Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Arazm Hosieny

Konzeption und Realisierung eines automatisierten Java-Codegenerators basierend auf BPMN 2.0

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Arazm Hosieny

Konzeption und Realisierung eines automatisierten Java-Codegenerators basierend auf BPMN 2.0

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Stefan Sarstedt
Zweitgutachter : Prof. Dr. Olaf Zukunft

Abgegeben am 13. Mai 2011

Arazm Hosieny

Thema der Masterarbeit

Konzeption und Realisierung eines automatisierten Java-Codegenerators basierend auf BPMN 2.0

Stichworte

BPMN 2.0, Java, Codegenerator, MDA, Architektur

Kurzzusammenfassung

Diese Masterarbeit beschäfigt sich mit der automatischen Generierung von Java-Sourcecodes aus BPMN 2.0 Modellen. Die Codegenerierung basiert auf dem MDA-Konzept. Für das Erreichen der Zielsetzung durchleuchtet der Autor die Grundlagen der Themengebiete BPMN 2.0 und MDA. Darauf folgt eine Marktanalyse und eine Recherche existierender Ansätze. Resultierend aus den Ergebnissen kommt u.a. die oAW-Plattform für die Realisierung der automatischen Codegenerierung zum Einsatz. Ebenfalls ein Bestandteil dieser Arbeit ist die Konzeption und Architektur. Eine große Herausforderung stellen die Abbildungen der BPMN 2.0 Elemente aus einer Spezifikationsteilmenge in Java dar. Abschließend wird ein Prototyp realisiert und mit Hilfe von Testfällen getestet.

Arazm Hosieny

Title of the paper

Design and implementation of an automated Java code generator based on BPMN 2.0

Keywords

BPMN 2.0, Java, code generator, MDA, architecture

Abstract

The objective of this master thesis is the automatic generation of Java-sourcecodes relating to BPMN 2.0 models. The code generation is based on the MDA concept. In order to achieve this objective, the author scrutinises the basics of the areas of BPMN 2.0 and MDA. In a next step, a market analysis is presented along with the findings of research done with regard to the currently existing approaches. Based on these, among other things, the oAW platform serving the realisation of the automatic code generation is applied. A further part dealt with in the course of this thesis is the provision of a conception and an architecture. Quite a big challenge is the depiction of of the BPMN 2.0 elements as part of a specification subset in Java. Finally, a prototype will be worked up and tested with the help of some test examples.

Ich widme diese Arbeit ganz besonderen Menschen;
meiner Frau,
meiner Mutter,
meiner Schwiegermutter,
meinem Vater †,
und meinem Schwiegervater †.

Danksagung

Der Entstehung dieser Arbeit standen viele Menschen zur Seite. An dieser Stelle möchte ich mich für die gute Betreuung, produktive Zusammenarbeit und für das ausführliche Feedback während der Entstehung dieser Arbeit meinen besonderen Dank meinem betreuenden Prüfer Herrn Prof. Dr. Stefan Sarstedt von der Hochschule für Angewandte Wissenschaften aussprechen. Dank dieser erfolgreichen Zusammenarbeit ist ein wissenschaftlich stark gefördertes Produkt entstanden. Ein Dankeschön geht ebenfalls an meinen Zweitgutachter Herrn Prof. Dr. Olaf Zukunft für sein stets großes Interesse und gute Empfehlungen.

Im Weiteren herzlichen Dank an meine liebe Ehefrau Julia Hosieny, ebenfalls eine Masterstudentin der HAW, für ihre moralische Unterstützung und konstruktive Kritik bei der Entstehung dieser Masterarbeit. Sie hat mich von Anfang bis zum Ende des Masterstudiums begleitet und war immer eine starke Stütze. Ein weiterer Dank gilt an meine schätzenswerte Mutter, Najiba Hosieny B.A., zu der ich immer hochgeschaut habe. Ein besonderer Dank geht ebenfalls an meinen liebenswerten Neffen Behnam Subin, an meine hoch geschätzten Schwester Arian Subin und meinen liebenswürdigen Schwager Herrn Dr. Behrus Subin für die vielfältigen Hilfestellungen und die mentale Unterstützung.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

Die computerunterstützte Informationsverarbeitung sorgt für ein digitales Umfeld der neuen Generation. Mit ihrer Unterstützung sind zum ersten Mal in der Geschichte komplexe miteinander verwobene Arbeitsvorgänge automatisch steuerbar. Der digitalen Informationsverarbeitung ist ebenfalls der technische Fortschritt in den letzten 50 Jahren und somit die Steigerung der Produktivität zuzurechnen [Thome (2006)]. Seitdem sind ständig wachsende Anforderungen und eine zunehmende Komplexität der Softwaresysteme zu erkennen. Zur Bewältigung komplexer Anforderungen sind Lösungskonzepte erforderlich, die die Komplexität der Anwendungen handhabbar machen.

1.1. Motivation

Die internationale Standardisierungsorganisation Object Management Group (OMG), gegründet im Jahre 1989, gilt inzwischen als ein offenes Konsortium mit ca. 800 Firmen weltweit. Sie stellt die erforderlichen Konzepte und Technologien unter dem Begriff Model Driven Architecture (MDA) dar [oAW (2011)]. Mit dem MDA-Ansatz verschiebt die OMG Initiative den Fokus bei der Software-Entwicklung aus Manual erzeugtem Code auf die Modellierung [Thomas O. Meservy (2005)]. Es werden Technologien zur Qualitätssicherung, Kostenreduktion und Komplexitätsreduktion erarbeitet, die auf der Metaebene basieren. D.h. es sollen durch Formalisierung von Modellen mit Hilfe von Generatoren automatisch Codes generiert werden (Modellieren anstatt Programmieren) [Roland Petrasch (2006)] [Frankel (2003)].

Es existieren mittlerweile auf der Basis der MDA viele Modellierungswerkzeuge, die Plattform unabhängige Modelle entwickeln. Ein renommiertes Beispiel für die Modellierung von Klassendiagrammen stellt die UML dar, und im Bereich der Geschäftsprozessmodellierung hat sich Business Process Modeling Notation (BPMN) bewährt. Business Process Modeling Notation mit der Version 2.0 (BPMN 2.0) ist ebenso eine von der OMG und Workflow Management Coalition (WfMC) entwickelte Modellierungssprache zur Darstellung von Geschäftsprozessen [OMG (2011)].

Es existieren viele unterschiedliche Sprachen, mit denen Softwaresysteme entwickelt werden können. Heutzutage kommen in den meisten Fällen höhere Programmiersprachen zum Einsatz, da sie den Entwicklungsaufwand gegenüber anderen Programmiersprachen (Low-Level-Sprachen) schmälern. Des Weiteren bietet das objektorientierte Konzept eine große Unterstützung in der Entwicklung und Wartung von Softwaresystemen. Dieses Konzept verfolgt den Ansatz, komplexe Systeme in Komponenten und die Komponenten wiederum in Objekte aufzuteilen [E.-E. Doberkat (2002)]. Die Aufteilung erfolgt in logisch abgegrenzten Einheiten, und die Kommunikation zwischen den Einheiten verläuft unter klar definierten Schnittstellen. Zu dem objektorientierten Konzept gehört ebenfalls die Wiederverwendung von Elementen. Infolgedessen unterstützt die Objektorientierung die Verständlichkeit des Systems und somit die Handhabung. Java ist eine weitverbreitete objektorientierte höhere Programmiersprache, die die erwähnten Eigenschaften aufweist.

Die Motivation dieser Masterarbeit ist die automatische Überführung von Geschäftsprozessen in Softwaresysteme. Zu diesem Zweck sollen die oben erwähnten Themengebiete vereinigt werden. Mittels der MDA-Ansätze sollen Geschäftsprozesse, die mit Hilfe von BPMN modelliert werden, durch die Generierung von Java-Sourcecode in Softwaresysteme überführt werden. Der Generator soll für die Generierung von Softwareanwendungen, basierend auf kleineren Geschäftsprozessen, genutzt werden, um die Entwickler bei ihrer Arbeit zu entlasten, wobei Veränderungen von Unternehmensprozessen, sowie die Entwicklung neuer Geschäftsprozesse von bereits bestehenden Systemen inbegriffen sind. Zudem soll der Generator ebenso für Studenten oder ähnliche Lerngruppen zum Einsatz kommen, bei denen im Vordergrund das Verstehen und Erlernen der Modellierung von Geschäftsprozessen unter Verwendung von BPMN 2.0 stehen.

1.2. Aufgabenstellung und Zielsetzung

Das Hauptziel dieser Masterarbeit unter diesen Voraussetzungen ist zum einen die Erarbeitung eines geeigneten Systemdesigns und einer Architektur zur Realisierung eines Java-Sourcecode-Generators, zum anderen die Konzeption zur Abbildung von BPMN 2.0 Elementen in Java. Um detailliert auf die Abbildungen der BPMN 2.0 Elemente eingehen zu können, wird eine Spezifikationsteilmenge, basierend auf einem Fallbeispiel, gebildet. Der Generator soll den Namen BPMN-to-Java-Sourcecode-Generator, abgekürzt B2JSG, tragen. Des Weiteren sind für die Zielsetzung einige ergänzende nützliche Schritte durchzuführen. Zu nennen sei u.a. die Durchführung einer Marktanalyse und die Überprüfung existierender Ansätze, um zu untersuchen, mit welchen existierenden Ansätzen die Realisierung dieses Systems möglich ist. Die unterschiedlichen Alternativen für die Umsetzung des B2JSG sollen untersucht und einige verwendet werden. Die folgenden Punkte legen die Rahmenbedingungen für diese Masterarbeit fest.

1.3. Gliederung der Arbeit

Die Arbeit ist in sieben Kapitel unterteilt:

- Das zweite Kapitel schneidet die Grundlagen der BPMN 2.0 an. Es wird erörtert, zu welchem Zweck sich der Einsatz von BPMN eignet. Daraufhin wird eine Spezifikationsteilmenge der BPMN 2.0 Elementen vorgestellt. Zu den Grundlagen gehören ebenfalls die MDA-Konzepte.
- Im dritten Kapitel folgt eine Anforderungsanalyse. Es werden die technischen und fachlichen Anforderungen aufgestellt
- In Kapitel vier wird eine Marktanalyse für BPMN-Modellierungstools durchgeführt. Anschließend folgt eine Marktanalyse und eine Überprüfung/Bewertung existierender Ansätze von Codegeneratoren.
- Das fünfte Kapitel beschäftigt sich mit dem Systemdesign. Das Systemdesign umfasst die Erarbeitung einer geeigneten Architektur auf Basis der Anforderungen und der Ergebnisse der Marktanalyse. Abschließend folgen eine Aufzählung der verwendeten Technologien und deren Entscheidungshintergrund.
- Im vorletzten Kapitel folgt das Abbildungskonzept der BPMN 2.0 Elemente aus der Spezifikationsteilmenge in Java. In der Abbildungskonzeption wird ebenfalls auf die Realisierung eingegangen und auf auftretende Problematiken und Hinweise hingewiesen. Am Ende dieses Kapitels werden die Testfälle dargestellt.
- Die Masterarbeit schließt in Kapitel sieben mit einem Fazit und der Erörterung aller relevanten Ergebnisse ab. Die Arbeit schließt mit dem Ausblick ab.

2. Grundlagen

Im Kapitel Grundlagen werden die relevanten Themenbereiche dieser Masterarbeit für das Grundverständnis behandelt. Diese Themenbereiche umfassen BPMN 2.0 und die Codegenerierung. Sie legen somit das Fundament für die folgenden Kapitel. Im ersten Abschnitt des Kapitels wird einleitend auf die BPMN-Definition eingegangen, gefolgt von der Darstellung der Nutzbarkeit und der Vorstellung der Klassifikation des Prozesses auf verschiedenen Ebenen. Anschließend wird eine Teilmenge der BPMN-Spezifikationselemente, basierend auf einem Logistikprozess eines Unternehmens, bestimmt und im Detail erläutert. Im zweiten Teil dieses Kapitels wird auf die Codegenerierung eingegangen. In diesem Kontext wird auch ein Einblick in MDA und MOF gewährt.

2.1. BPMN

An dieser Stelle wird die Definition der Modellierungssprache für Geschäfsprozesse (BPMN) präsentiert.

2.1.1. Definition

The Object Management Group (OMG) has developed a standard Business Process Modeling Notation (BPMN). The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation [Group (2010)].

Der Autor hat, wie oben zu erkennen ist, die Definition von OMG als Quelle angegeben, da die OMG BPMN 2.0 spezifiziert hat. Er ist bei der Recherche auf weitere Definitionen

gestoßen [Remco M. Dijkman a (2008)] [Jan Recker (2005)], die im Wesentlichen auf die Definition der OMG zurückzuführen sind.

