories mithilfe dieser Umgebung zu analysieren und unterschiedliche Sichten auf die Inhalte zu definieren. Für die Analyse bietet ADOxx® sowohl vordefinierte Abfragen sowie benutzerdefinierte Abfragen. Diese Abfragen können einfach mittels JavaScript ausgeführt, angepasst und erstellt werden.

In ADOxx® ist es des Weiteren möglich unterschiedliche Sichten auf die Modelle zu definieren. Mit der Definition von Sichten können sowohl Objekte als auch Attribute aus- oder eingeblendet werden. Durch diese Möglichkeit kann die Implementierung des Kontextmanagements vereinfacht werden, indem für die unterschiedlichen Anwendungsbereiche unterschiedliche Sichten definiert werden.

### User- und Rechtemanagement

ADOxx® besitzt bereits ein umfangreiches User- und Rechtemanagement. Mit Hilfe des Usermanagements können sowohl unterschiedliche Benutzergruppen als auch einzelne Benutzer verwaltet werden. Eigenschaften, die beim Benutzer eingestellt werden können, sind unter anderem Benutzername, Vorname, Nachname, E-Mail-Adresse, Passwort und benutzerspezifische Informationen. Das Rechtemanagement von ADOxx® funktioniert auf unterschiedlichen Ebenen. So können Rechte für Modelle und Modellgruppen, Objekte und Objektgruppen, Benutzer und Benutzergruppen, Dateien und Dateigruppen, Komponenten sowie Sprachen definiert werden. Grundsätzlich kann man bei den Rechten zwischen erlaubt und verboten unterscheiden. Beim Erlauben des Zugriffs kann weiters zwischen lesendem und schreibendem Zugriff unterschieden werden. Es gibt noch einige spezielle Formen der Rechte, die jedoch in diesem Zusammenhang nicht relevant sind.

### Mehrsprachigkeit

Für die Gewährleistung der Mehrsprachigkeit werden in ADOxx® unterschiedliche Sichten definiert und der Benutzer kann zwischen unterschiedlichen Sprachen wählen. Die Umsetzung der Mehrsprachigkeit kann einfach durch die Eingabe der unterschiedlichen Sprachen innerhalb des Metamodells realisiert werden.

#### Vorgefertige Lösungen für Publishingmöglichkeiten

ADOxx® bietet bereits einige Möglichkeiten der Veröffentlichung von Objekten und Modellen. Daher können vorhandene Lösungen verwendet werden, um die Inhalte des MM-Repositories in unterschiedlichen Ausgabeformaten anzuzeigen. Für die Ausgabe des Inhalts müssen nur wenige Anpassungen durchgeführt werden. Die Objekte des MM-Repositories können standardmäßig als DOC, HTML oder PDF exportiert werden.

#### Administration-Toolkit

ADOxx® stellt für die Administration ein eigenes Werkzeug zur Verfügung. Damit kann das Metamodell (siehe Kapitel 4.4) mittels grafischer Toolunterstüztung erstellt werden. Das Administration-Toolkit wird ebenfalls für die Verwaltung der Benutzer und Benutzerrechte verwendet. Zusätzliche Einstellungen und Funktionalitäten können mit dem Administration-Toolkit durchgeführt werden. Der Benutzer des Administration-Toolkits benötigt grundlegendes Wissen im Bereich Modellierung und wird durch das Werkzeug gut unterstützt.

#### Integrierte Modellierungsumgebung

Da ADOxx® eine eigene Modellierungsumgebung zur Verfügung stellt, kann diese einfach in das MM-Repository eingebunden werden. So können die Objekte des MM-Repositories innerhalb dieser Umgebung modelliert werden und anschließend im MM-Repository mit den entsprechenden beschreibenden Informationen versehen werden. Es ist ebenfalls möglich, dass die Modellierungsmethoden, die im MM-Repository verwaltet werden, in ADOxx® geladen und anschließend verwendet werden können.

Durch die von ADOxx® mitgelieferte Modellierungsumgebung können die Inhalte des MM-Repositories einfach durch grafische Unterstützung repräsentiert und manipuliert werden. Dabei sind alle gewohnten Funktionalitäten der Modellierungsumgebung vorhanden. Es können Objekte und Modelle angelegt werden, Referenzen zwischen Objekten erzeugt werden sowie andere Funktionalitäten, wie Analyse und Veröffentlichung, verwendet werden.

### 4.6.2 Nachteile

Jedoch hat ein MM-Repositorykonzept auf Basis einer Metamodellierungsplattform, wie ADOxx®, auch einige Nachteile im Gegensatz zur Verwendung von Datenbank-basierten Repositories, welche bereits für die unterschiedlichsten Anwendungsgebiete entwickelt wurden.

Dadurch sind folgende Nachteile gegeben: keine Caching-Möglichkeiten und keine Versionierungsmechanismen. Diese müssen zusätzlich implementiert werden.

#### Caching

Ein Nachteil von Metamodellierungsplattformen ist die zusätzliche Implementierung von Cachingmechanismen. Cachingmechanismen im Bereich Datenbanken sind auf unterschiedlichen Ebenen, wie Daten- und Abfragen-Ebene, implementiert und können einfach verwendet werden. Unterschiedliche Caching-Verfahren in Datenbanken werden in Härder u. Bühmann [2004] analysiert und verglichen. Für die Optimierung des Zeitverhaltens und der Verfügbarkeit des MM-Repositories müssen geeignete Cachingmechanismen implementiert werden. Jedoch wird Caching nicht mehr genauer in dieser Arbeit betrachtet, da es sich um eine Forschungsarbeit handelt und Performanceoptimierung erst in weiterer Folge an Bedeutung gewinnt.

#### Versionsmanagement

Datenbanken bieten bereits unterschiedliche Mechanismen für die Versionierung der Daten an. In ADOxx® muss die Versionierung von Objekten zusätzlich implementiert werden. Durch die eigene Implementierung kann das Versionsmanagement genau an die Bedürfnisse des MM-Repositories angepasst werden, so können dem Benutzer Vorschläge für Vorgängerversion oder ähnliches angeboten werden.

### 4.6.3 Schlussfolgerung

Es gibt jedoch auch einige Funktionalitäten, die sowohl bei der Umsetzung mit einer Metamodellierungsplattform als auch bei der Verwendung von Datenbanken zusätzlichen Aufwand darstellen oder bei beiden Lösungen vorhanden sind. So muss für das Testen des MM-Repositories entweder ein vorhandenes Testframework für ADOxx® angepasst werden oder selbst Tests für das Datenbank-basierte MM-Repository erstellt werden. Für die Umsetzung eines MM-Repositories müssen in beiden Fällen die unterschiedlichen Technologien erlernt und angewendet werden können.

Für ADOxx® ist sowohl spezifisches Wissen zu ADOxx® und JavaScript notwendig als auch Modellierungserfahrungen. Bei der Implementierung mittels Datenbanken sind Structured Query Language (SQL) Kenntnisse notwendig. In beiden Fällen werden Erfahrungen mit Webservern und der Implementierung von Services für die Benutzerinteraktion empfehlenswert. Loggingmechanismen sind in unterschiedlicher Form in beiden Fällen vorhanden. Abfragen

müssen in beiden Fällen umgesetzt werden jedoch auf unterschiedliche Weise. So benötigt man für ADOxx® JavaScript und für Datenbanken SQL.

Eventmanagment ist ebenfalls in beiden Fällen vorhanden. In ADOxx® werden Events beim Ausführung von unterschiedlichen Funktionen, wie dem Erzeugen von Objekten innerhalb des MM-Repositories, ausgelöst. Ein Beobachter (Observer) kann sich beim Eventlistener registrieren und wird danach automatisch benachrichtigt, wenn das gewünschte Event aufgetreten ist. Die Funktionalität kann für das Notificationmanagement verwendet werden. In Datenbanksystemen können ebenfalls bei Funktionsaufrufen eigene Events ausgelöst werden.

Lockingmechanismen sind in beiden Umsetzungsmöglichkeiten vorhanden. Dabei kann ADOxx® auf Objekt- oder Attributebene sperren. In Datenbanken kann durch Transaktionen Locking realisiert werden, welche auf unterschiedlichen Ebenen implementiert sein können. Durch das Transaktionsmanagement werden die ACID-Eigenschaften[13] laut Haerder u. Reuter [1983] gewährleistet. Dies ist sowohl in ADOxx® als auch in Datenbanksystemen vorhanden. Im MM-Repository muss das Transaktionsmanagement sowohl die Objekte innerhalb von ADOxx® als auch zusätzliche Daten berücksichtigen, wodurch zusätzliche Mechanismen implementiert werden müssen.

In dem speziellen Anwendungsfall für die Speicherung und Verwaltung von Modellierungsmethoden und Modellen überwiegen die Vorteile des MM-Repositorykonzepts auf Basis einer Metamodellierungsplattform. Hier ist vor allem wichtig, dass in einer späteren Erweiterung die Kombination der Modellierungsumgebung mit dem MM-Repository einfacher realisiert werden kann als mit einem Datenbank-basiertem Repository. Des Weiteren sind die Bereitstellung des Usermanagements, die lose Kopplung zwischen Metamodell und Objekten, vorgefertigte Lösungen für Publishing sowie die Unterstützung durch die Modellierungsumgebung bei der Repräsentation und Manipulation der Objekte wesentliche Vorteile, die für die Verwendung von ADOxx® sprechen.

[13] (A) Atomarität, (C) Konsistenz, (I) Isolation, (D) Dauerhaftigkeit

# 5 Modellierungsmethoden-Repository: Implementierung

Eine vollständige Implementierung der zusätzlich notwendigen Funktionen und Mechanismen des MM-Repositories im Rahmen dieser Arbeit ist auf Grund des Umfangs nicht möglich. Daher werden einzelne Funktionalitäten des MM-Repositories gesondert betrachtet und im Rahmen dieser Arbeit implementiert. Um einen Überblick über die Möglichkeiten der Implementierung mit Hilfe eines Metamodellierungsansatzes zu bekommen, wurden An- und Abmeldung, ein Versionsmechanismus und das Konfigurationsmanagement ausgewählt und implementiert.