2.1.2. Ziel der Entwicklung und Anwendungsdomäne

Das primäre Ziel zur Entwicklung eines Standards für alle Business-Anwender wurde mit der ersten Version der BPMN-Spezifikation im Jahr 2004 unter der Leitung von Stephen A. White von IBM verwirklicht [Stephen A. White (2004)]. Sie eignet sich sowohl für die Business-Analysten, die die ersten Entwürfe der Prozesse kreieren, als auch für die technischen Entwickler, die für die Implementierung und Umsetzung der Technologie verantwortlich sind. Darüber hinaus erfüllt BPMN die Anforderungen des Managements zur Verwaltung und Überwachung der Prozesse. BPMN folgt der Tradition der Flussdiagramm-Notationen, dadurch findet sie eine hohe Akzeptanz bei den Business-Anwendern [Silver (2009)] [Group (2010)]. Im Folgenden wird vertiefend darauf eingegangen.

2.1.3. Historie

Historisch gesehen hatte vor der Entwicklung der BPMN jedes Unternehmen seine individuellen Schreibweisen, Methoden und Werkzeuge. Bei der Entwicklung der BPMN wurde von der Business Process Management Initiative (BPMI), einem Konsortium aus Vertretern von Software-Unternehmen, eine standardisierte grafische Notation zur Darstellung von Prozessbeschreibungen angestrebt. Daher hat das BPMI-Team die besten Ideen aus vielen unterschiedlichen Notationen auf dem Markt in einer einzigen Notation konsolidiert. Inzwischen ist die BPMI ein Bestandteil der Object Management Group (OMG). Die OMG und die Workflow Management Coalition stehen bei der Entwicklung und Forschung miteinander in Kooperation [Group (2010)] [Allweyer (2008)].

Die BPMN-Notation erbt und kombiniert Elemente aus einer Reihe von bisher vorgeschlagenen Notationen zur Geschäftsprozessmodellierung, darunter ebenso die XML Process Definition Language (XPDL) und Komponenten der Aktivitätsdiagramme der Unified Modeling Notation (UML). Die BPMN Prozessmodelle bestehen aus Aktivitäts-Knoten, sie modellieren Geschäftsereignisse, die die geleistete Arbeit von Menschen oder von Software-Anwendungen darstellen. Die Kontrollknoten regeln die Ablaufsteuerung zwischen den Aktivitäten. Aktivitäts- und Kontrollknoten können mittels Sequenz- und Message-Flüsse in beliebiger Weise verbunden werden [Remco M. Dijkman a (2008)]. Im Verlauf dieser Arbeit werden einige Spezifikationselemente 2.1.5 detailliert erläutert. BPMN wurde mit einer soliden mathematischen Grundlage entwickelt, so dass mit ihrer Hilfe eine präzise Ausführung

der Sprache erzeugt werden kann [Silver (2009)].

2.1.4. Klassifikation der Prozess-Ebenen (-Levels)

Von Silver und Freund werden BPMN-Prozesse in verschiedene Ebenen (Levels) unterteilt [Silver (2009)] [Jakob Freund (2010)]. Die Klassifikationen stellen den Detaillierungsgrad von Prozessen dar, sind jedoch kein Bestandteil der BPMN-Spezifikation [Group (2010)]. Silver unterteilt BPMN-Prozesse in drei Ebenen, Freund dagegen in vier. Die ersten beiden Ebenen von Freund ähneln der Darstellung vom Silver. Die dritte Ebene von Silver entspricht der dritten und vierten Ebene von Freund. Nachfolgend werden die Ebenen laut Freund in der Abbildung 2.1.4 grafisch dargestellt, die ebenso die Ebenen laut Silver darstellen, mit der Ausnahme, dass die Ebenen drei und vier der Ebene drei von Silver entsprechen. Im oberen Bereich der Pyramide ist die fachliche Sicht und unten die technische Sicht zu erkennen.

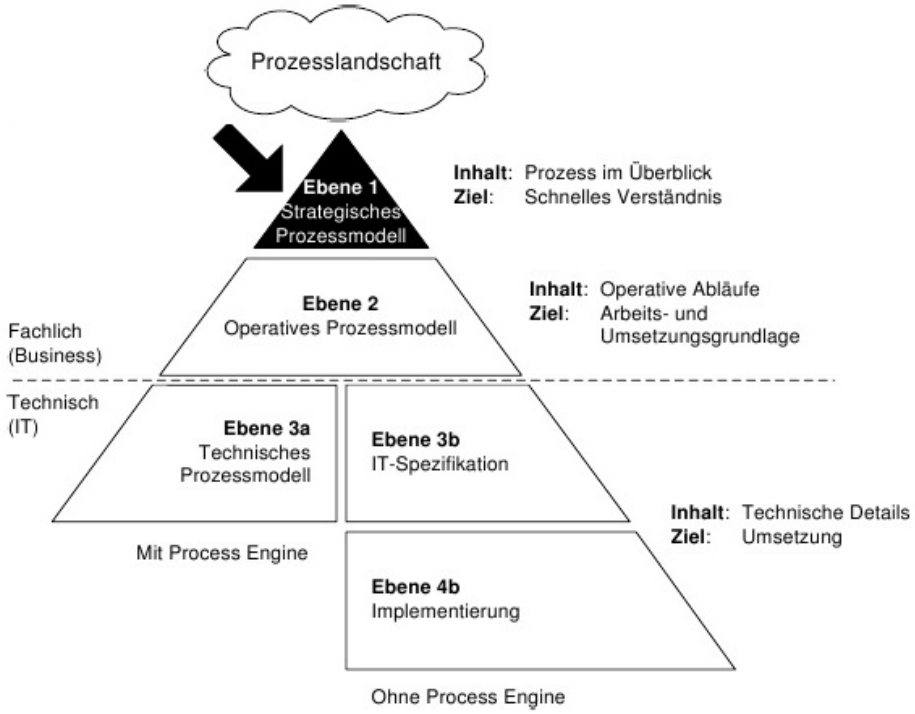


Abbildung 2.1.: Klassifikation der Prozess-Ebenen nach [Jakob Freund (2010)]

Die Klassifikation der Prozess-Ebenen ist für diese Masterarbeit ebenfalls von Relevanz, da im Verlauf dieser Arbeit zu untersuchen ist, welcher Detaillierungsgrad zu wählen ist. Es ist ebenfalls entscheidend, für welche Anwendergruppen sich der Codegenerator am besten eignet. Zunächst wird auf die Klassifikation der Ebenen eingegangen und später wird über die Zuordnung des Codegenerators auf einer Ebene reflektiert.

2.1.4.1. Descriptive modeling (Level I)

In der ersten BPMN-Ebene (beschreibende Modellierung) geht es um das Dokumentieren des Prozessablaufs. Sie verwendet einige Basis Elemente der BPMN-Notation, die durchwegs von traditionellen Flussdiagrammen zur Beschreibung der Reihenfolge der Aktivitäten und ihrer Zugehörigkeit bzw. Verantwortlichkeit in der Organisationseinheit bekannt sein dürften. Ausnahmepfade und/oder Regeln, die in der BPMN-Spezifikation vorgeschrieben sind, können hierbei ignoriert werden. Die Modellierung auf Level I erlaubt eine Verfeinerung in Level II und Level III, sodass keine größeren strukturellen Veränderungen vorgenommen werden müssen.

2.1.4.2. Analytical modeling (Level II)

BPMN Level II (analytische Modellierung) nutzt die komplette BPMN-Notation, um den Prozessablauf präziser zu beschreiben, einschließlich der Ausnahmepfade, die für die Key Performance Indikatoren wichtig sind. Auf dieser Ebene sind die Modelle nicht ausführbar. Sie vernachlässigen technische Details der Spezifikation (Datenstrukturen und Ausdrücke). Sie sind erforderlich, um das Modell auf einer Prozess-Engine ausführen zu können. BPMN Level II spiegelt eine business-orientierte Perspektive wider und wird von Business-Analysten verstanden.

2.1.4.3. Executable modeling (Level III)

BPMN Level III (ausführbare Modelle) ist vergleichbar mit der zweiten Ebene, mit dem Unterschied, dass die ausführbaren Details vollständig in BPMN-Standard-Attribute realisiert sind. Dieses Level umfasst Level III und IV des Freund'schen Denkansatzes. Das erzeugte Modell weist einen hohen Detaillierungsgrad auf Level drei auf, sodass es direkt zur Automatisierung eines Geschäftsprozesse verwendet werden kann [Grass (2010)].

Ein grundlegendes Ziel der OMG Model Driven Architecture (MDA) ist, Systeme mit grafischen Modellen anstatt der Codierung, zu erzeugen. Dieses ist ebenfalls das Ziel dieser Masterarbeit. BPMN auf der Ebene III ist ausgerichtet auf Entwickler, nicht auf SW-Architekten oder Analysten. Die automatisierte Generierung eines Systems erfordert einen sehr hohen Detaillierungsgrad. Es wird das Ziel angestrebt, Entwickler in die Lage zu versetzen, aus einem BPMN-Diagramm (Modell) mit Hilfe des Codegenerators ein System oder Teile eines Systems zu erstellen. Für die Umsetzung des Codegenerators käme Level III in Frage.

2.1.5. BPMN 2.0 Spezifikation

Wie bereits erwähnt, ist BPMN 2.0 durch die OMG spezifiziert. Das nachfolgende Unterkapitel umfasst einen Einblick in die Spezifikationselemente der BPMN 2.0. Basierend auf einem Fallbeispiel 2.3, wird eine Teilmenge der Spezifikationselemente bestimmt. Unter anderem soll es möglich sein, aus diesem Fallbeispiel am Ende der Realisierung 6.6 den Java-Sourcecode zu generieren. Für einen detaillierteren Einblick wird auf die BPMN-Spezifikation [Group (2010)] der OMG verwiesen.

2.1.5.1. BPMN 2.0 Basiselemente

Die OMG teilt die Spezifikationselemente der BPMN 2.0 in fünf Basiskategorien ein. Abbildung 2.2 stellt die fünf Basiskategorien als Baumgraphen mit den Zweigen und Blättern dar. Die Blätter entsprechen den Spezifikationselementen, die wiederum weitere Unterelemente beinhalten können.

Im Folgenden werden die fünf verschiedenen Basiskategorien der BPMN 2.0 Notationselemente vorgestellt.

Flow Objects
Flow Objects sind eine der wichtigsten Basiselemente, mit denen das Verhalten von Business Prozessen definiert werden kann. Sie werden in drei Flow Objekte (Events, Activities und Gateways) unterteilt. Im Baumgraphen sind sie als Blätter der Flow Objekte abgebildet.

Data
Data spezifizieren die Kommunikation zwischen Prozessen. Sie bieten Activities die Möglichkeit, gespeicherte Informationen außerhalb der Prozessgrenzen abzurufen oder zu aktualisieren, wobei Activities die zu verrichtende Arbeit sind, die in einem Unternehmensprozess

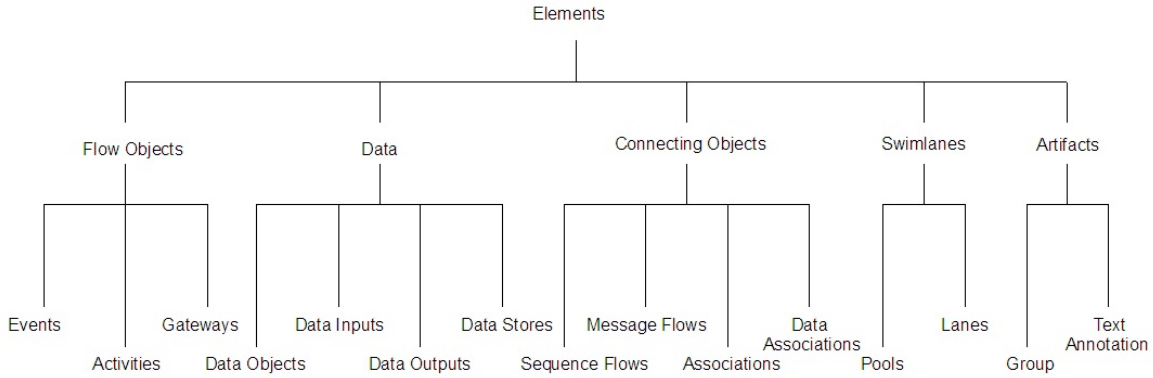


Abbildung 2.2.: Darstellung der BPMN 2.0 Spezifikationselemente als Baumgraphen

durchgeführt wird. Darauf wird später näher eingegangen. Ebenso dienen sie der Datenmodellierung. Zum Beispiel wird bestimmt, ob persistente Daten ein- (Input) oder abfließen (Output). Das Lesen eines Dokuments wäre in diesem Fall der Input und das Erstellen oder Bearbeiten der Output.

Connecting Objects

Es existieren vier Möglichkeiten, Flow Objects miteinander oder mit Informationen zu verbinden. Diese Möglichkeiten werden unter dem Begriff Connection Objects zusammengefasst. Einige dieser Verbindungsobjekte sind für das Erstellen des Fallbeispiels 2.3 unbedingt erforderlich.

Swimlanes

In einem BPMN-Diagramm können mehrere miteinander kommunizierende Prozesse modelliert werden. Swimlanes abstrahieren die beiden existierenden Alternativen; Pools und Lanes.

Artifacts

Artifacts werden verwendet, um zusätzliche Informationen über den Prozess zu liefern. Es wird zwischen zwei standardisierten Artifacts unterschieden, aber es ist den Modellierern oder den Modellierungstools freigestellt, so viele Artifacts hinzuzufügen, wie sie benötigen. Es könnte eine zusätzliche Aufgabe darin bestehen, eine größere Anzahl von Artifacts zu standardisieren, der für den generellen Gebrauch anwendbar wäre. Die aktuelle Liste von Artifacts beinhaltet „Group“ und „Text Annotation“.

2.1.5.2. Spezifikationsteilmenge, basierend auf einem Fallbeispiel

In BPMN 2.0 existieren die Notationselemente der Basiskategorien, um Prozesse grafisch abzubilden. Dazu besteht die Möglichkeit, die gesamte Menge der Spezifikationselemente zu verwenden. Allerdings würde das Realisieren einschließlich des Abbildens aller Elemente den Rahmen dieser Masterarbeit sprengen, da im Verlauf dieser Masterarbeit der theoretische sowie der praktische Teil durchgeführt wird. Eine Alternative bestände darin, alle Elemente aus einem Ast des Baumgraphen 2.2 zu wählen. Doch es lassen sich mit zu einem Ast gehörigen Elementen auch keine vollständigen Prozesse abbilden. Der Autor hat sich für eine Teilmenge der Spezifikationselemente aus mehreren Ästen entschieden. Es ist am einfachsten, eine geeignete Menge zu finden, indem ein Prozess modelliert wird, und die enthaltenen Elemente als Teilmenge dienen. Als Beispiel sei die nachfolgende Abbildung 2.3 zu verwenden:

Fallbeispiel für ein Logistiksystem
Bei der Abbildung 2.3 handelt es sich um ein BPMN-Diagramm. Das BPMN-Diagramm stellt ein vereinfachtes Zusammenspiel eines Verschiffungsprozesses mit seinem Businesspartner und dem Kunden dar. Dieses Fallbeispiel verwendet eine gebräuchliche Teilmenge der Notationselemente der BPMN 2.0, mit der Prozesse abgebildet werden können. Dabei wird gezeigt, wie der Kundenauftrag für eine Güterverschiffung abgewickelt wird:

Es kommt ein Verschiffungsauftrag des Kunden an. Das Verschiffungsunternehmen nimmt diesen Auftrag entgegen, welches für die Transportverwaltung der Seefracht zuständig ist. Die Ladestelle belädt die Ware, parallel bereitet der Service die Dokumente vor und steht für Kundenanfragen zur Verfügung. Der Kunde kann jederzeit vor der Ankunft des Containers im Zielort den Status anfragen. Eine Alternative besteht darin, den Kunden vom Service über den Status informieren zu lassen. Ist die Beladung erfolgreich, wird der volle Container der Spedition zum Transport übergeben. Andernfalls wird über den Service der Kunde informiert. Während die Spedition den Container liefert, schickt der Service die Dokumente in die Reederei. Liegen die Dokumente und der Container bei der Reederei vor, wird der Container verschifft. Kommt der Container im Zielort an, geht eine Bestätigung über den gelieferten Container an den Service. Der Service teilt dies dem Kunden mit, und der Prozess wird terminiert. Bei der vereinfachten Prozessdarstellung wurde darauf verzichtet, die Containerbestellung seitens des Verschiffungsunternehmens darzustellen. Zudem muss das Verschiffungsunternehmen einen Container bei der Reederei voranmelden, bevor die Spedition aktiv wird.

Für die Darstellung des Logistikprozesses wurden einige Spezifikationselemente verwendet. An dieser Stelle geht der Autor auf die einzelnen hier eingesetzten Notationselemente der BPMN 2.0 ausführlicher ein:

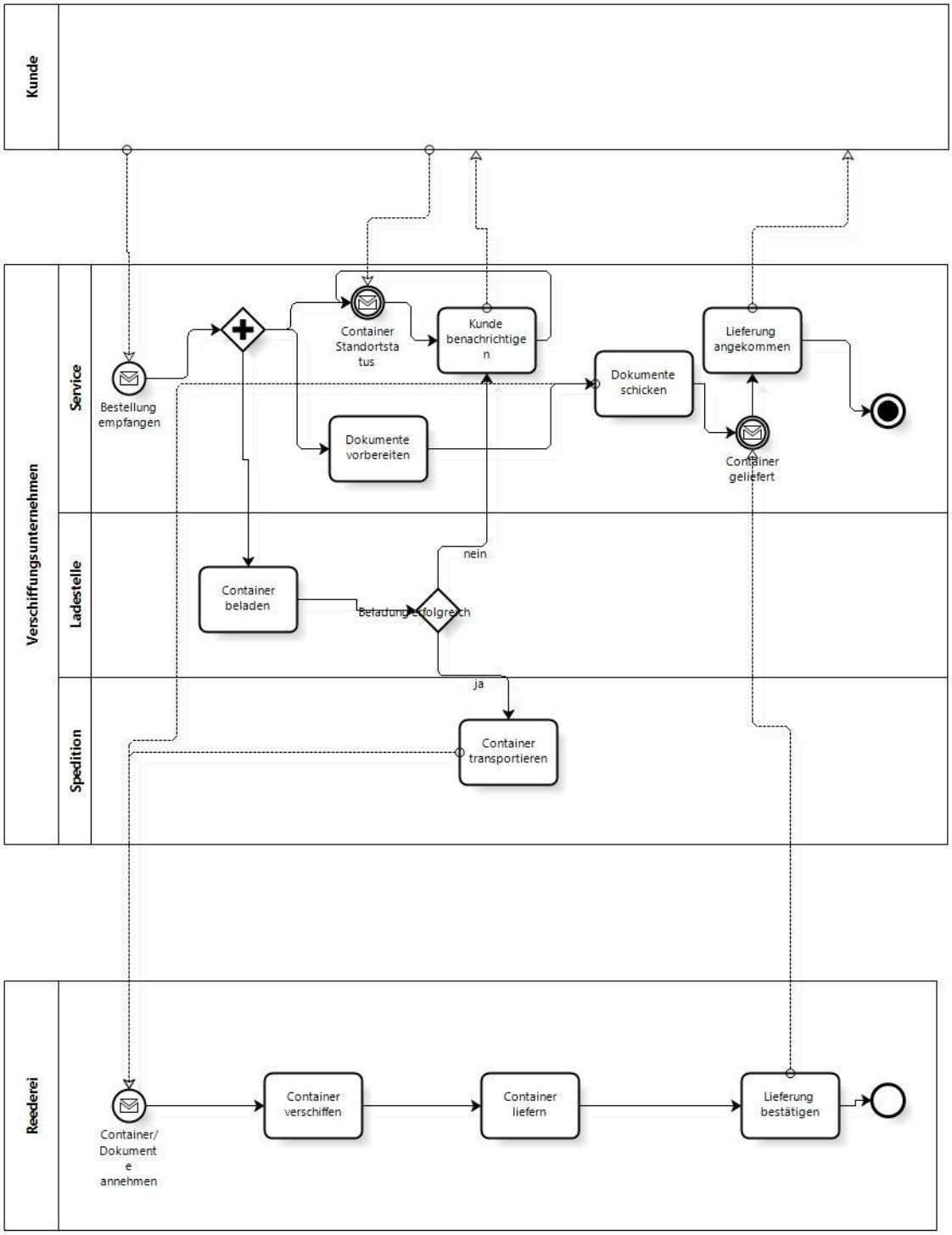


Abbildung 2.3.: Fallbeispiel für ein Logistiksystem

Flow Objects: Events

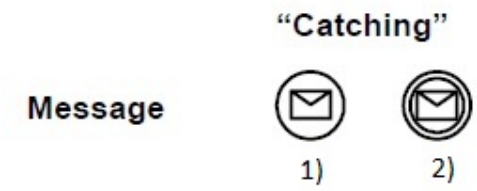
Ein Event (deutsch: Ereignis) definiert ein Ereignis. Ereignisse können beispielsweise erreichte Resultate (End-Events) oder wichtige Meldungen während des Prozessablaufs signalisieren. Dieses Ereignis wird durch einen Auslöser (Trigger) angesteuert. Generell wird ein Ereignis als ein Kreis mit leerem Kern dargestellt, der im inneren einen Marker zur Differenzierung verschiedener Auslöser oder Auswirkungen enthält. Es existieren drei Arten von Ereignissen, abhängig davon, an welcher Stelle sie innerhalb des Prozessverlaufs auftreten. Nachfolgend die drei Arten:

1. Start: Wie der Name andeutet, zeigt das Start Event, den Startpunkt eines Prozesses.
2. Intermediate: Das Intermediate Event tritt innerhalb eines Geschäftsprozesses zwischen dem Start- und Endevent auf. Es beeinflusst den Ablauf des Prozess, aber startet bzw. beendet den Prozess nicht.
3. End: Das End Event beendet einen Prozess.

Notation	
Event (Generell)	
Start Event	
Intermediate Event	
End Event	

Ein Event kann durch einen sogenannten Marker detaillierter beschrieben werden. In der grafischen Darstellung wird dies durch ein Icon, das innerhalb des Event-Kreises platziert ist, veranschaulicht. Die Marker können entweder beschreiben, wie ein Event angetriggert, also ausgelöst wird („catch“) oder welches Resultat („throw“) erstellt wird. Start Events können lediglich Catch-Marker, End Events nur Throw-Marker und Intermediate Events beide Typen enthalten. Darüber hinaus werden Events von der OMG in verschiedene Typen aufgeteilt. An dieser Stelle soll jedoch nicht näher darauf eingegangen werden. Mehr Informationen dazu erhält der Leser in der einschlägigen Literatur [Group (2010)]. Im Fallbeispiel werden lediglich die nachfolgenden Events eingesetzt:

1. „Message Start Event“ für „Bestellung empfangen“ und „Container/Dokumente annehmen“
2. „Message Intermediate Event“ für den „Container Standortstatus“ und „Container geliefert“



Flow Objects: Activities

Eine Activity ist ein Oberbegriff für die Arbeit, die das Unternehmen in einem Process ausführt. Eine Activity kann atomar oder nicht-atomar (zusammengesetzt) sein. Die Activity-Arten, die einen Teil des Prozessmodells bilden, sind: Task & Sub-Prozesse, wobei unter zwei Tasks und vier Sub-Processen unterschieden wird. Der wesentliche Unterschied zwischen Task und Sub-Process besteht darin, dass Tasks atomar und Sub-Prozess nicht atomar sind. Das bedeutet, dass ein Sub-Process eine zusammengesetzte Activity darstellt, die in feinere Details aufgeteilt werden kann. Der Verfasser dieser Arbeit beschränkt sich auf Tasks.

Tasks: Wie bereits erwähnt, handelt es sich bei den Tasks um atomare Activities, die innerhalb eines Prozesses enthalten sind. Ein Task wird verwendet, wenn die Arbeit in dem Prozess nicht in feinere Activities unterteilt werden kann.

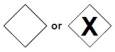
Notation	
Activity (Generell)	
Task (Atomic)	

Flow Objects: Gateways

Ein Gateway wird verwendet, um die Divergenz und Konvergenz eines Sequence Flows in einem Prozess und in einer Choreografie zu kontrollieren. Es ist möglich, mit Hilfe von Gateways Verzweigungen und Zusammenführungen der Pfade zu bestimmen. Die Notation besteht aus einer leeren Raute, die Markierungen innerhalb des Hohlkörpers erlaubt. Die

Markierungen repräsentieren die Art der Verhaltenskontrolle. Nach der „Bestellung empfangen“ kommt im Fallbeispiel ein „Parallel Gateway“ zum Einsatz. Dieser erlaubt die parallele Ausführung von „Container Standortstatus“, „Dokumente vorbereiten“ und „Container beladen“. Nach der Ausführung des „Container beladen“ kommt ein „Exclusive Gateway“ (Beladung erfolgreich) zum Einsatz.

Notation

Gateway (Generell)	
Exclusive: Entscheidung und Zusammenführung	
Parallel: Gabelung und Zusammenführung	

Connecting Objects

Es existieren vier alternative Möglichkeiten, Flow Objects miteinander oder mit Informationen zu verbinden. Im BPMN-Diagramm sind die ersten beiden der vier nachfolgenden Objekte (Connecting Objects) realisiert:

Connections Objects: Sequence Flows

Ein Sequence Flow wird verwendet, um innerhalb eines Prozesses die Reihenfolge, der Flow Objects und deren Ausführungen darzustellen. Im BPMN-Diagramm des Fallbeispiels ist das beispielsweise am Prozess der Reederei zu beobachten. Vom Start-Event (Container/Dokumente annehmen) ausgehend führt ein „Sequence Flow“ zur nächsten Activity (Container Verschiffen).