## 5.1 Infrastruktur

Ein Überblick der Infrastruktur des MM-Repositories wird in Abbildung 5.1 dargestellt. Für den Benutzer gibt es unterschiedliche Möglichkeiten auf das MM-Repository zugreifen zu können. Er kann entweder einen Rich-Client oder ein Web-Interface für den Zugriff verwenden. Durch die Verwendung eines Applikationssservers innerhalb des MM-Repositories können diese unterschiedlichen Zugriffsmöglichkeiten einfach durch unterschiedliche Annotationen erstellt werden.

Die gesamten Services, die für den Benutzer zur Verfügung stehen, werden innerhalb des Service Layers von einem GlassFish-Applikationsserviers angeboten. Da Enterprise JavaBeans (EJB) verwendet werden, können unterschiedliche Services wie Webservices mittels Simple Object Access Protocol (SOAP) oder REpresentational State Transfer (REST) diese benutzen. Zusätzlich kann ebenfalls ein Rich-Client implementiert werden, welcher diese EJB verwendet. Da GlassFish Java unterstützt werden alle Interfaces in Java implementiert.

Die Basisfunktionalitäten des MM-Repositories werden innerhalb des Business Layers realisiert. Da in dieser Schicht bereits ADOxx® R3[14] verwendet wird, müssen diese in JavaScript implementiert werden. Die allgemeinen Basisfunktionalitäten wie Speichern, Aktualisieren und Löschen von Objekten und Relationen sind bereits vorhanden und können in weiterer Folge von den entwickelten Services verwendet werden.

[14] Die Erweiterung R3 steht für die dritte Release des Metamodellierungsplattform ADOxx®.

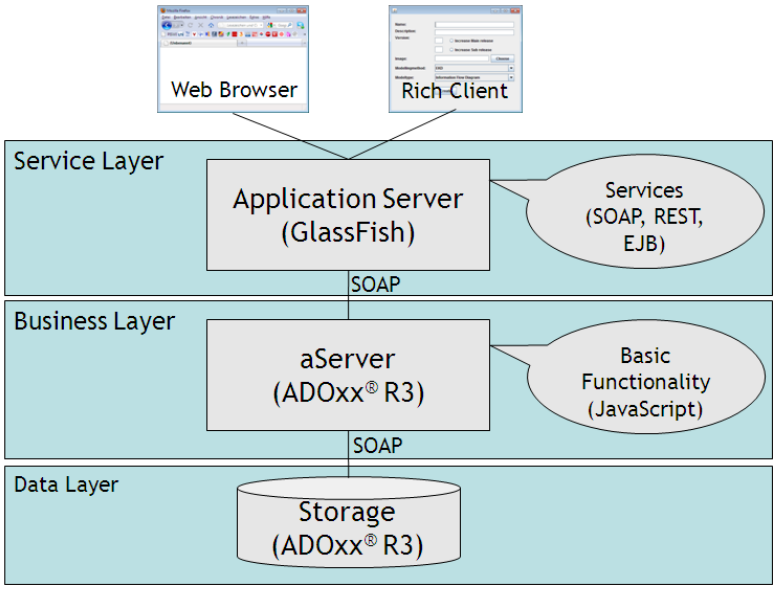



Abbildung 5.1: Infrastruktur

Für die Speicherung der Objekte auf Datenebene wird ebenfalls ADOxx® R3 verwendet. Als Datenmodell wurde das MM-Repository-Metamodell des Konzepts (siehe Kapitel 4.4) in ADOxx® R3 umgesetzt. Dies bedeutet, dass nun alle Klassen aus der Klassenhierarchie (siehe Abbildung 4.2) sowie alle Relationen (siehe Abbildung 4.3) in der Metamodellierungsumgebung vorhanden sind.

## 5.2 An- und Abmeldung vom MM-Repository

Im ersten Schritt der Umsetzung wird von der Benutzerverwaltung die An- und Abmeldung realisiert. Dazu zählen die folgenden Funktionen aus dem Konzept (siehe Kapitel 4.2.2):

- Benutzer anmelden
- Benutzer abmelden

Das Metamodell des MM-Repositories musste für die Umsetzung dieser Funktionalitäten nicht erweitert werden, da es sich dabei um eine Standardfunktionalität von ADOxx® R3 handelt.

Die Benutzerschnittstellen sind in Kapitel 4.3.1 des Konzepts zu finden und beinhalten Interfaces für das An- und Abmelden vom MM-Repository. Beim Anmelden muss der Benutzer Benutzername und Passwort eingeben und anschließend den Button “Login“ klicken. Abbildung 5.2 zeigt das Fenster für die Anmeldung zum MM-Repository. Danach werden die eingegeben Daten

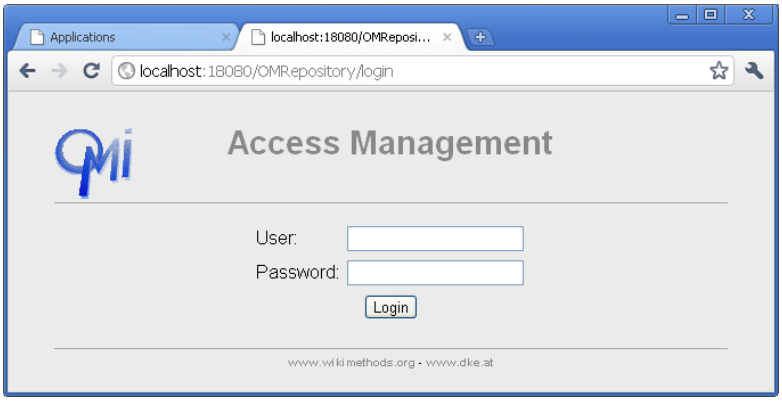



Abbildung 5.2: Screenshot: Login-Seite

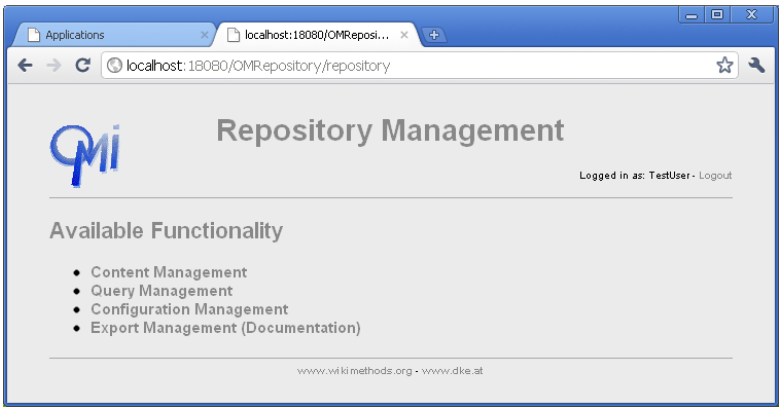



Abbildung 5.3: Screenshot: Überblick über die MM-Repositoryfunktionalitäten mit Logout-Möglichkeit

überprüft und die Anmeldung am aServer von ADOxx® R3 durchgeführt und der Benutzer bekommt eine Liste der möglichen MM-Repository-Funktionalitäten, welche in Abbildung 5.3 abgebildet ist. Der Benutzer ist solange angemeldet bis er den Link “Logout“ klickt, welcher auf jeder Seite des MM-Repository-Managements im rechten oberen Bereich zu finden ist.

Für die Realisierung des Zugriffsmanagements wurden Servlets mit Sessions verwendet, um den Benutzer sowie die Verbindung zu speichern. In jedem Servlet muss daher bei jedem Request überprüft werden, ob es bereits eine Session gibt. Wenn es noch keine Session gibt, muss sich der Benutzer anmelden und nach erfolgreicher Anmeldung wird eine Session angelegt und die Verbindung zum aServer von ADOxx® R3 hergestellt (siehe Listings 5.1, 5.2, 5.3).

```
1 HttpSession s = request.getSession();
2 if (s != null && !s.isNew()) {
3   Object usrobj = s.getAttribute("user");
4   if(usrobj == null) {
```

```
5       return false;
6     }
7     this.user = usrobj.toString();
8     repAccess = (GenericDataAccessService)s.getAttribute("repAccess");
9     return true;
10  }
11  else {
12    response.sendRedirect("/OMRepository/login");
13    return false;
14  }
```

Listing 5.1: Session überprüfen

```
1  try {
2    accessRest.init(user, pass);
3    HttpSession s = request.getSession(true);
4    if (s != null) {
5      s.setAttribute("user", user);
6      s.setAttribute("repAccess", accessRest);
7    }
8    response.sendRedirect("/OMRepository/repository");
9  } catch (RepositoryError e) {
10 }
```

Listing 5.2: Login

```
1  try {
2    BasicServiceProxyFactory bspf = new BasicServiceProxyFactory(user, password,
         serveraddress);
3    creds = bspf.getCreds();
4    rs = bspf.createRepositoryServiceProxy(creds);
5    IServiceProxyFactory pxf = new MultiSessionServiceProxyFactory(serveraddress);
6    aServer = (JsShellServiceProxy) pxf.createJsShellServiceProxy(creds);
7  }
8  catch(Exception e) {
9    throw new RepositoryError("Failed to initialize OM Repository Access!", e);
10 }
```

Listing 5.3: Verbindung zu aServer

Damit das Abmelden vom MM-Repository funktioniert, muss die Session, welche beim Anmelden erzeugt wurde zerstört werden. Des Weiteren muss die Verbindung zum aServer von ADOxx® R3 getrennt werden. Ein Auszug des Codes für das Abmelden ist in Listing 5.4 abgebildet.

```
1  StringBuilder sb = new StringBuilder();
2  HttpSession s = request.getSession();
```

```
3
4  repAccess.invalidate();
5  s.invalidate();
6
7  sb.append(HTMLFormer.getHTMLStart("Access Management", user));
8  sb.append("<div align='center'>" + user + ", you are logged out!!!<br/><br/>")
       ;
9  sb.append("<a href='./login'>Login</a></div>");
10 sb.append(HTMLFormer.getHTMLEnd());
11
12 response.getOutputStream().print(sb.toString());
13 response.setContentType("text/html");
```

Listing 5.4: Abmelden

## 5.3 Versionierungsmechanismus

Für ein funktionierendes Versionsmanagement benötigt das MM-Repository Versionsmechanismen, welche in Kapitel 4.2.6 definiert wurden. Dazu zählt vor allem die Funktion für die Vergabe von Versionsnummern (*getVersionnumber(object, mainrelease, subrelease)*). Dabei wird ein Objekt übergeben. Des Weiteren muss der Benutzer auswählen, ob die Mainrelease- und/oder Subrelease-Nummer erhöht werden soll.