Notation

Sequence Flow	

Connections Objects: Message Flows

Message Flows sind dafür zuständig, den Nachrichtenfluss zwischen mindestens zwei Teilnehmern, die sende- und empfangsbereit sind, über die Prozessgrenzen hinaus darzustellen. Im BPMN-Diagramm stellen die „Message Flows“ die Kommunikation zwischen dem Verschiffungsunternehmen, dem Kunden und der Reederei sicher. Demnach stellen das Verschiffungsunternehmen, der Kunde und die Reederei die Teilnehmer des Prozesses dar.

Notation

Message Flow

Swimlanes

Es existieren zwei Wege der Gruppierung der primären Modellierungselemente mit Swimlanes. Im BPMN-Diagramm sind im Verschiffungsprozess, sowohl Pools, als auch drei Lanes (Service, Ladestelle und Spedition), zu beobachten.

Swimlanes: Pools

Ein Pool ist die grafische Darstellung eines Teilnehmers in einer Zusammenarbeit (Collaboration). Er agiert auch als Swimlane und stellt einen grafischen Behälter dar, der eine Partitionierung für eine Reihe von Activities aus anderen Pools verkörpert, in der Regel im Kontext von B2B Situationen. Ein Pool kann interne Details aufweisen oder kann als eine „Black box“ definiert sein.

Notation

Pool

Swimlanes: Lanes

Eine Lane ist eine Sub-Partition innerhalb eines Pools und erweitert die gesamte Länge eines Pools, entweder vertikal oder horizontal. Lanes werden verwendet, um die „Flow-Objects“ zu organisieren und kategorisieren. Sie repräsentieren beispielsweise Rollen oder Organisationseinheiten im Unternehmen.

Notation

Lane

Orchestration vs Choreography und Collaboration

Zum Abschluss dieses Abschnitts geht der Autor auf drei Begriffe ein, die ebenfalls ein

Bestandteil von BPMN 2.0 sind und im BPMN-Diagramm des Fallbeispiels 2.3 zur Anwendung kommen. Drei in diesem Zusammenhang sehr relevante Begriffe sind Orchestration, Choreography und Collaboration. Dieser Abschnitt wird die Termini erläutern, die im Praxisbeispiel exemplarisch den Prozessablauf des Verschiffungsunternehmens visualisieren.

Orchestration
Orchestration Modelle repräsentieren eine spezifische Geschäfts- oder Organisationen-Sicht eines Prozesses, wobei die BPMN-Spezifikation den Begriff der Orchestration als Synonym für Prozesse verwendet. Unter einem Prozess wird in BPMN, der Ablauf von Aktivitäten innerhalb einer Organisation, verstanden. Die Durchführung der Güterverschiffung im Verschiffungsunternehmen ist ein Beispiel für einen Prozess (Orchestration) [Group (2010)] [Allweyer (2009a)] [Stephen A. White (2008)].

Choreography und Collaboration
Ein Collaboration-Diagram enthält zwei oder mehrere Pools, die miteinander interagieren. Das in BPMN 2.0 dargestellte BPMN-Diagramm der Abbildung 2.3 ist ebenfalls ein Collaboration-Diagramm. Neben den Aktivitätenabläufen (activity-flows) des internen Verschiffungsprozesses, zeigt das Diagramm die Interaktionen des eigenen Prozesses mit externen Teilnehmern (sichtbar durch Nachrichtenflüsse oder message flows). Dabei werden Nachrichtenflüsse an der Grenze des Black-Box-Pools (Kunde) angelegt und mit den Aktivitäten (activities) und Ereignissen (events) des internen Verschiffungsprozesses direkt verbunden. In BPMN 2.0 werden die Nachrichtenflüsse zwischen dem Kunden und dem Verschiffungsunternehmen als Collaboration bezeichnet; die Nachrichtenflüsse zwischen dem Verschiffungsunternehmen und seinem Business-Partner Reederei (B2B) werden Choreography genannt.

2.2. Codegenerator und MDA

Im letzten Abschnitt hat der Autor exemplarisch einen Logistikprozess 2.3 gezeigt. Allerdings ist das Hauptziel dieser Masterarbeit die Architektur und Realisierung eines automatisierten Java-Codegenerators, basierend auf BPMN 2.0. Auf Grund dessen steht ebenfalls die Codegenerierung im Mittelpunkt. Unter diesem Gesichtspunkt geht der Verfasser dieser Masterarbeit auf die Codegenerierung ein.

Ein Codegenerator ist ein Übersetzer, der dazu dient, aus einem Modell einen Quellcode einer Programmiersprache zu erzeugen. Eine weitverbreitete Möglichkeit, einen Code zu

generieren, ist die Verwendung eines Metamodells. Abstrakt betrachtet, beschreibt das Metamodell die Elemente einer Modellierungssprache, ihre Beziehungen untereinander sowie Einschränkungen bzw. Modellierungsregeln. Weiter unten wird darauf näher eingegangen. Im Prinzip durchläuft ein Codegenerator die nachfolgenden fünf Phasen nacheinander, um vom Modell ausgehend den Sourcecode zu generieren:

- Modelle einlesen
- Modelle verlinken - Modelle gleicher o. verschiedener Instanzen miteinander verweben
- Modelle validieren
- Modelle transformieren
- Code generieren (auf Basis von Modellen)

[Thomas Stahl (2007)] [Thomas Stahl (2005)]

2.2.1. Codegenerierung

Die Codegenerierung steht in direkter Verbindung mit der Model Driven Architecture (MDA). Wie bereits erwähnt unterliegt MDA wie BPMN 2.0 der Führung der Standardisierungsinitiative der OMG. Sie befasst sich mit der modellgetriebenen Softwareentwicklung (MDSD - Model Driven Software Development) und sorgt für das Zusammenführen einer Reihe neuer Trends im Unternehmen.

MDA dehnt sich über eine Reihe von Technologien sowie Komponenten-basierte Entwicklungen, Design Patterns, Middleware und mehrstufige Systeme aus. Sie trägt dazu bei, diese Technologien zu unterstützen [Frankel (2003)]. MDSD steht für Methoden bzw. Techniken, mit deren Unterstützung aus formalen Modellen automatisiert lauffähige Software erzeugt werden. MDA strebt nach Interoperabilität zwischen Werkzeugen und Standardisierung von Modellen für Anwendungsbereiche. Roland Petrasch: „Einmal modellieren statt immer wieder zu programmieren“ [Roland Petrasch (2006)]. MDSD konzentriert sich auf die Bereitstellung von praktisch einsetzbaren Bausteinen für Softwareentwicklungsprozesse [Thomas Stahl (2007)].

2.2.2. MOF

Meta Object Facility (MOF), in Zusammenhang mit der Codegenerierung und der modellgetriebenen Softwareentwicklung, bildet die Basis der MDA. In der Meta-Architektur ist MOF in der Metametamodellebene (M3) angesiedelt. Insgesamt benutzt die OMG eine vier Modellierungsebenen-Architektur (four-layered architecure). In der OMG Terminologie werden diese Ebenen als M0, M1, M2 und M3 bezeichnet [Anneke Kleppe (2004)]. Mit MOF lassen sich fast alle Modellierungssprachen auf der Metaebene (M2) beschreiben. Sie verfügt über Konzepte zur Definition oder Manipulation von Modellen, abgeleitet vom Metamodell. Dazu nutzt MOF eine UML-Teilmenge als Notation. Unter anderem wird UML selbst über MOF definiert [Roland Petrasch (2006)] [OMG]. Die nachfolgende Abbildung 2.4 verdeutlicht die Meta-Architektur unter Verwendung des UML-Metamodells. Sie stammt von Blanc [X. Blanc (2000)] und wurde vom Autor überarbeitet. Auf der linken Seite der Abbildung ist ein Beispiel abgebildet, und rechts davon auf der selben Höhe wurde die entsprechende Ebene platziert:

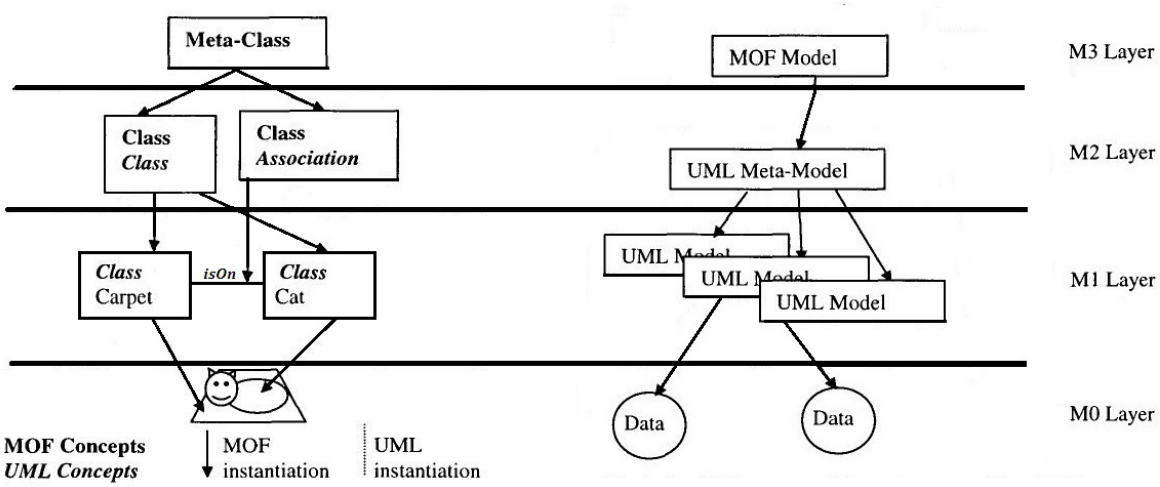


Abbildung 2.4.: OMG four-layer [X. Blanc (2000)]

Der Unterschied zwischen den einzelnen Ebenen besteht durchgängig darin, dass die obere Ebene die untere modilliert. Das bedeutet, M1 ist das Modell von M0, M2 das Modell von M1 und M3 das Modell von M2. Das Modell eines Modells wird als Metamodell bezeichnet und konsequenterweise ein Modell eines Metamodells als ein Metametamodell.

Um auf das Beispiel der vier Ebenen zurückzukommen, bildet die unterste Ebene M0 die Realität ab. Als Beispiel sei hierfür die Abbildung 2.4 mit der Katze auf dem Teppich benutzt. Die Modellierung der Katze auf dem Teppich führt zur nächst höheren Ebene M1. Dafür kann eine beliebige Modellierungssprache gewählt werden. Für dieses Beispiel wird UML

verwendet. Daraus entsteht eine Klasse „Cat“ und eine Klasse „Carpet“ mit einer Assoziation „isOn“.

Die nächste höhere Ebene M2 modelliert das Modell (Metamodell). Infolgedessen wird in diesem Beispiel die Modellierungssprache UML modelliert, d.h. es entsteht ein UML-Metamodel. Wie bereits oben erwähnt und in der Abbildung zu sehen, wird die UML-Notation für die Meta- und Metametamodellierung verwendet. Im Beispiel 2.4 besteht das UML-Metamodel aus der Klasse „Class“ und der Klasse „Association“. Die Modellierung des Metamodells ergibt das MOF-Modell.

[OMG] [X. Blanc (2000)] [Roland Petrasch (2006)][Frankel (2003)]

2.2.3. Modell-Transformationen

Zu den Grundkonzepten der MDA gehört überdies ein Verständnis von Platform Independent Model (PIM), Platform Specific Model (PSM), Model-Model Transformation und Model- Code Transformation, die zu den Grundlagen gehören und im Verlauf der Arbeit eine wichtige Rolle spielen werden. Abbildung 2.5 visualisiert, in welchem Kontext die Termini verwendet werden. Anschießend werden sie vom Autor näher erläutert.

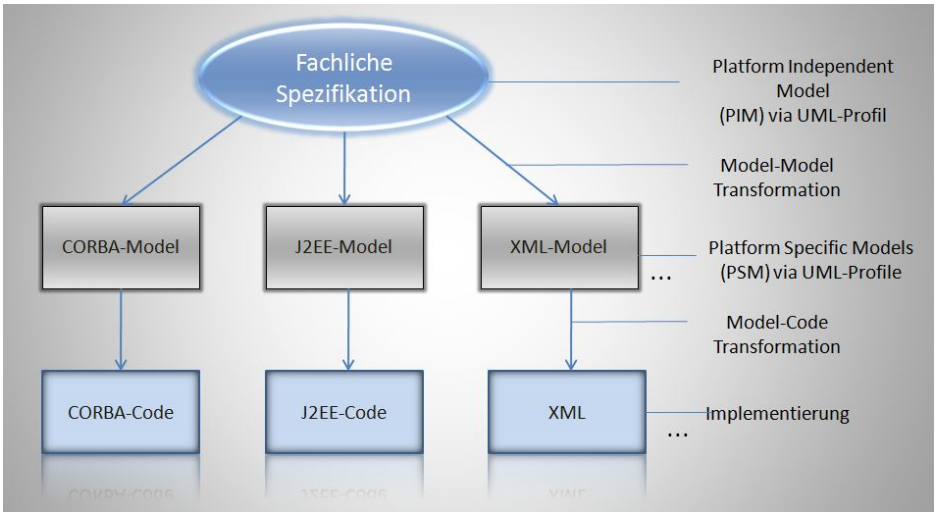


Abbildung 2.5.: MDA Grundprinzipien [Peter Roßbach (2003)]

PIM steht in der modellgetriebenen Softwareentwicklung für das plattformunabhängige Modell. Das Modell ist plattformunabhängig und spezifiziert die Fachdomäne. PIM beschränkt sich auf die Modellierung der fachlichen Aspekte. Für die formale Modellierung hat sich die UML-Notation herauskristallisiert [UML (2011)]. Durch eine Modelltransformation wird das fachlich vollständig abhängige Modell in eine PSM überführt. Wie die Abbildung zeigt, kann

beispielsweise ein CORBA-Model, J2EE-Model und XML-Model eine mögliche Zielplattform sein. Zum einen existiert die Möglichkeit, mit Hilfe von Werkzeugen PSM-Modelle durch Model-Model Transformation in andere PSM-Modelle zur transferieren. Zum Anderen können die PSM-Modelle durch Model-Code Transformation in Codes umgewandelt werden. Die direkte Transformation von einem PIM-Modell in einen Code wird als Translationist bezeichnet. Dagegen wird von einem Elaborationist gesprochen, wenn eine Transformation von einem PIM-Modell in einen Code mit der Zwischentransformation eines PSM-Modells durchgeführt wird.

[Y.CHE (2008)] [Peter Roßbach (2003)] [Thomas Stahl (2005)] [Sommerville (2007)] [Thomas Stahl (2007)] [Roland Petrasch (2006)]

3. Anforderungsanalyse

Zu Beginn dieses Kapitels werden Anforderungen an das System dargelegt, um das am Anfang dieser Masterarbeit vordefinierte Hauptziel realisieren zu können. Dabei soll ein System (B2JSG) entwickelt werden, das aus der ausgearbeiteten Teilmenge der BPMN-Spezifikationselemente im Abschnitt 2.1.5 Java-Sourcecode generiert wird.

Der Verfasser dieser Arbeit geht nach Starkes [Starke (2009)] Verfahren zur Anforderungsanalyse vor. Die Anforderungsanalyse beinhaltet die Erstellung einer Systemidee sowie die Aufstellung technischer und fachlicher Anforderungen.
Für die Entwicklung der Systemidee definiert Starkes einige Schlüsselfragen, die vom System-Architekten im Folgenden kurz und prägnant beantwortet werden:

Was ist die Kernaufgabe des Systems? Welches sind die wichtigsten Elemente der Fachdomäne?
Die Kernaufgabe des Systems besteht darin, aus einem modellierten BPMN-Diagramm, automatisiert Java-Sourcecode zu generieren. Zusätzlich soll das System dem aktuellen Stand entsprechen, und demnach müssen die BPMN-Diagramme der BPMN-Spezifikation 2.0 unterliegen. Der wichtigste Aspekt der Fachdomäne ist die Prozessgenerierung auf der Sourcecode Ebene.

Wie wird das System genutzt?
Der Benutzer erstellt ein BPMN-Diagramm mit einem externen Tool, welches mit dem B2JSG kooperiert. Basierend auf diesem Diagramm wird der Java-Sourcecode automatisch generiert.

Von wem wird das System genutzt?
Das System wird von Software-Entwicklern zur Softwareerstellung genutzt, um Teile eines Systems automatisch aus BPMN-Modellen generieren zu lassen. Anschließend soll das System mit geringem Aufwand erweitert werden können. Zusätzlich sollen Studenten und andere Lerngruppen das System zur Lernzwecken verwenden können.

Über welche Schnittstellen zu anderen Systemen verfügt das System?
Das System soll mit einem Modellierungstool kooperieren können. Das externe Tool soll die Möglichkeit bieten, ein BPMN-Diagramm zu erstellen. Das Output des externen Tools dient für dieses System als Input.

Wie wird das System gesteuert?
Der Benutzer steuert die Generierung an.

Welche Art Benutzeroberfläche hat das System?
Diese Frage wird zu einem späteren Zeitpunkt beantwortet. Die Benutzeroberfläche wird abhängig von der Architektur bestimmt.

3.1. Anforderungen

Die Anforderungen eines Systems orientieren sich grundsätzlich an den individuellen Unternehmens-Prozessen. Dabei werden Anforderungen in technische und fachliche Anforderungen gegliedert. Die fachlichen Anforderungen beschreiben die fachliche Sicht auf das System. Die technischen Anforderungen orientieren sich am technischen Umfeld des Softwareprojekts, umfassen zum Beispiel alle relevanten technischen Voraussetzungen, die den Java-Codegenerator (B2JSG) betreffen.

3.1.1. Technische Anforderungen

Die technischen Anforderungen legen Qualitätsmerkmale fest, die das System (B2JSG) nach der Entwicklung aufweisen muss:

Funktionalität / Kompatibilität
Das System muss mit anderen Systemen kooperieren können. Ein anderes System (Modellierungstool) erzeugt ein BPMN-Diagramm, daraufhin generiert B2JSG den Java- Sourcecode zum dazugehörigen BPMN-Diagramm. Zu diesem Zweck muss der B2JSG den Output des Modellierungstools (BPMN-Diagramms) parsen, interpretieren und verarbeiten.

Zuverlässigkeit
Hierbei wird vom B2JSG verlangt, bei der Ausführung korrekte Ergebnisse zu liefern. D.h. der resultierende Sourcecode muss exakt das widerspiegeln, was im BPMN-Diagramm fixiert wurde.

Übertragbarkeit
Das System muss Plattform unabhängig sein und auf verschiedene Computersysteme mit unterschiedlicher Architektur, unterschiedlichem Prozessor, Compiler und Betriebssystem übertragen werden können.

Benutzerfreundlichkeit
Die Einbindung der generierten Java-Sourcecode-Dateien in eine Entwicklungsumgebung soll unterstützt werden. Damit würden die erzeugten Dateien ohne nachträgliches Importieren in die Entwicklungsumgebung eingebunden werden. Somit würde der Benutzer bzw. Entwickler die Möglichkeit haben, problemlos den generierten Sourcecode einzusehen, zu verändern und auszuführen.

Wiederverwendbarkeit / Änderbarkeit
Die Software-Module müssen so konzipiert werden, dass sie für ein anderes System wiederverwendet werden können. Dazu ist eine geeignete Architektur notwendig, die aus lose gekoppelten und kohärenten Komponenten besteht. Lose Kopplung ermöglicht geringe Auswirkungen auf andere Module. Gleichzeitig können durch geringe Veränderungen Module in anderen Kontexten wiederverwendet werden. Durch Kohärenz werden untereinander stark abhängige Elemente zusammengeführt.

Erweiterbarkeit
Unter anderem muss das Hinzufügen von weiteren Spezifikationselementen mit geringem Aufwand in B2JSG realisierbar sein. Eine weitere Zielsetzung bei der Konzeption ist die Austauschbarkeit des Modellierungstools.

[Kleuker (2009)] [Müller-Lindenberg (2005)] [Zöller-Greer (2002)]

3.1.2. Fachliche Anforderungen

In diesem Abschnitt werden primär die Anforderungen beleuchtet, die üblicherweise aus fachlicher Sicht aufgestellt werden. Anhand dieses Überblicks über die fachlichen Zielsetzungen und Anforderungen wird das Einsatzgebiet des B2JSG aufgezeigt. Die Abbildung 3.1 weist fünf Use-Cases auf. Zwei dieser Use-Cases, „Java-Sourcecode ansehen“ und „Java-Sourcecode in Projekt importieren“, stellen zwar nicht der Kern dieser Arbeit dar, dennoch sorgen sie dafür, dass eine gebündelte Lösung für die Anwender zur Verfügung gestellt wird. Nach Thomas Stahl [Thomas Stahl (2007)] „bedarf es einer Plattform, um die einzelnen Lösungen (im Folgenden Komponenten genannt) zu einem Generator zusammenzusetzen, auf der die verschiedenen Komponenten über eine einfache Konfiguration zusammengeschachtelt und individuell konfiguriert werden können“. Nachfolgend sind die in der Praxis anzutreffenden Szenarien dargestellt:

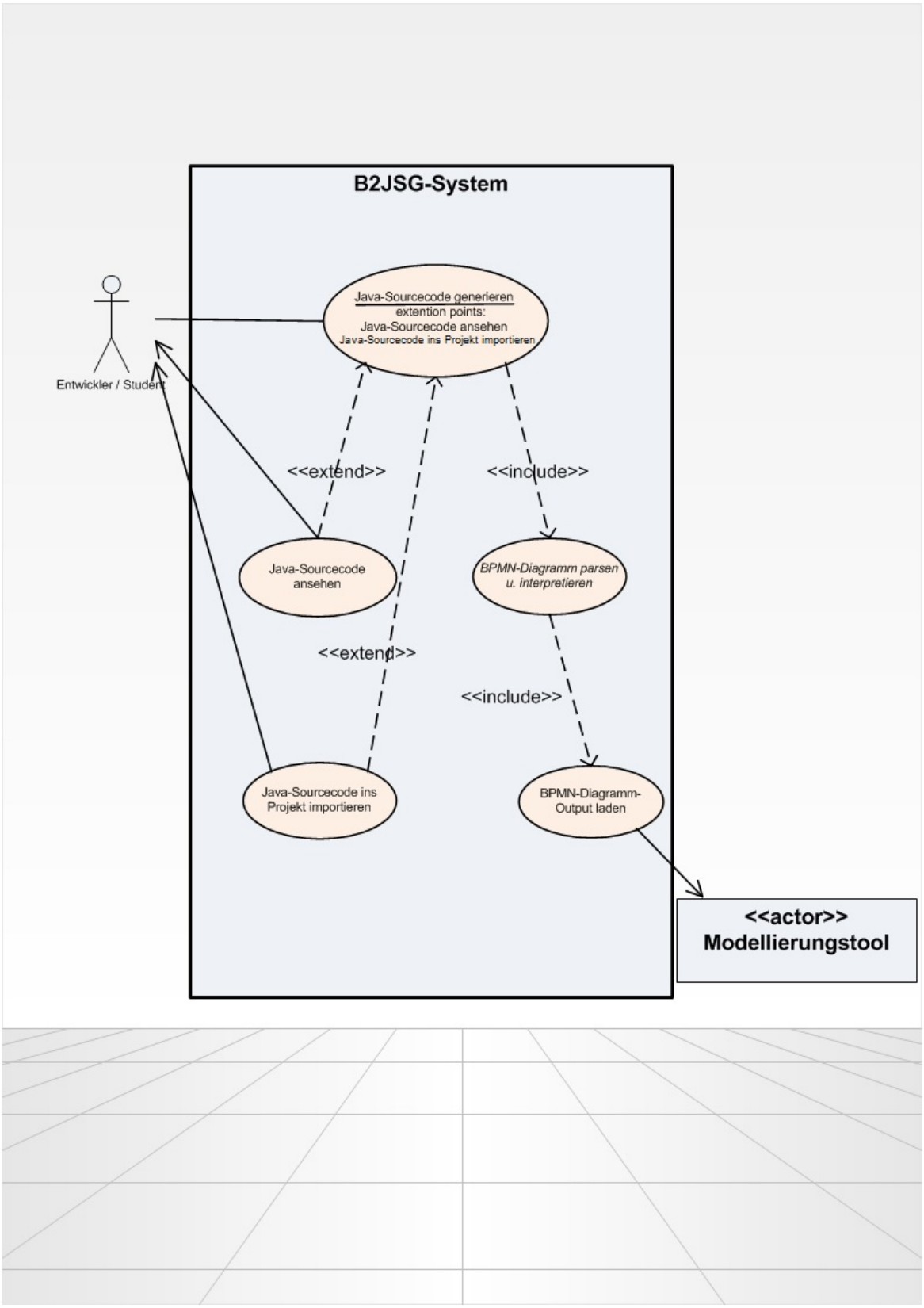


Abbildung 3.1.: Anwendungsfall-Diagramm

Titel	Java-Sourcecode generieren
Akteur	Anwender
Ziel	Der Anwender möchte aus einem BPMN-Diagramm (Modell), den Java-Sourcecode erzeugen.
Auslöser	Der Anwender entscheidet sich, aus dem BPMN-Diagramm den Java-Sourcecode generieren zu lassen, und steuert diese Aktion an.
Vorbedingungen	BPMN-Diagramm-Output muss vorhanden sein
Nachbedingungen	Der Anwender erhält den Java-Sourcecode zum BPMN-Diagramm
Erfolgsszenario	1. Der Anwender ruft das B2JSG-System auf 2. Er navigiert zum Pfad des BPMN-Diagramms 3. Danach führt er die Funktion generiere Javacode aus 4. Das System greift auf die Interpretation des BPMN-Diagramms zu 5. Anschließend wird der Java-Sourcecode erzeugt
Fehlerfälle	1.a.1 Der Pfad ist nicht korrekt: Das System zeigt eine Message und fordert zur Eingabe des korrekten Pfades auf. 1.a.2 Das Format ist nicht korrekt: Das System zeigt eine Message und fordert zur Eingabe des korrekten Pfades zum richtigen Format auf.
Bemerkung	—