Für die Verwaltung von versionierbaren Objekten musste das Metamodell des MM-Repositories um die Klasse *Versioned Object* erweitert werden, welche Attribute für die Versionierung enthält. Diese Attribute sind: Beschreibung, Kommentar, Erstellungsdatum, Mainrelease-Nummer, Subrelease-Nummer, Branch sowie eine Markierung für obsolete Objekte. Für den nachfolgend implementierten Versionsmechanismus sind vor allem Mainrelease-Nummer und Subrelease-Nummer von Bedeutung. Des Weiteren wurde die Relation *has predecessor* erstellt, welche die Vorgängerversion eines versionierbaren Objekts speichert.

Der Versionsmechanismus wurde als zentrale Basisfunktionalität am aServer von ADOxx® R3 implementiert und wird in Listing 5.5 definiert. Diese Funktion wird in weiterer Folge von einem Servlet des MM-Repositories aufgerufen, um die Subrelease- und Mainreleasennummer automatisch vergeben zu können.

```
1  getVersionnumber: function OMRepAccess_getVersionnumber(aObjid, bMain, bSub) {
2    var aRepInstID = fix2AdoID(aObjid);
3    var aRepInst = cRepo.getRepositoryInstanceWithID(aRepInstID);
4    aRepInst.load(true);
5    var aMain = OMRepAccess.getValueOfAttribute(aRepInst, "MAIN_RELEASE");
```

```
6    var aSub = OMRepAccess.getValueOfAttribute(aRepInst, "SUB_RELEASE");
7
8    if (bMain) {
9      aMain++;
10   }
11   if (bSub){
12     aSub++;
13   }
14   let aEntity  = [];
15   aEntity.push({id: 'ID', optionValue: aRepInst.ID.toString()});
16   aEntity.push({id: 'MAIN_RELEASE', optionValue: aMain});
17   aEntity.push({id: 'SUB_RELEASE', optionValue: aSub});
18
19   return aEntity;
20 }
```

Listing 5.5: Versionsnummer automatisch erhöhren

## 5.4 Konfigurationsmanagement

Funktionen, die as Konfigurationsmanagement zur Verfügung stellt, wurden bereits im Konzept beschrieben (siehe Kapitel 4.2.7). Diese sind:

- Konfiguration erstellen
- Objekte von einer Konfiguration entfernen
- Objekte zu einer Konfiguration hinzufügen
- Konfiguration löschen

Um ein Konfigurationsmanagement innerhalb des MM-Repositories zu realisieren, wurde das Metamodell um die Klasse *Configuration* sowie um die Relation *has components* erweitert. Diese Konzepte wurden bereits im Konzept beschrieben. Da diese Konzepte einen wesentlichen Bestandteil für die Implementierung des Konfigurationsmanagements bilden, werden sie ihr nochmals erwähnt.

Eine *Configuration* fasst unterschiedliche *Versioned Objects* zusammen. Die *Configuration* wird durch einen Namen beschrieben und besteht aus einer Markierung, um feststellen zu können, ob sich Komponenten der *Configuration* geändert haben. Für die Verbindung zwischen *Configuration* und *Versioned Objects* wird eine Relation *has components* benötigt.

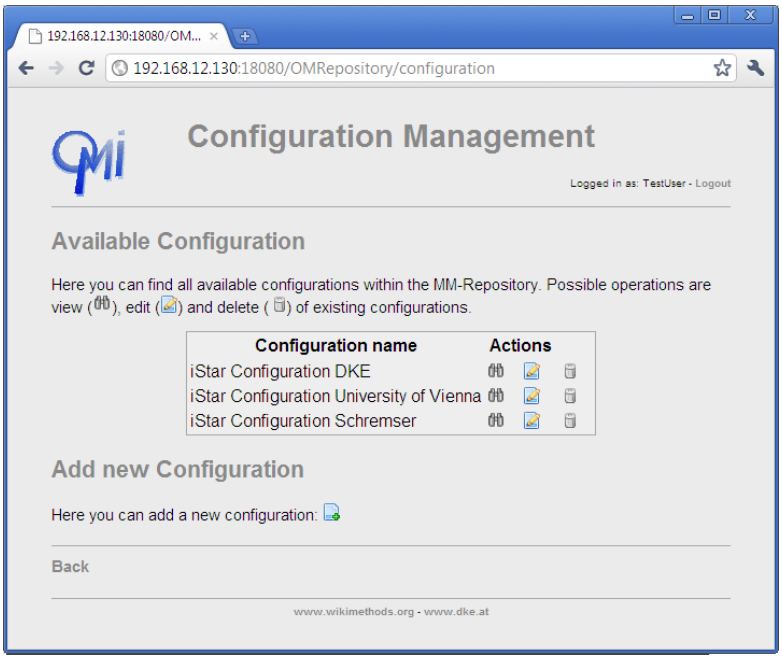



Abbildung 5.4: Screenshot: Übersicht zum Konfigurationsmanagement

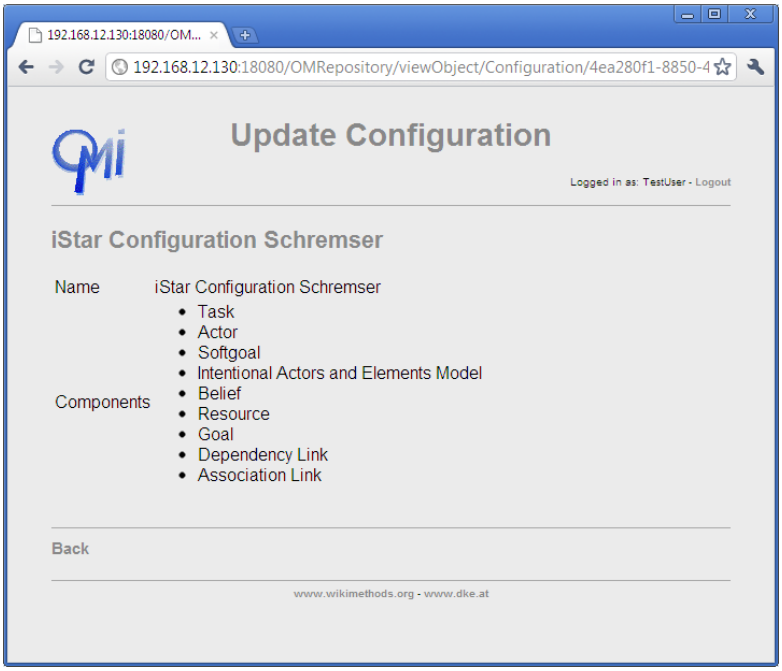



Abbildung 5.5: Screenshot: Konfiguration anzeigen

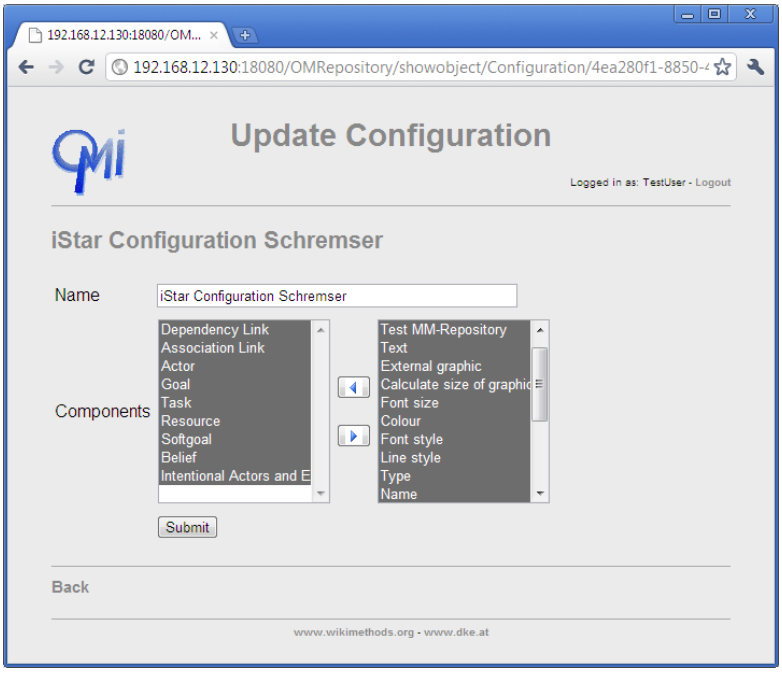



Abbildung 5.6: Screenshot: Konfiguration editieren

Die Benutzerschnittstellen für das Konfigurationsmanagement wurde ebenfalls bereits im Konzept (siehe Kapitel 4.3.4) kurz beschrieben. Dafür werden nun Servlets implementiert, welche die Verwaltung von Konfigurationen unterstützen. Dadurch kann der Benutzer Konfigurationen erstellen, anzeigen, verändern sowie löschen. Das Web-Interface für das Konfigurationsmanagement wird in Abbildung 5.4 dargestellt und beinhaltet eine Liste mit allen verfügbaren Konfigurationen sowie zu jeder Konfiguration drei Buttons für Anzeigen, Editieren und Löschen. Eine vorhandene Konfiguration kann wie in Abbildung 5.5 angezeigt werden. Im unteren Bereich des Interfaces kann der Benutzer eine neue Konfiguration anlegen indem er den dafür vorgesehenen Hinzufügen-Button klickt.

Wie bereits erwähnt können vorhandene Konfigurationen editiert werden, indem der Benutzer den dafür vorgesehenen Button klickt. Danach erscheint ein Fenster, indem die Konfiguration geändert werden kann. Dieses Fenster wird in Abbildung 5.6 angezeigt. Dadurch können sowohl Komponenten zu einer Konfiguration hinzugefügt als auch entfernt werden. Wenn der Benutzer den Button zum Löschen der Konfiguration klickt, wird diese vom MM-Repository entfernt. Durch Klicken des Hinzufügen-Buttons kann der Benutzer eine neue Konfiguration erstellen.