Tabelle 3.1.: Java-Sourcecode generieren

Titel	BPMN-Diagramm parsen und interpretieren
Akteur	Anwender
Ziel	Mit der Unterstützung eines Modellierungstools wird ein BPMN-Diagramm erstellt. Das erstellte BPMN-Diagramm wird vom Tool z.B. in XML-Format exportiert (Output). Das B2JSG-System verwendet dieses Diagramm als Input. Eine zentrale Rolle des B2JSG-Systems ist das Parsen und Interpretieren der BPMN-Spezifikationselemente aus dem BPMN-Diagramms, um daraus den konkreten Java-Sourcecode abzubilden.
Auslöser	Der Anwender löst die Java-Sourcecode-Generierung aus und somit ebenfalls das Parsen des BPMN-Diagramms und die damit verbundene Interpretation.
Vorbedingungen	Ein BPMN-Diagramm muss als Input vorhanden sein
Nachbedingungen	Es steht eine Interpretation des BPMN-Diagramms zur Verfügung
Erfolgsszenario	1. Der Anwender ruft das B2JSG-System auf 2. Er navigiert zum Pfad des BPMN-Diagramms 3. Danach führt er die Funktion generiere Javacode aus 4. Das BPMN-Diagramm wird als Input geladen 5. Das BPMN-Diagramm wird geparst 6. Die geparsten BPMN-Spezifikationselemente werden interpretiert, d.h. die Elemente werden in Java-Sourcecode abgebildet
Fehlerfälle	1.b.1 Der Pfad ist nicht korrekt: Das System zeigt eine Message und fordert zur Eingabe des korrekten Pfades auf. 1.b.2 Das Format ist nicht korrekt: Das System zeigt eine Message und fordert zur Eingabe des korrekten Formats auf.
Bemerkung	—

Tabelle 3.2.: BPMN-Diagramm parsen und interpretieren

Titel	BPMN-Diagramm-Output laden
Akteur	Anwender
Ziel	Laden
Auslöser	Der Anwender löst die Java-Sourcecode-Generierung und somit ebenfalls das Laden BPMN-Diagramm-Output aus.
Vorbedingungen	BPMN-Diagramm-Output (Modell) muss vorhanden sein
Nachbedingungen	Das BPMN-Diagramm wurde ins System (B2JSG) geladen und steht zur Weiterverarbeitung bereit
Erfolgsszenario	1. Anwender ruft das B2JSG-System auf 2. Er navigiert zum Pfad des BPMN-Diagramms 3. Danach führt er die Funktion generiere Javacode aus 4. Die Daten werden vom BPMN-Diagramm-Output geladen
Fehlerfälle	1.c.1 1 Der Pfad ist nicht korrekt: Das System zeigt eine Message und fordert zur Eingabe des korrekten Pfades auf. 1.c.2 Das Format ist nicht korrekt: Das System zeigt eine Message und fordert zur Eingabe des korrekten Formats auf.
Bemerkung	—

Tabelle 3.3.: BPMN-Diagramm-Output laden

Titel	Java-Sourcecode ansehen
Akteur	Anwender
Ziel	Der Anwender möchte den aus dem BPMN-Diagramm erzeugten Java-Sourcecode ansehen
Auslöser	Der Anwender löst die Funktion Java-Sourcecode ansehen per Knopfdruck aus
Vorbedingungen	Java-Sourcecode liegt bereits vor oder Java-Sourcecode erzeugen wurde bereits vorher ausgeführt
Nachbedingungen	Der Java-Sourcecode wird angezeigt
Erfolgsszenario	1. Der Anwender ruft per Knopfdruck die Funktion (Java-Sourcecode ansehen) auf 2 Der Java-Sourcecode kann jetzt angesehen und bearbeitet werden
Fehlerfälle	1.d.1 Der Java-Sourcecode wurde nicht erzeugt: Das System zeigt eine Message und fordert zur Generierung des Java-Sourcecodes auf.
Bemerkung	—

Tabelle 3.4.: Java-Sourcecode ansehen

Titel	Java-Sourcecode ins Projekt importieren
Akteur	Anwender
Ziel	Den aus dem BPMN-Diagramm erzeugten Java-Sourcecode in eine Entwicklungsumgebung (z.B. Eclipse oder Netbeans) importieren
Auslöser	Der Anwender löst die Funktion Java-Sourcecode importieren per Knopfdruck aus.
Vorbedingungen	Java-Sourcecode liegt bereits vor oder die Erzeugung des Java- Sourcecodes wurde bereits vorher ausgeführt
Nachbedingungen	Der Java-Sourcecode wird in die Entwicklungsumgebung importiert
Erfolgsszenario	1. Der Anwender ruft per Knopfdruck die Funktion (Java- Sourcecode importieren) auf 2. Der Java-Sourcecode wird in die Entwicklungsumgebung importiert.
Fehlerfälle	1.e.1 Java-Sourcecode wurde nicht erzeugt: Das System zeigt eine Message und fordert zur Generierung des Java-Sourcecodes auf.
Bemerkung	—

Tabelle 3.5.: Java-Sourcecode ins Projekt importieren

4. Marktanalyse und existierende Ansätze

In diesem Kapitel werden eine Marktanalyse und die Analyse der existierenden Ansätze erstellt. Die Marktanalyse beinhaltet Produkte, die auf dem Markt verfügbar sind. Die Recherche existierender Ansätze beschränkt sich auf Forschungen und Publikationen aus Fachquellen. Zur detaillierten Beschreibung der Unterschiede und des Ablaufs der Vorgehensweise einer Marktanalyse und Analyse der existierenden Ansätze sei auf die Literatur [Hosieny (2011)] verwiesen. In dieser Masterarbeit wird für BPMN-Tools lediglich eine Marktanalyse durchgeführt. Die Analyse existierender Ansätze ist hierbei nicht notwendig, da das Hauptziel nicht die Entwicklung eines BPMN-Modeler ist, sondern die Entwicklung eines Java-Codegenerators ist. Infolgedessen wird zuerst eine Marktanalyse für BPMN-Modellierungstools, die den Input für den Codegenerator liefern, durchgeführt. Anschließend folgt die Marktanalyse und die Analyse existierender Ansätze für Codegeneratoren.

4.1. Marktanalyse der BPMN-Modellierungstools

In diesem Abschnitt wird nach verfügbaren BPMN-Modellierungstools recherchiert. Dabei sollen kostenlose Systeme, u.a. ebenfalls Open-Source-Systeme, analysiert werden, die sich auf dem Markt etabliert haben. Wie bereits mehrfach angemerkt, ist das Hauptziel dieser Masterarbeit die Architektur und Realisierung eines automatisierten Java-Codegenerators, basierend auf BPMN 2.0. Daher steht die Codegenerierung im Mittelpunkt. Dennoch ist ein BPMN-Modellierungstool ein wichtiger Bestandteil dieser Masterarbeit, da das zu entwickelnde System (B2JSG) mit dem Modellierungstool eng kooperiert. Das Modellierungstool liefert das BPMN-Diagramm (Modell), worauf diese Arbeit aufbaut. Im Vordergrund stehen hierbei die aufgestellten Anforderungen, die in Kapitel 3 dargestellt werden.

In Anlehnung an Thomas Allweyer [Allweyer (2008)], wird von Bartels [Giso Bartels (2007)] eine Fallstudie über eine Vielzahl an BPMN-Modellierwerkzeugen durchgeführt. Zu Beginn der Marktanalyse ist der Verfasser dieser Masterarbeit auf diese Fallstudie gestoßen. Am Anfang gestaltete sich die Fallstudie als sehr hilfreich, zu einem späteren Zeitpunkt

erwies sich diese für die Masterarbeit jedoch als unbrauchbar. Es wurde deutlich, dass einige Produkte detaillierter beschrieben werden und andere Produkte, insbesondere die Open-Source-Produkte, dagegen nur unzulänglich beschrieben werden. Somit waren die Informationen bezüglich der Open-Source-Produkte nicht ausreichend. Exemplarisch sei der „BPMN modeler“ erwähnt; er wurde sehr kurz beschrieben und ein wichtiger Aspekt, dass dieses Tool nicht ausgereift ist, wurde nicht erwähnt. Das hat zur Folge, dass dieses Tool für den Zweck dieser Masterarbeit derzeit nicht verwendet werden kann.

Einige der nachfolgenden Tools wurden zudem in verschiedenen Eclipse- Entwicklungsumgebungen (z.B. Galileo, Ganymede und Helios) mit unterschiedlichen Versionen getestet. Dabei existieren andere Tools, die unabhängig von der Entwicklungsumgebung arbeiten, bzw. werden sie in einer Entwicklungsumgebung eingebettet mitgeliefert:

4.1.1. Intalio - BPMS Designer

Der BPMS-Designer [BPMS-Designer] ist ein Business Process Management System (BPMS) Designer, der sich für die Modellierung von Business-Prozessen eignet. In diesem Tool bietet er unter anderem die Unterstützung für Business-Analysten, Software-Ingenieure und Systemadministratoren bei der Modellierung von Business-Prozessen und eine anschließende Verwendung auf dem Intalio BPMS-Server. Der Intalio BPMS-Server ist eine Implementierung des neuen BPEL 2.0 Standards, der die Unterstützung für verteilte Transaktionen und menschliche Workflows durch Standard-Erweiterungen bereitstellt. BPMS Designer ist auf der populären Eclipse-Plattform mit einer Ansammlung von Eclipse-Plugins aufgesetzt. Er ist mit vielen Betriebssystemen (z.B. Linux, Mac OS X und Microsoft Windows), die die Eclipse-Workbench unterstützen, kompatibel.

Die aktuelle Distribution des Intalio 6.0.3 umfasst eine Vielzahl an Plugins und liefert den vollen Eclipse SDK 3.4.2 Umfang mit. Es besteht keine Notwendigkeit, Eclipse separat zu installieren. Erstellte BPMN-Diagramme können in PDF, PNG oder SVG exportiert werden. Das SVG-Format beschreibt zweidimensionale Vektorgraphiken und gehört der Sprachfamilie von XML [Eisenberg (2002)] an.

Der Autor hat zur Modellierung von BPMN-Diagrammen den BPMS Designer mit der Version 6.0.3 ausprobiert. Der BPMS Designer erwies sich als sehr stabil, jedoch wird die BPMN-Spezifikation 2.0 noch nicht unterstützt. Die Auslieferung des neuen den BPMN 2.0 unterstützenden Editors sollte im Herbst 2010 erfolgen. Noch ist die Auslieferung jedoch nicht erfolgt. Abbildung 4.1 zeigt die Architektur des BPMS Designers und die umfangreichen Services. Für diese Arbeit wäre der rot markierte Bereich zur Modellierung von BPMN-Diagrammen von Relevanz.