Für die Realisierung der Liste von verfügbaren Konfigurationen müssen alle Objekte vom Typ *Configuration* geladen werden und anschließend in tabellarischer Form mit den möglichen Operationen ausgegeben werden. Diese Tabelle wird durch den Code aus Listing 5.6 erzeugt.

```
1  try {
2    UUIDArray idarray = repAccess.loadIDsOfModelClass(type);
3    if(idarray != null) {
4      UUID[] extrid = idarray.getData();
5      if(extrid != null) {
6      sb.append("<div␣align='center'><table␣class='objectlist'>");
7      sb.append("<tr><th␣min-width='250'>Configuration␣name</th><th␣colspan='2'>
           Actions</th></tr>");
8      for(UUID cid : extrid) {
9        Configuration conf = (Configuration)repAccess.loadModelObject(type.
             newInstance() ,cid);
10       sb.append("<tr><td␣min-width='250'>" + conf.getName() + "</td>");
11       sb.append("<td␣width='30'><a␣href=\"./showobject/Configuration/" + conf.
             getId() + "\"><img␣src=\"./images/edit.png\"/></a></td>");
12       sb.append("<td␣width='30'><a␣href=\"./deleteObject/" + conf.getId() + "
             \"><img␣src=\"./images/delete.png\"/></a></td></tr>");
13       }
14       sb.append("</table></div>");
15     }
16   }
17   else
18     sb.append("<div␣align='center'>No␣configuration␣available!</div>");
19 }
20 catch(Exception e) {
21   sb.append("<h2>Error␣in:␣" + type.getName() + "</h2><p><span␣class=\"error\">"
         + e.getMessage() + "</span></p>");
22   e.printStackTrace();
23 }
```

Listing 5.6: Liste mit verfügbaren Konfigurationen ausgeben

# 6 Anwendungsbeispiele: i*, SDbD, OM-TV

Die Open Models Initiative bietet eine gute Grundlage für die Illustration der unterschiedlichen Interaktionsmöglichkeiten mit dem entwickelten MM-Repository. Daher wird in diesem Kapitel zuerst die Open Models Initiative vorgestellt. Im Anschluss werden drei Anwendungsbeispiele innerhalb der Open Models Initiative beschrieben. Das erste Beispiel soll die Speicherung und Verwaltung von unterschiedlichen Versionen einer Modellierungssprache anhand der i*-Methode widerspiegeln. Beim Anwendungsbeispiel der SDbD-Methode liegt der Fokus auf der Definition des Vorgehensmodells der Modellierungsmethode sowie der Erstellung geeigneter Dokumentation. Das dritte Beispiel konzentriert sich vor allem auf den Zugriff und vorhandenen Webschnittstellen.

## 6.1 Open Models Initiative

Die Open Model Initiative[15] wurde gegründet, um eine wissenschaftliche Community zu schaffen, die sich mit dem Erzeugen, Wartung, Modifikation, Verteilung und Analyse von Modellierungsmethoden und domain-spezifischen Modellen beschäftigt. Die zentrale Vision der Open Model Initiative ist Modellierungsmethoden und deren Anwendung jedem zur Verfügung zu stellen (Karagiannis u. a. [2010]). Die Initiative ist in drei Säulen strukturiert, welche in Abbildung 6.1 dargestellt wird. Diese drei Säulen sind Community, Projekte und Foundations, welche als Basis-Framework die ADOxx® Plattform verwenden.

Die erste Säule der Struktur der Open Models Initiative beschäftigt sich mit dem Business Modell der Open Models Initiative. Aus diesem Grund werden hier vor allem Mission, Struktur, Prozesse und Erlös (Motivation) definiert, aber auch die rechtlichen und technischen Aspekte werden analysiert. Die Mission der Open Models Initiative wurde bereits angesprochen: Modelle für alle. Die Struktur der Initiative ist stark an Open Source Communities angelehnt, welche in Raymond [1998] ausführlich analysiert wurden. Daraus abgeleitet sind für die Open Models Initiative laut Karagiannis u. a. [2010] folgende Eigenschaften wichtig: nahezu selbst-organisiert, zurückhaltende Führung, offene und geteilte Inhalte sowie aktive Benutzer-Teilnahme. Laut

[15] www.wikimethods.org

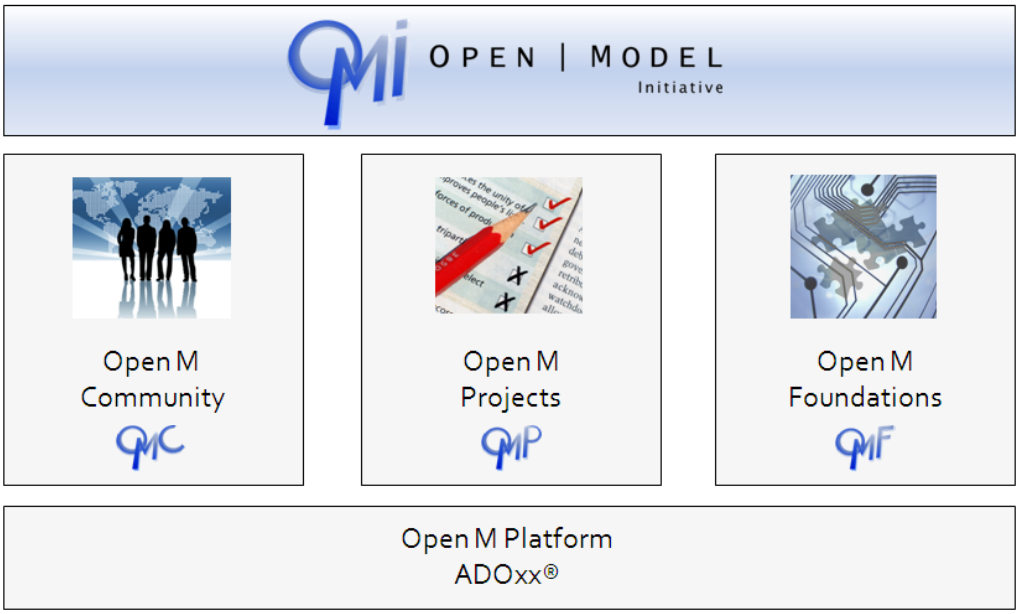



Abbildung 6.1: Struktur der Open Models Initiative (vgl. Karagiannis u. a. [2010])

Karagiannis u. a. [2010] sind drei Hauptprozesse innerhalb der Open Models Community vorhanden. Diese sind Entwicklungsprozess, Genehmigungsprozess und Wartungsprozess, welche wiederum in Subprozesse unterteilt werden können. Zum Entwicklungsprozess zählt (1) die Initialisierung eines Projekts, (2) Durchführung einer Anforderungsanalyse, (3) Auswählen von Modellierungsbereich, Modellierungswerkzeug und Modellierungsmethode sowie (4) Erzeugen der Modellierungsmethode. Der Genehmigungsprozess beinhaltet die Subprozesse (5) Kontrolle der gemeldeten Fehler und Anregungen, (6) Initialisierung des Wartungsprozesses, (7) Aktualisierung des Release-Status und (8) Genehmigung der Modellierungsmethode. (9) Beseitigung von Fehlern und (10) Implementierung von weiteren Vorschlägen sind Subprozesse des Wartungsprozesses. Daher ist die Etablierung der wissenschaftlichen Aspekte in der Metamodellierung das Ziel der Open Models Initiative.

Innerhalb der Open Model Initiative sind unterschiedliche Projekte enthalten, welche in drei unterschiedliche Arten unterteilt werden: Entwicklung von Modellierungsmethoden, Verwendung von Modellierungsmethode und Entwicklung von Metamodellierungswerkzeugen. Diese unterschiedlichen Projekte können nun von unterschiedlichen Mitgliedern der Initiative entwickelt und verwendet werden. Dabei werden Projekte meist innerhalb von unterschiedlichen Communities, wie Software Engineering, Requirements Engineering usw., entwickelt. Beispiele für Projekte

innerhalb der Open Model Initiative sind Referenzimplementierungen zu i*[16] und SDbD[17]. Eine vollständige Liste der Projekte finden Sie unter http://www.openmodels.at/web/omi/omp.

Die dritte Säule der Open Model Initiative beschäftigt sich mit der Bereitstellung von Grundlagen für Modellierungsmethoden sowie Modellierungsumgebungen. Dazu zählen unter anderem auch Konzept und Spezifikation von Modellierungsmethoden und deren Umsetzung innerhalb einer Modellierungsmethode.

Um die Open Models Initiative etablieren zu können, wurde eine Internetplattform entwickelt, um eine Interaktion und Diskussion zwischen den Benutzern sowie den Austausch und die Entwicklung von Modellierungsmethoden und Modellen zu unterstützen. Unter anderem soll die Plattform ein Modell-Repository zur Verfügung stellen, um die unterschiedlichen Modellierungsmethoden und Modelle verwalten zu können. Laut Karagiannis u. a. [2010] soll es möglich sein Modelle im Original-Format hochladen und herunterladen zu können. Zusätzlich soll das Modell als Bild gespeichert werden und falls es in einem nicht frei zugänglichen Tool entwickelt wurde, soll das Modell in ein Format transformiert werden, welches von einem frei zugänglichen Tool interpretiert werden kann. Die Plattform soll außerdem eine Möglichkeit der Darstellung und Suche der Modelle zur Verfügung stellen. Daher müssen zusätzliche Informationen, wie Modellname, kurze textuelle Beschreibung, Informationen über die Modellierungsmethode und Modellierungsumgebung sowie Stichwörter, die das Modell beschreiben, zu den Modellen abgelegt werden.

Anwendungsbeispiele innerhalb der Open Models Initiative für das MM-Repository sind unter anderem die Definition von Modellierungsmethoden, die Verwaltung der Modellierungsmethoden sowie der Zugriff auf die unterschiedlichen Inhalte des MM-Repositories. Aus diesem Grund werden in den nachfolgenden Kapiteln diese Anwendungsbeispiel anhand unterschiedlicher Modellierungsmthoden erläutert.

## 6.2 Speicherung und Verwaltung: Ein i*-Anwendungsfall

Für die Speicherung einer Modellierungssprache wird das Content-Management des MM-Repositories benötigt, um die unterschiedlichen Objekte anlegen zu können. In diesem Anwendungsbeispiel wird die i*-Methode[18] im MM-Repository abgebildet und verwaltet. Als Quelle für die unterschiedlichen Objekte der Modellierungsmethode wurde sowohl Abdulhadi [2007]

[16] http://www.openmodels.at/web/istar/
[17] http://www.openmodels.at/web/sdbd
[18] http://www.openmodels.at/web/istar/

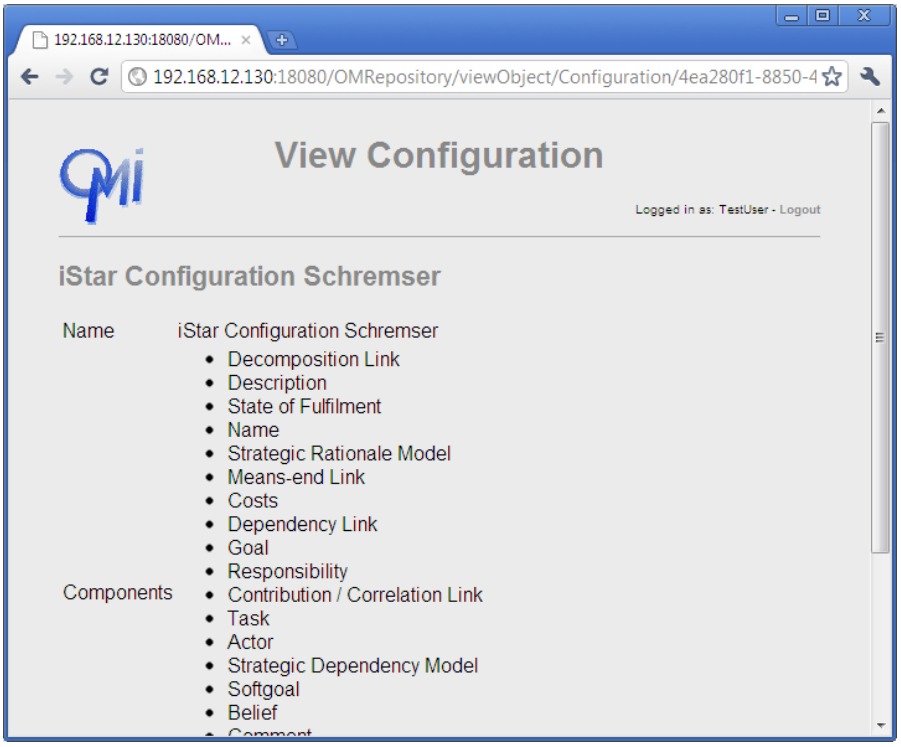



Abbildung 6.2: Screenshot: Auszug aus der i*-Methode

als auch Schwab u. a. [2010] herangezogen. Dabei ist zu beachten, dass es unterschiedliche Interpretationen und daher Versionen der Objekte der Methode gibt.

Eine Interpretation der Modellierungssprache i* besteht aus zwei Modelltypen *Strategic Dependency Model* und *Strategic Rationale Model*. Der Modelltyp *Strategic Dependency Model* besteht wiederum aus diversen Klassen und Relationsklassen. Die benötigten Klassen sind *Actor*, *Goal*, *Task*, *Resource*, *Softgoal* und *Belief*. Zusätzlich werden noch folgende Relationsklassen benötigt: *Dependency Link*, *Association Link*, *Means-end Link*, *Decomposition Link* sowie *Contribution / Correlation Link*. Diese Klassen und Relationsklassen wiederum besitzen diverse Attribute, wie *Name*, *Description*, *Comment*, *Responsibility*, *Costs*, *Priority* oder *State of Fulfilment*. Hier ist jedoch zu beachten, dass diese Auflistung nicht vollständig ist. Für die Speicherung der Notation kann zu jeder Klasse und Relation eine Notation in Form eines Images gespeichert werden. Um die diversen Interpretationen einer Methode abbilden zu können, ist es hier möglich unterschiedliche Notationen zu definieren. Eine dieser Notationen kann in weiterer Folge bei der Erstellung einer Konfiguration ausgewählt werden. In Abbildung 6.2 wird eine Konfiguration mit allen gewünschten Objekten dargestellt, welche als Basis für die Entwicklung einer Interpretation der i*-Methode herangezogen werden kann.

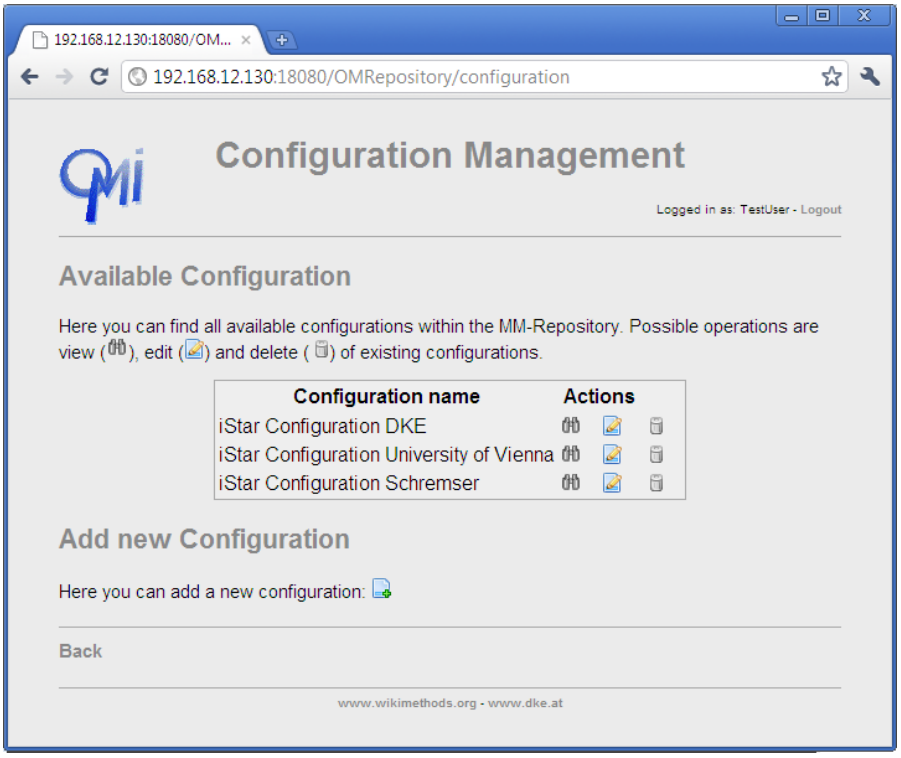



Abbildung 6.3: Screenshot: Diverse Interpretationen der i*-Methode

Wie bereits erwähnt ist es möglich verschiedene Interpretationen derselben Methode zu definieren. Dies kann unter anderem bedeuten, dass eine andere Notation für Klassen und Relationen verwendet wird. Es ist jedoch auch möglich, dass unterschiedliche Interpretationen der Syntax definiert und verwaltet werden. Dabei sind einer Modellierungssprache andere Auslegungen von Klassen, Relationen und Modelltypen zugeordnet. Bei der Erstellung von diversen Konfigurationen muss jedoch darauf geachtet werden, dass die Spezifikation der Modellierungssprache und in weiterer Folge die Spezifikation der Modellierungsmethode eingehalten werden. In Abbildung 6.3 werden unterschiedliche Konfigurationsmöglichkeiten der i*-Methode aufgelistet, die anschließend angezeigt, geändert oder gelöscht werden können.

## 6.3 Speicherung und Dokumentation: Ein SDbD-Anwendungsfall

Da man nicht nur Modellierungssprachen innerhalb des MM-Repositories definieren und verwalten kann, wird anhand der SDbD-Methode[19] das Vorgehen näher betrachtet. Im Methodenhandbuch (Roussopoulos u. Karagiannis [2008]) wird sowohl die Modellierungssprache als auch

[19] http://www.openmodels.at/web/sdbd

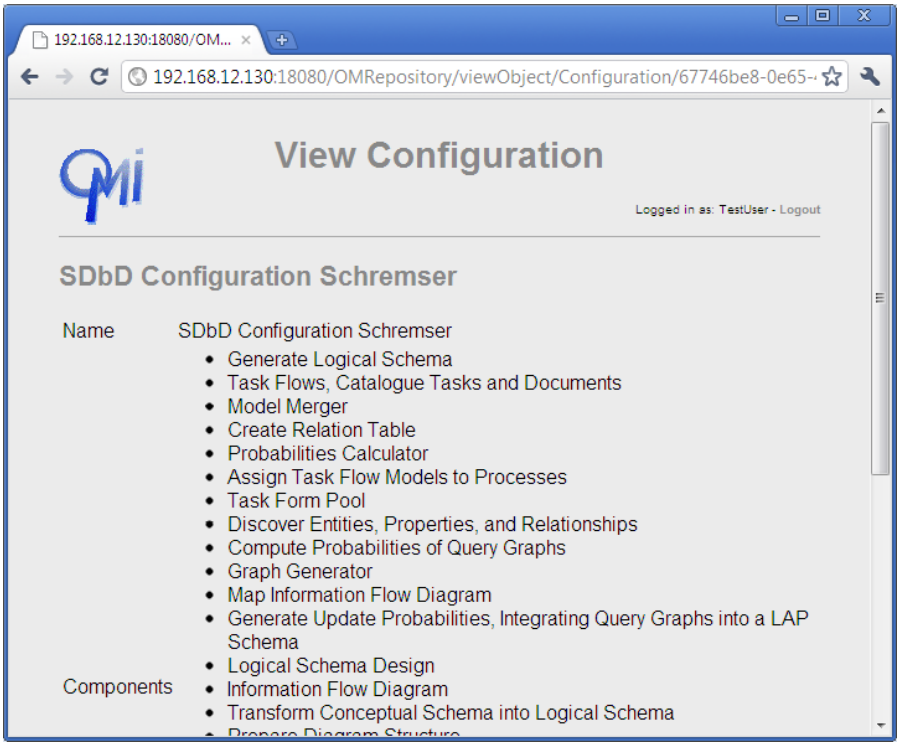



Abbildung 6.4: Screenshot: Vorgehen der SDbD-Methode

das Vorgehen definiert, jedoch wird anschließend auf das Vorgehen der SDbD-Methode genauer eingegangen. Anhand dieses Anwendungsbeispiel soll zuerst das Vorgehen im MM-Repository gespeichert werden. In einem weiteren Schritt soll es möglich sein, das zuvor definierte Vorgehen in einer geeigneten Form exportieren zu können. Dadurch wird den Anwendern der Modellierungsmethode eine Dokumentation der Vorgehensweise geboten, um diesen bei der Anwendung zu unterstützen.