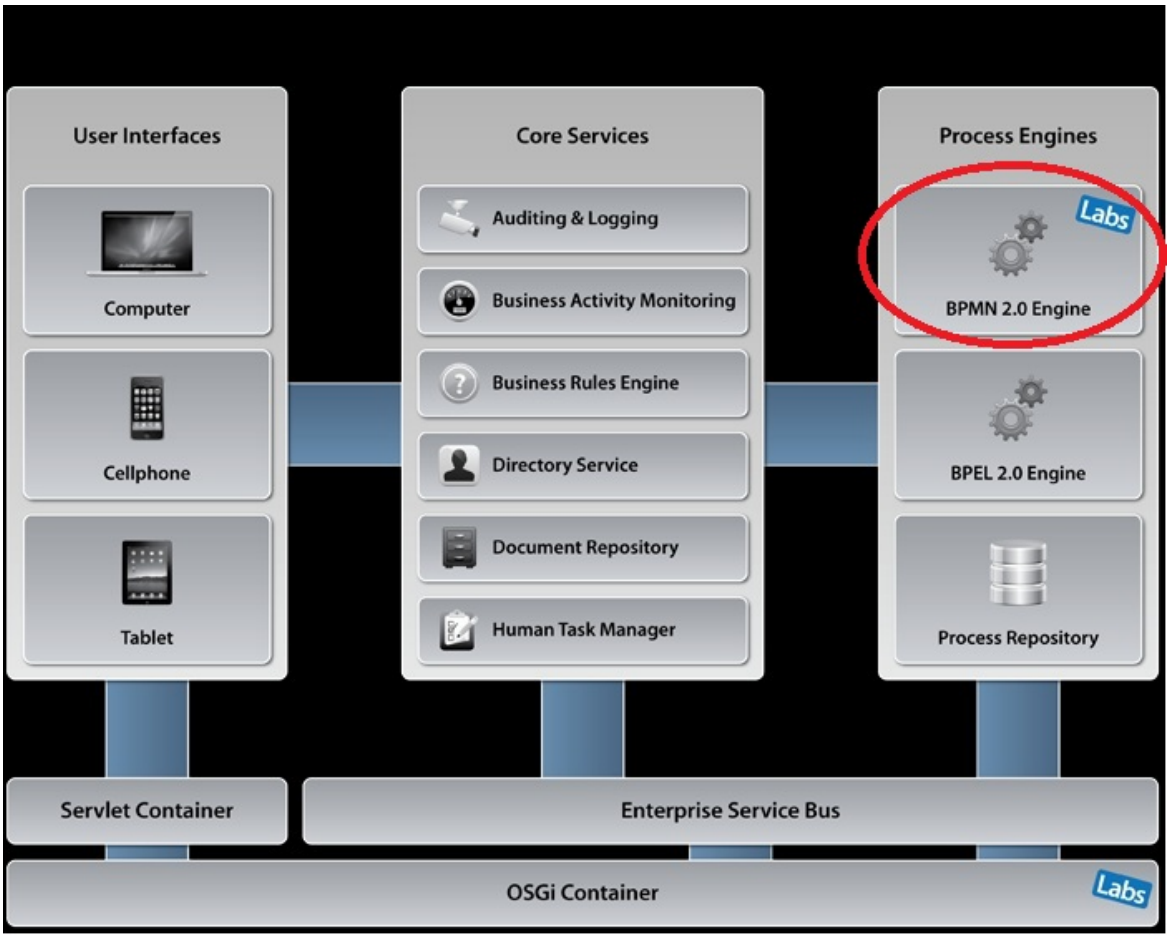


Abbildung 4.1.: Intalio - BPM Architektur

4.1.2. STP BPMN Modeler

Es existiert eine wachsende Gemeinschaft der Verwender eines BPMN-Werkzeugs. Allerdings war anfangs kein BPMN-Modellierungstool in Eclipse vorhanden. Dabei ist BPMN einer der wichtigsten Standards, die jedes gute Business Prozess Modellig System (BPMS) unterstützen sollte. Vor diesem Hintergrund hat das Intalio-Designer-Team beschlossen, seinen BPMN Modeler mit der gesamten Codebasis (als Eclipse-Plugin verpackt) der Eclipse Foundation im Rahmen des STP-Projekts [STP] - SOA-Tool-Plattform zu spenden.

Der BPMN Modeler nutzt die gesamte Funktionalität des Eclipse Graphical Modeling Frameworks (GMF). Das GMF-Framework verfolgt den modellgetriebenen Ansatz (MDA) zur Erzeugung grafischer Editoren unter Verwendung von Eclipse Modeling Framework (EMF) und Graphical Editing Framework(GEF). GMF bewerkstelligt die Integration von EMF und GEF [Eclipse-GMF (2005)], wobei EMF ein Java-Framework und ein Tool zur Code-Generierung basierend auf einem strukturierten Datenmodell ist [Eclipse-EMF (2005)]. Die GEF ermöglicht die grafische Darstellung von Modellen; seine Architektur ist durch das Konzept des Model-View-Controller (MVC) geprägt [Eclipse-GEF (2005)] [Ismael Ghalimi (2006)].

Der STP BPMN Modeler bietet dem Modellierer u.a. folgende Möglichkeiten der Nutzung und potentieller Erweiterungen an:

- Erstellen von BPMN-Diagrammen zur Dokumentation von Prozessen oder Workflows.
- Erweiterung eines Editors per Drag & Drop zur Unterstützung anderer anwendungsspezifischen Nutzungsmöglichkeiten.
- Implementieren einer bestimmte Version der BPMN-Spezifikation.
- BPMN-Diagramme in JPG, PNG, SVG und EMF Formate exportieren.

Der STP BPMN Modeler steht für jede Eclipse-Entwicklungsumgebung zum Herunterladen bereit. Die Erfahrung des Autors mit diesem Tool hat gezeigt, dass das System noch nicht ausgereift ist. Bei der Modellierung trivialer Prozesse kam es häufiger zu Laufzeitfehlern, die zum Systemabsturz führten. Das hatte zur Folge, dass die bereits erstellen Diagramm später entweder nicht wiederhergestellt werden konnten oder dass die Entwicklungsumgebung nicht wiedergestartet werden konnte. Erst nach dem Löschen der Metadaten im Workspace (.metadata) konnte Eclipse hochgefahren werden. Unter Verwendung verschiedener nachstehend aufgeführter Eclipse-Entwicklungsumgebungen [Eclipse (2010)] erwies sich das Tool als instabil.

- Eclipse Helios Version 3.6
- Eclipse Galileo Version 3.5
- Ganymede Version 3.4.2

4.1.3. ARIS Express

ARIS Express ist ein kostenloses Modellierungs-Tool für die Modellierung, die Analyse und das Management von Geschäftsprozessen. Entwickelt wurde das Tool von der IDS Scheer Software AG.

ARIS Express basiert auf Java und setzt eine Java 1.6.10 Version bzw. eine höhere Version voraus. Zudem beinhaltet das Tool viele Features der ARIS Plattformprodukte, solche wie ARIS Business Architect und ARIS Business Designer. Für die Nutzung von ARIS wird ein gültiges Konto bei der ARIS Community vorausgesetzt. Dieses Tool unterstützt verschiedene Modellierungsnotationen, wie zum Beispiel BPMN 2.0, ereignisgesteuerte Prozessketten (Event-driven Process Chain, Abk.: EPC) [Anne Gross (2009)], Organigramme, etc.

Das erste Release wurde im Juli 2009 für registrierte Beta-Tester der ARIS Community publiziert. Später folgte ein weiterer Beta-Test, bis das offizielle Release der Version 1.0 im September 2009 verfügbar war. Die wesentlichen Veränderungen und eine Umstellung auf BPMN 2.0 der vergangenen Versionen wurde erst bei der ARIS Express 2.0 beobachtet. Derzeit steht ARIS Express 2.2 für die Betriebssysteme Windows, Linux und Mac OS X zur Verfügung. ARIS Express kann Diagramme in unterschiedliche Formate exportieren (JPEG, PNG, PDF, EMF, ADF). ADF ist das Dateiformat von ARIS Express. Diagramme in ADF-Format können problemlos wieder in ARIS Express importiert werden [ARIS-Community (2010)].

ARIS Express 2.2 eignet sich für die Modellierung von BPMN 2.0 sehr gut. Durch das ADF-Format kann der alte Zustand eines Projekts wiederhergestellt werden. Es sollte als selbstverständlich angesehen werden, dass ein abgespeichertes Diagramm erneut aufgerufen werden kann. Bedauerlicherweise ist dies nicht immer der Fall. Diese Erkenntnis wurde durch das Testen der Tools gewonnen. Als Beispiel sei der BPMN Modeler zu nennen 4.1.2. Das ARIS Express Modellierungstool ist stabil.

4.1.4. BIZAGI

Das britische Unternehmen BizAgi, als Tochtergesellschaft der Vision Software, hat sich seit seiner Gründung im Jahre 1989 auf die Entwicklung von Technologien und Methoden konzentriert, die sich mit der Verbesserung im Bereich der Geschäftsprozesse beschäftigen. Der BizAgi Modeler ist kostenlos und vollständig nutzbar. Das Projekt ist aber nicht mit einer Open-Source-Lizenz versehen [Jakob Freund (2008)]. Das Resultat von über 20 Jahren Entwicklung sind unter anderem zwei kostenlos verfügbare Produkte:

BizAgi BPMN Process Modeler Version 1.5.1
Der Modeler kann auf Basis von BPMN zur Modellierung von BPMN-Diagrammen und der Dokumentation des Prozesses genutzt werden. Er lässt sich sehr einfach auf der Webseite von BizAgi herunterladen und funktioniert uneingeschränkt ohne Anlegen eines Benutzerkontos auf der Webseite von BizAgi. Der BizAgi Modeler wurde mit Microsoft .NET entwickelt. Für die Betriebnahme, werden lediglich Windowsbetriebssysteme vorausgesetzt.

BizAgi BPM suite Xpress Edition Version 9.1.4
Der BizAgi BPM suite Xpress ist ebenfalls kostenlos verfügbar. Dieses Tool bietet das Exportieren und Ausführen von modellierten Prozessen in der BizAgi BPM suite. Anwender werden mit Hilfe eines Assistenten durch alle notwendigen Schritte geführt, um den Prozess zu automatisieren und ihn in eine laufende Anwendung (Workflow) zu schalten. BizAgi und Instituto de Crédito Oficial (ICO) erhielten für ihre weltweit repräsentative BPM-Lösungen den europäischen Gold Award 2010, von Future Strategies Inc verliehen [BizAgi+ICO (2010)]. Eins der relevanten Merkmale dieses Modellierungstools sind die unterschieden Exportierungsmöglichkeiten der BPMN-Diagramme. BizAgi bietet für das Exportieren des BPMN-Diagramms die Formate Word, PDF, Visio und insbesondere XPDL an. Zudem erweist sich die Software als stabil [BizAgi (2010)] [Limited (2006)].

4.1.5. Model Development Tools (MDT)

Das Modell Development Tools (MDT) Projekt ist eines der vielen Modellierungsprojekte von Eclipse. Ein MDT-Teilprojekt beschäftigt sich mit der Entwicklung eines Metamodels basierend auf der OMG Spezifikation der BPMN 2.0. Mit diesem Projekt werden zwei Ziele verfolgt:

- Umsetzung und Implementierung von Industriestandards der Metamodelle.
- Entwicklung beispielhafter Tools für die Entwicklung von Modellen, basiert auf deren Metamodellen.

Das Projekt beschäftigt sich mit der Entwicklung eines auf BPMN 2.0 basierenden Modellierungstools. Dieses Projekt befindet sich derzeit in der Entwicklungsphase. Mit einem „release“ ist Ende 2011 zu rechnen [Eclipse-MDT] [Hille-Doering (2010)].

4.1.6. Bewertung der BPMN-Modellierungstools

Alle Tools, deren Verwendung für den Codegenerator in Frage kämen, werden hier zusammengefasst dargestellt. Sie wurden nach folgenden Kriterien bewertet:

- **Dokumentation** - Es sollen qualitative und quantitative Dokumentationen (z.B. Reference Manual), sowie Beispiele und Tuturials etc. für das Tool verfügbar sein.
- **Stabilität** - Das Tool darf weder „abstürzen“ noch dürfen andere unerwartete Fehler auftreten.
- **BPMN 2.0 Unterstützung** - Das Tool muss die BPMN 2.0 Spezifikation unterstützen.
- **Entwicklungsstadium** - Es ist in der Regel vorteilhaft, im Gegensatz zu einem Prototypen oder einem Tool, das die ersten Release-Versionen herausgebracht hat, auf ein ausgereiftes, bewährtes Tool zurückzugreifen.
- **Output** - Der Output (BPMN-Diagramm) muss in XML- oder XMI-Format vorliegen.
- **Bedienbarkeit** - Für den Anwender sollte das Tool verständlich und leicht bedienbar sein.
- **Grafische Modellierung** - Das Modellierungstool bietet ein grafisches Modellierungswerkzeug für die Erstellung des Modells an.

[Starke (2009)]

Der Autor hat noch weitere Tools untersucht, jedoch konnten diese nicht in die engere Wahl gezogen werden. Aufgrund dessen werden an dieser Stelle lediglich die fünf vorgestellten BPMN-Tools in der Tabelle 4.1 bewertet. Für den Autor stellen der „Output“ und die „BPMN 2.0 Unterstützung“ Ausschlusskriterien dar.

Wie aus der Tabelle ersichtlich, erfüllt BIZAGI als einziges Tool alle Kriterien.

4.2. Marktanalyse für BPMN-Codegeneratoren

Das Hauptziel dieser Masterarbeit besteht darin, einen BPMN-Sourcecode-Generator (B2JSG) zu entwickeln, der die Anforderungen, die im letzten Kapitel dargestellt wurden, erfüllt. Bevor mit der Konzeption und Realisierung des B2JSG begonnen wird, soll eine Marktanalyse vorhandener Softwaresysteme durchgeführt werden. Damit möchte der Autor

Kriterien/BPMN-Tools	Intalio	STP Modeler	ARIS Express	BIZAGI	MDT
Dokumentation	+	+-	++	++	+
Stabilität	++	- -	++	++	-
BPMN 2.0 Unterstützung	-	-	++	++	-
Entwicklungsstadium	+-	-	+	++	- -
Output	+	+	- -	+	-
Bedienbarkeit	+-	+-	+	++	-
Grafische Modellierung	++	++	++	++	++

Tabelle 4.1.: Modellierungstool Evaluation

sicherstellen, ob eine Neuentwicklung notwendig und förderlich ist. Bei vorhandenen Systemen soll überprüft werden, ob sie die Anforderungen in allen Punkten erfüllen.

Auf dem Markt haben sich einige grafische Modellierungs-Plattformen etabliert. Als Beispiel seien „Together“ vom Unternehmen Borland [Borland (2011)] und „Java Developer“ der Firma „Peak Engineering“ [Engineering (2011)] genannt. Diese grafischen Modellierungs-Plattformen ermöglichen das Designen von Modellen und die anschließende Java-Sourcecode Generierung. Mit dieser Thematik beschäftigen sich kleine bis große Unternehmen. Die Recherche hat allerdings ergeben, dass kein Tool existiert, das aus dem BPMN 2.0 Diagramm (Modell) Java-Sourcecode generiert. Infolgedessen wird nachfolgend direkt auf die existierenden Ansätze eingegangen.