Um das Vorgehen speichern zu können, müssen die Phasen und Schritte der Methode erörtert werden. Des Weiteren ist es wichtig zu definieren, welche Modelltypen und Mechanismen innerhalb der unterschiedlichen Phasen und Schritte vom Benutzer verwendet werden sollen. Das Vorgehen der SDbD-Methode unterteilt sich in die Phasen *Environment Analysis*, *System Analysis and Specification*, *Conceptual Modeling*, *Logical Schema Design* und *Task Emulation & Optimisation*. Diese Phasen unterteilen sich wiederum in Schritte, welche Mechanismen und Modelltypen verwenden. Die Phase *Environment Analysis* unterteilt sich in die Schritte *Prepare Diagram Structure* und *Map Information Flow Diagram*. *Prepare Diagram Structure* und *Map Information Flow Diagram* benötigen den Modelltyp *Information Flow Diagram* und keine zusätzlichen Mechanismen. Die Phase *System Analysis and Specification* besteht aus den drei Schritten *Create Models*, *Map Task Flows, Catalogue Tasks and Documents* und

*Assign Task Flow Models to Processes*, welche die Modelltypen *Task Form Pool*, *Document Form Pool* und *Task Flow Diagram* benötigen. Die Phase *Conceptual Modeling* besteht aus den Schritten *Discover Entities, Properties, and Relationships* und *Express EPRs in a Multiple-Level Language*. Diese Schritte benötigen keine zusätzlichen Modelltypen jedoch den Mechanismus *Create Relation Table*. *Logical Schema Design* beinhaltet die Schritte *Choose E-R Diagram* und *Transform Conceptual Schema into Logical Schema*, wofür der Mechanismus *Generate Logical Schema* benötigt wird. Die letzte Phase *Task Emulation & Optimisation* steht aus den Schritten *Generate the Data Input Requirements of Each Task*, *Compute Probabilities of Query Graphs*, *Generate Update Probabilities*, *Integrating Query Graphs into a LAP Schema* und*Optimising the LAP Schema*. Verwendete Mechanismen sind *Graph Generator*, *Probabilities Calculator* und *Model Merger*. In Abbildung 6.4 wird das gesamte Vorgehen der SDbD-Methode zu einer Konfiguration zusammengefasst.

Nach der Speicherung des Vorgehens soll die Dokumentation erstellt werden. Dabei kann der Benutzer zwischen den Formaten HTML, DOC und PDF wählen. Die Dokumentation enthält einen Überblick über die unterschiedlichen Phasen. Danach kann eine Detailansicht der Phasen mit den unterschiedlichen Schritten dargestellt werden. Die Schritte wiederum beinhalten die verwendeten Modelltypen und Mechanismen. Ein Beispiel dieser Dokumentation kann aus dem SDbD-Methodenhandbuch (Roussopoulos u. Karagiannis [2008]) entnommen werden.

## 6.4 Zugriff auf das MM-Repository: OM-TV

OM-TV[20] ist ein weiteres Anwendungsbeispiel innerhalb der Open Models Initiative für die Verwendung des MM-Repositories. Dabei liegt der Fokus auf der Verwendung der vordefinierten Schnittstellen zum Zugriff auf die Inhalte des MM-Repositories.

Ziel von OM-TV ist das Verbreiten von Informationen über die Open Models Initiative auf synchrone Weise. Dabei steht vor allem die Präsentation der Projekte, deren Mitglieder und die Aktivitäten innerhalb des Projekts im Vordergrund, wodurch der internationale Charakter der Open Models Initiative gestärkt werden soll. OM-TV bietet sowohl einen allgemeinen Kanal mit allen Projekten und allgemeinen Informationen zu den Projekten als auch spezifische Projekt-Kanäle für die unterschiedlichen Projekte mit projektspezifischen Informationen. Durch die Inhalte der Projekt-Kanäle können vom Projektverantwortlichen festgelegt werden, wodurch OM-TV besser auf die Anforderungen und Bedürfnisse der Benutzer eingehen kann. Des Weiteren soll OM-TV auch für mobile Geräte zum Einsatz kommen. Dabei werden dieselben

[20] www.wikimethods.org

Informationen gesendet, jedoch ist die Ausgabe speziell auf das kleinere Format der mobilen Geräte angepasst.

Für diesen Anwendungsfall wurden zum MM-Repository die Klassen *OM User*, *OM Community*, *OM Project* und *Location* sowie die Relationsklassen *has location*, *has user*, *has community* und *has modellingmethod* hinzugefügt. Dadurch kann OM-TV alle notwendigen Daten durch geeignete Schnittstellen aus dem MM-Repository auslesen, die Daten für eine Videoausgabe aufbereiten und diverse Auswertungen durchführen. OM-TV gibt zum Beispiel zu jedem Projekt die Mitglieder in sequentieller Reihe mit deren Profilfoto und Heimatadresse aus. Um einen Überblick über die unterschiedlichen Standorte eines Projekts zu bekommen, werden alle Mitglieder mit Heimatadresse auf einer Landkarte präsentiert. Weitere Inhalte für die Ausgabe über OM-TV sind unter anderem Videos, Images, Auswertungen über die Modellierungsmethode sowie Anwendungsbeispiele der jeweiligen Modellierungsmethode. Diese Aufzählung ist jedoch nur ein Auszug aus den Möglichkeiten, die sich durch die Verwendung des MM-Repositories bieten.

# 7 Zusammenfassung und Ausblick

Durch die immer größer werdende Anzahl an Modellierungsmethoden und deren Anwendungsbereiche innerhalb der Open Models Initiative stieg die Forderung nach einem Repository zur Verwaltung dieser Modellierungsmethoden. Anhand von vorhandenen Repositorykonzepten wurden die Anforderungen und Funktionalitäten analysiert und für diesen spezifischen Anwendungsbereich adaptiert. Ein wichtiges Merkmal dieses Konzepts im Gegensatz zu üblichen in der Literatur vorhandenen Repositorykonzepten ist die Verwendung von Metamodellierungsansätzen anstelle von Datenbanktechnologien. Dadurch mussten im ersten Schritt theoretische Grundlagen sowohl zu Metamodellierungsansätzen sowie Repositorykonzepten erörtert werden, bevor mit der Entwicklung der Konzeption und Implementierung begonnen werden konnte.

Aus der Analyse der vorhandenen Repositorykonzepte sind vor allem die unterschiedlichen Funktionalitäten sowie die verschiedenen Möglichkeiten der Inhalte von Bedeutung, da diese die Grundlage für die Konzeption dieses Modellierungsmethoden-Repositories (MM-Repositories) legen. Die Funktionalitäten aus vorhanden Repositorykonzepten wurden erörtert und falls diese für das MM-Repository in Frage kamen in das Konzept aufgenommen. Diese Funktionalitäten wurden jedoch auch um spezifische Funktionalitäten des MM-Repositories erweitert. Darauf aufbauend wurden die möglichen externen Schnittstellen festgelegt, welche vom MM-Repository angeboten werden.

Im nächsten Schritt wurde die Erhebung der notwendigen Inhalte durchgeführt. Daher wurden die relevanten Komponenten einer Modellierungsmethode analysiert sowie die Anforderungen seitens der Open Models Initiative erhoben. In einem weiteren Schritt wurden diese Komponenten miteinander kombiniert und um Konzepte für die Speicherung von Metadaten und zusätzlichen Ressourcen wie Bilder oder Dokumente erweitert. Durch die Identifikation dieser benötigten Inhalte konnte eine Architektur des MM-Repositories definiert werden.

Im Anschluss an die Konzeption wurden einige MM-Repositoryfunktionalitäten und Web-Interfaces innerhalb der Open Models Initiative umgesetzt. Dabei lag der Fokus bei der Auswahl der Funktionalitäten auf Basisoperationen, die für den Betrieb des MM-Repositories zumindest notwendig sind.

Das Ergebnis dieser Arbeit ist daher sowohl die Konzeption des MM-Repositories sowie eine prototypische Implementierung ausgewählter Funktionalitäten und Web-Interfaces des Konzepts.

Die Arbeit bietet somit ein hohes Ausbaupotenzial, da das entwickelte Konzept als Spezifikationsgrundlage für eine weiterführende Implementierung des gesamten MM-Repositories verwendet wird. Durch die prototypische Umsetzung können zahlreiche im Konzept spezifizierte Funktionalitäten wie Such- und Abfragemöglichkeiten umgesetzt werden.

Jedoch bieten nicht nur die festgelegten Funktionalitäten Ausbaumöglichkeiten, sondern es können auch zusätzliche Funktionalitäten integriert werden. So können Exportmöglichkeiten für die im MM-Repository definierten Modellierungsmethoden geboten werden, wodurch Dokumentationen automatisiert erstellt werden können. Eine weitere Erweiterungsmöglichkeit ist die Integration des MM-Repositories mit einer Modellierungsumgebung, um die spezifizierten Modellierungsmethoden anwenden zu können.

Aus dieser kleinen Auswahl an Erweiterungsmöglichkeiten ist bereits ersichtlich, dass diese Arbeit eine gute Ausgangsbasis für weiterführende Implementierungen eines MM-Repositories auf Basis eines Metamodellierungsansatzes schafft.

# Literaturverzeichnis


[Abdulhadi 2007] Abdulhadi, Samer: *i* Guide Version 3.0.* `http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=67`. Version: 2007. – Zugriff: 2011-01-19 (zitiert auf der Seite 91).

[Alkhatib u. Scholl 2008] Alkhatib, Ramez ; Scholl, Marc H.: Efficient Compression and Querying of XML Repositories. In: *19th International Workshop on Database and Expert Systems Applications (DEXA 2008), 1-5 September 2008, Turin, Italy*, 2008, S. 365–369 (zitiert auf der Seite 12).

[Apps u. Macintyre 2000] Apps, Ann ; Macintyre, Ross: Dublin Core Metadata for Electronic Journals. 2000, S. 93+ (zitiert auf der Seite 14).

[Balzert 1998] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung.* Spektrum Akademischer Verlag, 1998 (zitiert auf den Seiten xx, 19 und 20).

[Baralis u. Fiori 2008] Baralis, Elena ; Fiori, Alessandro: Exploring Heterogeneous Biological Data Sources. In: *19th International Workshop on Database and Expert Systems Applications (DEXA 2008), 1-5 September 2008, Turin, Italy*, 2008, S. 647–651 (zitiert auf der Seite 13).

[Beer u. a. 2009] Beer, Christopher A. ; Pinch, Peter D. ; Cariani, Karen: Developing a flexible content model for media repositories: a case study. In: *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries.* New York, NY, USA : ACM, 2009 (JCDL '09), S. 97–100 (zitiert auf der Seite 12).

[Bernstein u. Rahm 2000] Bernstein, Philip ; Rahm, Erhard: Data Warehouse Scenarios for Model Management. In: *In International Conference on Conceptual Modeling*, Springer, 2000, S. 1–15 (zitiert auf der Seite 7).

[Bernstein 2003] Bernstein, Philip A.: Applying Model Management to Classical Meta Data Problems, 2003 (zitiert auf der Seite 7).

[Bernstein u. Dayal 1994] Bernstein, Philip A. ; Dayal, Umeshwar: An Overview of Repository Technology. In: *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, 1994 (zitiert auf den Seiten 8 und 9).

[Bernstein u. Haas 2008] Bernstein, Philip A. ; Haas, Laura M.: Information integration in the enterprise. In: *Commun. ACM* 51 (2008), Nr. 9, S. 72–79 (zitiert auf der Seite 8).

[Bernstein u. a. 2000a] Bernstein, Philip A. ; Levy, Alon Y. ; Pottinger, Rachel A.: A vision for management of complex models. In: *SIGMOD Record* 29 (2000), Nr. 4 (zitiert auf der Seite 7).

[Bernstein u. a. 2000b] Bernstein, Philip A. ; Levy, Alon Y. ; Pottinger, Rachel A.: A vision for management of complex models / Microsoft Research. http://research.microsoft.com/pubs/69782/tr-2000-53.pdf, 2000. – Forschungsbericht. – Zugriff: 2011-01-19 (zitiert auf den Seiten 7 und 8).

[Bézivin 2001] Bézivin, Jean: Towards a precise definition of the OMG/MDA framework. In: *Proceedings of ASE 01*, IEEE Computer Society Press, 2001, S. 273–280 (zitiert auf der Seite 6).

[Blanning 1982] Blanning, Robert W.: Data management and model management: a relational synthesis. In: *ACM-SE 20: Proceedings of the 20th annual Southeast regional conference*, 1982, S. 139–147 (zitiert auf der Seite 7).

[Calvanese u. a. 2009] Calvanese, Diego ; Giacomo, Giuseppe ; Lembo, Domenico ; Lenzerini, Maurizio ; Rosati, Riccardo: Conceptual Modeling for Data Integration. (2009), S. 173–197 (zitiert auf der Seite 8).

[Dublin Core Metadata Initiative 2010] Dublin Core Metadata Initiative: Dublin Core Metadata Element Set, Version 1.1 / DCMI. Version: 2010. `http://dublincore.org/documents/2010/10/11/dces/`. 2010. – Forschungsbericht. – Zugriff: 2011-01-19 (zitiert auf der Seite 24).

[Elam u. a. 1980] Elam, Joyce J. ; Henderson, John C. ; Miller, Louis W.: Model Management Systems: An Approach to Decision Support in Complex Organizations. In: *Proceedings of the First International Conference on Information Systems.* Philadelphia, USA, 1980, S. 98–110 (zitiert auf der Seite 7).

[Elmasri u. Navathe 2009] Elmasri, Ramez A. ; Navathe, Shamkant B.: *Grundlagen von Datenbanksystemen. Bachelorausgabe.* Pearson Studium, 2009 (zitiert auf der Seite 17).

[Engles 1970] Engles, R. W.: A Tutorial on Data Base Organization / IBM. Poughkeepsie, N.Y., 1970. – IBM Technical Report TR 00.2004 (zitiert auf der Seite 16).

[Fagin u. a. 2009] Fagin, Ronald ; Haas, Laura M. ; Hernández, Mauricio ; Miller, Renée J. ; Popa, Lucian ; Velegrakis, Yannis: Clio: Schema Mapping Creation and Data Exchange. (2009), S. 198–236 (zitiert auf der Seite 8).

[Freed u. Borenstein 1996] Freed, N. ; Borenstein, N.: *RFC 2046 Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. `http://tools.ietf.org/html/rfc2046`. Version: 1996. – Zugriff: 2011-01-19 (zitiert auf den Seiten 24 und 25).

[Haas u. a. 2009] Haas, Laura M. ; Hentschel, Martin ; Kossmann, Donald ; Miller, Renée J.: Schema AND Data: A Holistic Approach to Mapping, Resolution and Fusion in Information Integration. In: *Conceptual Modeling - ER 2009, 28th International Conference on Conceptual Modeling* Bd. 5829, Springer, 2009 (Lecture Notes in Computer Science), S. 27–40 (zitiert auf der Seite 8).

[Haerder u. Reuter 1983] Haerder, Theo ; Reuter, Andreas: Principles of transaction-oriented database recovery. In: *Journal ACM Computing Surveys (CSUR)* 15 (1983), December, S. 287–317 (zitiert auf den Seiten 37 und 78).

[Härder u. Bühmann 2004] Härder, Theo ; Bühmann, Andreas: Datenbank-Caching - Eine systematische Analyse möglicher Verfahren. In: *Informatik - Forschung und Entwicklung* 19 (2004), July, Nr. 1, S. 2–16 (zitiert auf der Seite 77).

[Haslhofer u. Klas 2010] Haslhofer, Bernhard ; Klas, Wolfgang: A survey of techniques for achieving metadata interoperability. In: *ACM Computer Surveys* 42 (2010), March, S. 7:1–7:37 (zitiert auf der Seite 15).

[Hodgins u. Duval 2002] Hodgins, Wayne ; Duval, Erik: Draft Standard for Learning Object Metadata / Learning Technology Standards Committee, IEEE. 2002. – Forschungsbericht (zitiert auf der Seite 15).

[Holmes u. a. 2010] Holmes, Ta'id ; Zdun, Uwe ; Daniel, Florian ; Dustdar, Schahram: Monitoring and Analyzing Service-Based Internet Systems through a Model-Aware Service Environment. In: *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering, CAiSE 2010*, Springer-Verlag, 2010, S. 98–112 (zitiert auf der Seite 12).

[Ignat u. Norrie 2006] Ignat, Claudia-Lavinia ; Norrie, Moira C.: Advanced Information Systems Engineering, 18th International Conference, CAiSE 2006, Luxembourg, Luxembourg, June 5-9, 2006, Proceedings. In: *CAiSE*, 2006, S. 190–204 (zitiert auf der Seite 12).

[Incorporated 2005] Incorporated, Adobe S.: XMP Specification / Adobe Systems Incorporated. 2005. – Forschungsbericht (zitiert auf der Seite 15).

[ISO/IEC 2001] ISO/IEC: *ISO/IEC 9126. Software engineering – Product quality.* ISO/IEC, 2001 (zitiert auf den Seiten xx, 19 und 20).

[Jacobs u. a. 2009] Jacobs, D ; Kotze, P ; Van Der Merwe, A: Towards an enterprise repository framework. In: *1st International Workshop on Advanced Enterprise Repositories (AER 2009) in conjunction with the 11th International Conference on Enterprise Information Systems (ICEIS 2009)* (2009) (zitiert auf der Seite 8).

[Karagiannis 2007] Karagiannis, Dimitris: Metamodeling as a Concept. In: *Current Trends in Informatics*, New Technologies Publications, 2007 (zitiert auf der Seite 7).

[Karagiannis u. a. 2010] Karagiannis, Dimitris ; Grossmann, Wilfried ; Höfferer, Peter: *Open model initiative - a feasibility study.* `http://cms.dke.univie.ac.at/uploads/media/Open_Models_Feasibility_Study_SEPT_2008.pdf`. Version: November 2010. – Zugriff: 2011-01-19 (zitiert auf den Seiten xx, 19, 24, 89, 90 und 91).

[Karagiannis u. Höfferer 2006] Karagiannis, Dimitris ; Höfferer, Peter: Metamodels in action: An overview. In: Filipe, Joaquim (Hrsg.) ; Shishkov, Boris (Hrsg.) ; Helfert, Markus (Hrsg.): *ICSOFT 2006, First International Conference on Software and Data Technologies*. Setúbal, Portugal, 2006 (zitiert auf der Seite 7).

[Karagiannis u. Kühn 2002] Karagiannis, Dimitris ; Kühn, Harald: Metamodelling Platforms. In: *EC-WEB '02: Proceedings of the Third International Conference on E-Commerce and Web Technologies*. London, UK : Springer-Verlag, 2002. – ISBN 3540441379, S. 182 (zitiert auf den Seiten xx, 5, 6 und 24).

[Kemper u. Eickler 2009] Kemper, Alfons ; Eickler, André: *Datenbanksysteme*. 7., aktualisierte u. erweiterte Auflage. Oldenbourg Verlag, 2009 (zitiert auf der Seite 17).

[Kensche u. a. 2007a] Kensche, David ; Quix, Christoph ; 0002, Xiang L. ; Li, Yong: GeRoMeSuite: A System for Holistic Generic Model Management. In: *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, 2007, S. 1322–1325 (zitiert auf der Seite 8).

[Kensche u. a. 2009] Kensche, David ; Quix, Christoph ; 0002, Xiang L. ; Li, Yong ; Jarke, Matthias: Generic schema mappings for composition and query answering. In: *Data Knowledge Engineering* 68 (2009), Nr. 7, S. 599–621 (zitiert auf der Seite 8).

[Kensche u. a. 2007b] Kensche, David ; Quix, Christoph ; Chatti, Mohamed A. ; Jarke, Matthias: GeRoMe: a generic role based metamodel for model management. (2007), S. 82–117 (zitiert auf der Seite 8).

[Kiefer u. a. 2009] Kiefer, Florian ; Arnold, Konstantin ; Kuenzli, Michael ; Bordoli, Lorenza ; Schwede, Torsten: The SWISS-MODEL Repository and associated resources.

In: *Nucleic Acids Research* 37 (2009), January, Nr. suppl 1, S. D387–D392 (zitiert auf der Seite 12).

[La Rosa u. a. 2009] La Rosa, Marcello ; Reijers, Hajo A. ; Aalst, Wil M.P. van d. ; Dijkman, Remco M. ; Mendling, Jan ; Dumas, Marlon ; Garcia-Banuelos, Luciano: APROMORE : An Advanced Process Model Repository / QUT Faculties and Divisions > Faculty of Science and Technology. 2009. – Forschungsbericht (zitiert auf der Seite 12).

[Mairiza u. a. 2010] Mairiza, Dewi ; Zowghi, Didar ; Nurmuliani, Nurie: An investigation into the notion of non-functional requirements. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA : ACM, 2010 (SAC '10), S. 311–317 (zitiert auf den Seiten xx, 19 und 20).

[Meier 2010] Meier, Andreas: *Relationale und postrelationale Datenbanken*. 7., überarbeitete Auflage. Springer-Verlag Berlin Heidelberg, 2010 (zitiert auf der Seite 17).

[Melnik 2004] Melnik, Sergey: *Lecture Notes in Computer Science*. Bd. 2967: *Generic Model Management: Concepts and Algorithms*. Springer, 2004. – ISBN 3–540–21980–3 (zitiert auf den Seiten 7 und 8).

[Melnik u. a. 2003] Melnik, Sergey ; Rahm, Erhard ; Bernstein, Philip A.: Rondo: a programming platform for generic model management. In: *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 2003, S. 193–204 (zitiert auf der Seite 8).

[Millard u. a. ] Millard, David E. ; Howard, Yvonne ; Watson, Julie ; Arrebola, Miguel: *Towards an Open Repository of Teaching Resources*. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.155.3932`. – Zugriff: 2011-01-19 (zitiert auf der Seite 12).

[Miller u. a. 2001] Miller, Renée J. ; Hernández, Mauricio A. ; Haas, Laura M. ; Yan, Lingling ; Howard Ho, C. T. ; Fagin, Ronald ; Popa, Lucian: The Clio project: managing heterogeneity. In: *SIGMOD Rec.* 30 (2001), Nr. 1, S. 78–83 (zitiert auf der Seite 8).

[Mylopoulos u. a. 1992] Mylopoulos, John ; Chung, Lawrence ; Nixon, Brian: Representing and Using Non-Functional Requirements: A Process-Oriented Approach. In: *IEEE Transactions on Software Engineering* 18 (1992), S. 483–497 (zitiert auf den Seiten 19 und 20).

[Mylopoulos u. a. 1999] Mylopoulos, John ; Chung, Lawrence ; Yu, Eric: From object-oriented to goal-oriented requirements analysis. In: *Commun. ACM* 42 (1999), Nr. 1, S. 31–37 (zitiert auf den Seiten xx, 19 und 20).

[Palanisamy u. a. 2010] Palanisamy, Giriprakash ; Devarakonda, Ranjeet ; Green, Jim ; Wilson, Bruce: Enabling Data Discovery through Virtual Internet Repositories. (2010), Oct. `http://arxiv.org/abs/1010.2440`. – Zugriff: 2011-01-19 (zitiert auf der Seite 13).

[Park u. Childress 2009] Park, Jungran ; Childress, Eric: Dublin Core metadata semantics: an analysis of the perspectives of information professionals. In: *Journal of Information Science* 35 (2009), December, Nr. 6, S. 727–739 (zitiert auf der Seite 14).

[Patashnik 2010] Patashnik, Oren: *BibTeX ing.* `http://newton.ex.ac.uk/tex/pack/bibtex/btxdoc/btxdoc.html`. Version: November 2010. – Zugriff: 2011-01-19 (zitiert auf den Seiten 14 und 15).

[Pottinger u. Bernstein 2008] Pottinger, Rachel ; Bernstein, Philip A.: Schema merging and mapping creation for relational sources. In: *Proceedings of the 11th International Conference on Extending Database Technology (EDBT)*, 2008, S. 73–84 (zitiert auf der Seite 8).

[Raymond 1998] Raymond, Eric S.: *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* Sebastopol, CA, USA : OReilly & Associates, Inc., 1998 (zitiert auf der Seite 89).

[Roussopoulos u. Karagiannis 2008] Roussopoulos, Nicholas ; Karagiannis, Dimitris: *Semantic Database Design Method Handbook.* `http://www.openmodels.at/c/document_library/get_file?uuid=c741f188-a1a3-4f63-a12e-dd6f315ded2a&groupId=12118`. Version: 2008. – Zugriff: 2011-01-19 (zitiert auf den Seiten 93 und 95).

[Saake u. a. 2008] Saake, Gunter ; Sattler, Kai-Uwe ; Heuer, Andreas: *Datenbanken: Konzepte und Sprachen.* 3., aktualisierte und erweiterte Auflage. mitp-Verlag, Redline GmbH, 2008 (zitiert auf den Seiten xx, 16 und 17).

[Schlageter u. Stucky 1983] Schlageter, Gunter ; Stucky, Wolffried: *Datenbanksysteme: Konzepte und Modelle.* Teubner, 1983 (zitiert auf der Seite 37).

[Schwab u. a. 2010] Schwab, Margit ; Karagiannis, Dimitris ; Bergmayr, Alexander: i* on ADOxx®: A Case Study. In: *Fourth International i* Workshop (iStar'10) at CAiSE'10, 22nd International Conference on Advanced Information Systems Engineering*, 2010, S. 92–97 (zitiert auf der Seite 92).

[de Sompel u. a. 2005] Sompel, Herbert V. ; Bekaert, Jeroen ; 0005, Xiaoming L. ; Balakireva, Lyudmila ; Schwander, Thorsten: aDORe: A Modular, Standards-Based Digital Object Repository. In: *Computer Journal* 48 (2005), Nr. 5, S. 514–535 (zitiert auf der Seite 13).

[Stachowiak 1973] Stachowiak, Herbert: Allgemeine Modelltheorie. (1973) (zitiert auf der Seite 5).

[Steiner 2006] Steiner, Rene: *Relationale Datenbanken, Grundkurs.* 6., überarbeitete und erweiterte Auflage. Vieweg Verlag, 2006 (zitiert auf der Seite 17).

[Weber u. Reichert 2008] Weber, Barbara ; Reichert, Manfred: Refactoring Process Models in Large Process Repositories. In: *Proceedings of the 20th international conference on Advanced Information Systems Engineering, CAiSE 2008.* Berlin, Heidelberg : Springer-Verlag, 2008, S. 124–139 (zitiert auf der Seite 12).

[Yan u. a. 2009] Yan, Zhiqiang ; Dijkman, Remco ; Grefen, Paul: *Business Process Model Repositories - Framework and Survey.* `http://alexandria.tue.nl/openaccess/Metis232409.pdf`. Version: Oktober 2009. – Zugriff: 2011-01-19 (zitiert auf der Seite 9).

[Zivkovic u. a. 2009] Zivkovic, Srdjan ; Kühn, Harald ; Murzek, Marion: An Architecture of Ontology-aware Metamodelling Platforms for Advanced Enterprise Repositories. In: *Proceedings of the 1st International Workshop on Advanced Enterprise Repositories (AER 2009), Collocated with 11th International Conference on Enterprise Information Systems (ICEIS 2009).* Milano, Italy, 2009, S. 95–104 (zitiert auf der Seite 8).

# Lebenslauf

## zu meiner Person



| | |
|---|---|
| **persönliche Daten** | Daniela Schremser<br>geboren am 23. Jänner 1986<br>in Neunkirchen<br>ledig |
| **Anschrift (Kontakt)** | Bürgerwiesenstraße 49<br>2640 Gloggnitz<br>Tel.: 0660 / 494 17 23<br>E-Mail: daniela.schremser@gmx.at |
| **Staatsbürgerschaft** | Österreich |

## Ausbildung

| | | |
|---|---|---|
| **Schulbildung** | 1992 – 1996 | Volksschule Gloggnitz |
| | 1996 – 2000 | Gymnasium Sachsenbrunn in Kirchberg |
| | 2000 – 2005 | HTBLuVA Wiener Neustadt, Abteilung für Elektronische Datenverarbeitung und Organisation<br>**Abschluss mit ausgezeichnetem Erfolg** |
| **Akademische Ausbildung** | 2005 – 2008 | Technische Universität Wien<br>Studienrichtung: Wirtschaftsinformatik<br>**Abschluss: Bachelor of Science (BSc)** |
| | seit 2008 | Universität Wien<br>Studienrichtung: Wirtschaftsinformatik |

## Berufserfahrung

| | | |
|---|---|---|
| **Praktika** | 2001 – 2006 | diverse Praktika im Bereich EDV bei<br>**Huyck Austria GmbH**, Abteilung EDV, in Gloggnitz<br>**Sparkasse Neunkirchen**, Abteilung EDV |
| **feste Beschäftigungen** | 2006 – 2007 | **Emig Datenverarbeitung GmbH**<br>Tätigkeit: Programmierung, Datenkonvertierung |
| | 2007 - 2010 | **Universität Wien**<br>Tätigkeit: Studienassistentin |
| | 2010 | **Universität Wien**<br>Tätigkeit: Projektmitarbeiterin (INNOTRAIN IT)<br>INNOvative TRAINing IT Cental Europe |

## Publikationen

**Moodlemoot 2009** in Bamberg

Michael Tesar, Daniela Schremser: Kombination von Moodle und browserbasierter Software

Michael Tesar, Daniela Schremser: Open Source Software zur Erstellung von Lehrmaterialien

Michael Tesar, Daniela Schremser: Yaplaf++ und Moodle: Ein Plagiatsfinder im Praxiseinsatz

**International Journal of Knowledge Management (IJKM)**

Hans-Georg Fill, Daniela Schremser, Dimitris Karagiannis: Generic Approach for the Semantic Annotation of Conceptual Models using a Service-oriented Architecture

## Konferenzen/Workshops

**Österreichischer IT- & Beratertag 2006** in Wien

Teilnahme

**8. Internationale Tagung Wirtschaftsinformatik 2007** (WI2007) in Karlsruhe

Teilnahme

**Multikonferenz Wirtschaftsinformatik 2008** (MKWI2008) in München

Teilnahme

**Europäisches Forum Alpbach 2008** in Alpbach/Tirol

Teilnahme

**9. International Tagung Wirtschaftsinformatik 2009** (WI2009) in Wien

Mitorganisation

**Moodlemoot 2009** in Bamberg

Teilnahme mit akzeptierte Beiträge

**Knowledge Science, Engineering and Management 2009** (KSEM2009) in Wien

Mitorganisation

**Multikonferenz Wirtschaftsinformatik 2010** (MKWI2010) in Göttingen

Teilnahme

## Sonstige Kenntnisse und Interessen

| | |
|---|---|
| **Sprachen** | Deutsch – Muttersprache |
| | Englisch – Gute Kenntnisse |
| **Führerschein** | Fahrzeugklasse B |