4.3. Existierende Ansätze für BPMN-Codegeneratoren

In diesem Abschnitt geht der Autor auf einige existierende Ansätze und Techniken zur Codegenerierung ein. Diese Techniken werden in einigen weitverbreiteten MDA-Tools zur Codegenerierung eingesetzt. Die detaillierte Auflistung aller Codegenerierungskonzepte würde den Rahmen dieser Masterarbeit sprengen. Demzufolge beschränkt sich der Autor auf die wesentlichen Konzepte.

4.3.1. Templates und Filtering

Die Abbildung 4.2 zeigt eine Codegenerierungstechnik, die als Templates und Filtering bekannt ist. Mittels Templates und Filtering kann auf einfachste Weise laut [Roland Petrasch (2006)] Code generiert werden. Es wird kein Metamodell, wie in den MDA-Ansätzen beschrieben, verwendet.

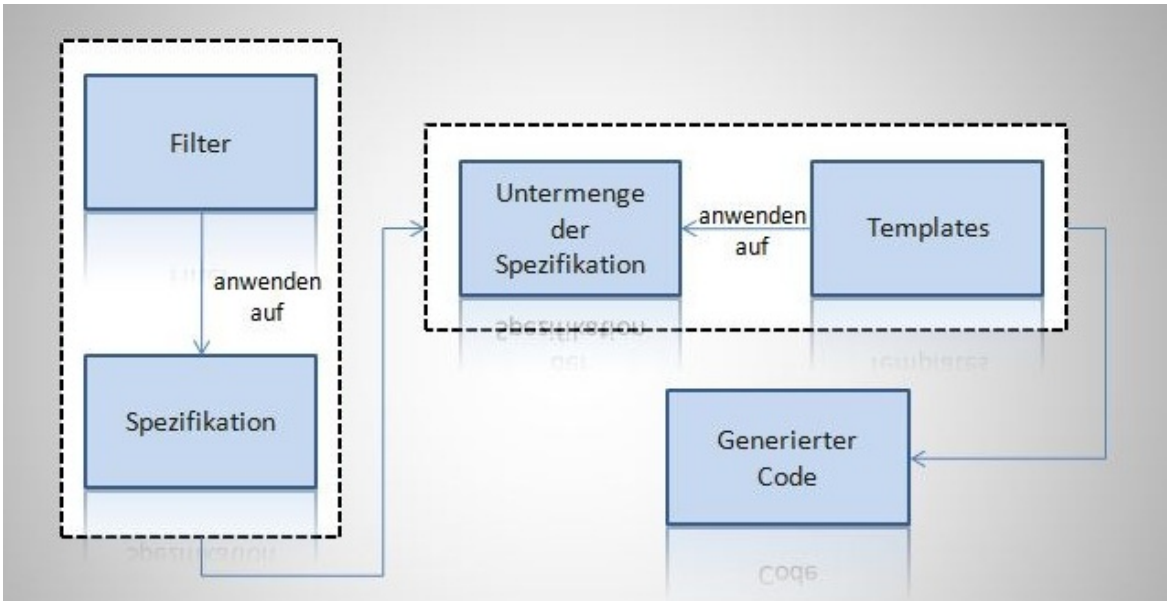


Abbildung 4.2.: Templates und Filtering

Die Templates führen mit Hilfe der XSLT-Sprache Iterationen über das Modell, das in XML-Format vorliegt, durch. Die XSLT-Sprache ist eine Programmiersprache zur Transformation von XML-Dokumenten [Tidwell (2002)]. Zusätzlich beinhalten die Templates den zu generierenden Code (XSLT-Stylesheet). Die Transformationsregeln von XSLT sind im sogenannten XSLT-Stylesheet verfasst, das das XML-Dokument direkt in ein anderes Stylesheet transferieren lässt. Der Nachteil dieser Stylesheets besteht darin, dass sie bei größeren Systemen schnell unübersichtlich werden. Als ein weiterer Nachteil bei diesem Verfahren ist anzuführen, dass grundsätzlich auf der Abstraktionsebene des „XML-Metamodells“ gearbeitet wird.

4.3.2. Templates und Metamodell

Dieses Verfahren beruht auf einer mehrstufige Generierung. Wie Abbildung 4.3 zeigt, wird das Modell, das in XML- oder XMI-Format vorliegt und auf einem Metamodell basiert, geparst. Anschließend wird eine Instanz des Metamodells gebildet. Das Metamodell wurde

bereits im Abschnitt 2.2 erläutert. Hierbei stellt das Metamodell die Rolle der abstrakten Syntax dar. Das Modell und das Metamodell werden miteinander verknüpft und werden mit den Templates zusammen zur Codegenerierung verwendet.

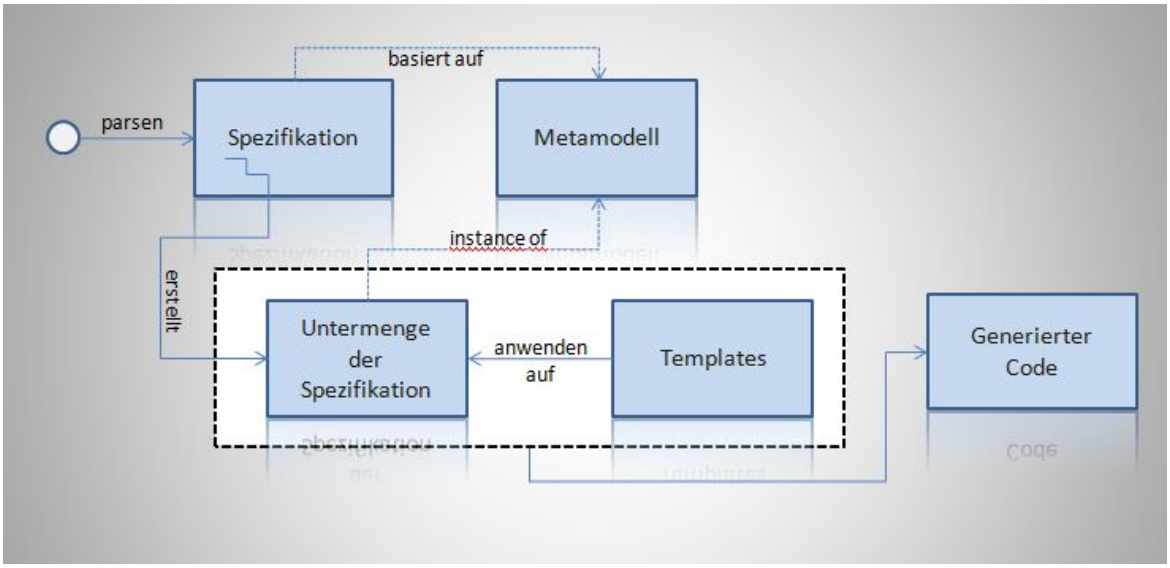


Abbildung 4.3.: Templates und Metamodell

In der zuvor dargestellten Technik „Template und Filtering“ wurde die Verifikation des Modells, falls vorhanden, in die Templates verlagert. Der besondere Vorteil dieser Technik liegt darin, dass die Verifikation in das Metamodell ausgelagert werden kann. Somit wird eine Trennung der Codegenerierungslogik und der Logik für die Verifikation des Modells (konkrete Ausprägung in XML bzw. XMI) erreicht. Die Verifikation des Modells kann in einer von den Templates unabhängigen Programmiersprache modelliert werden. Die Technik der „Templates und Metamodell“ ist ein von MDA verfolgter Ansatz und daher in diesem Kontext von hoher Relevanz.

OpenArchitectureWare (oAW)

oAW ist eine auf „Templates und Metamodell“ basierende Plattform für die modellgetriebene Softwareentwicklung (engl. MDSD). Sie ist eine in Java implementierte Open-Source verfügbare Tool-Suite, mit der aktuellen Version 4.3.1. Der Kern der oAW-Plattform stammt aus der kommerziellen Tool-Suite b+m Generator Framework der b+m Informatik AG. Zu einem späteren Zeitpunkt wurde die oAW-Plattform als ein Open Source Projekt auf SourceForge.net unter der LGPL Lizenz publiziert. Sie erregte in kürzester Zeit große Aufmerksamkeit in der Modellierungsgemeinde. Das Projekt der OpenArchitectureWare ist sehr gut dokumentiert, und es werden auch viele praktische Bespiele auf der Seite der [oAW (2011)] bereitgestellt,

die wohl zu deren Erfolg und dem hohen Bekanntheitsgrad geführt haben dürften. Um dies nochmals zu unterstreichen, versteht sich oAW nicht als MDA-Generator, sondern als eine Plattform für MDSD-Projekte. Es ist zum Generieren von Sourcecode erforderlich, dass zu einem beliebigen Modell ein Metamodell und die entsprechenden Templates vorhanden sind. Mittlerweile wurde unter anderem durch die starke Eclipse-Integration ab der Version vier der Code des oAW-Kerns nach Eclipse portiert.

Technisch gesehen ist die Integration dieses Frameworks in andere Entwicklungsumgebungen mit geringem Aufwand verbunden. Die Codegenerierung wird durch ein Ant-Skript initiiert, wodurch die Eingabemodelle auf Fehler überprüft und dann mit der Unterstützung der Tempates in den Sourcecode überführt werden. Die Templates werden mittels der Template-Sprache Xpand definiert und werden in Dateien mit dem Präfix „xpt“ persistiert (z.B. Template.xpt). Sie beinhalten Befehle der Xpand-Sprache und den zu generierenden Text. Die Xpand-Befehle unterscheiden sich vom Text dadurch, dass sie in spitze Klammern («IF», Guillemets) eingeschlossen werden. Die allgemeine Struktur der Template-Dateien wird am folgenden Beispiel 4.3.2 verdeutlicht. An dieser Stelle wird dem Leser eine kurze Einführung über die Expand-Sprache gewährt. Für eine detaillierte Einführung wird auf die folgenden Quellen verwiesen [oAW (2011)] [Sven Efftinge (2006)], aus denen die Beispiele stammen.

Listing 4.1: Allgemeine Struktur von Template-Dateien

```
«IMPORT meta::model»
«EXTENSION my::ExtensionFile»

«DEFINE javaClass FOR Entity»
        «FILE fileName()»
        package «javaPackage()»;
        «EXPAND TemplateFile::definitionName FOR myModelElement»
        public class «name» {
                // implementation
        }
        «ENDFILE»
«ENDDEFINE»
```

Eine Template-Datei besteht aus einer beliebigen Anzahl von IMPORT-Anweisungen, gefolgt von einer beliebigen Anzahl von EXTENSION-Anweisungen und mindestens eines DEFINE-Blocks.

IMPORT

Listing 4.2: Xpand - IMPORT

```
«IMPORT meta::model»
```

Dieses IMPORT ähnelt der Import-Anweisung in Java (import meta.model.*). Unter Verwendung von IMPORT können das Metamodel und Java-Funktionalitäten eingebunden werden.

Extension

Listing 4.3: Xpand - EXTENSION

```
«EXTENSION my::ExtensionFile»
```

Extensions bieten eine flexible und komfortable Möglichkeit, den Funktionsumfang der Xpand-Sprache mittels Einbinden von Java zu erweitern. Infolgedessen kann der gesamte Funktionsumfang von Java über die Extensions eingesetzt werden.

DEFINE

Das zentrale Konzept von Xpand sind die Define-Blöcke (ebenfalls als Templates bekannt). Ein DEFINE-Block repräsentiert einen Definitionsbereich für ein Element aus dem XMI-Modell. In diesem Bereich werden die konkreten Sourcecode (z.B. Java) Abbildungen definiert. DEFINE-Blöcke können verschachtelt verwendet werden. Die Verschachtelung wird mit Hilfe des XPAND-Befehls durchgeführt. Der XPAND-Befehl wird ausgeführt, um andere DEFINE-Blöcke aufzurufen.

LET

Listing 4.4: Xpand - LET

```
«LET expression AS variableName»
a sequence of statements
«ENDLET»
```

LET dient zur Definition von lokalen Variablen. Während der Ausführung des LET-Blocks wird der Wert von expression an die Variable variableName gebunden. Die Variable kann innerhalb dieses LET-Blocks verwendet werden.

FOREACH

Listing 4.5: Xpand - FOREACH

```
«FOREACH expression AS variableName [ITERATOR iterName] [SEPARATOR
expression]»
a sequence of statements using variableName to access the
current element of the iteration
«ENDFOREACH»
```

Äquivalent zu Java ist die Iteration über Listen möglich. Expression repräsentiert eine Liste. Während der Iteration werden die Elemente von expression an die Variable variableName gebunden. Im FOREACH-Block kann über die Variable variableName auf das aktuelle Element der Liste zugegriffen werden.

Kommentare

Listing 4.6: Xpand - Kommentare

```
«REM» Kommentar «ENDREM»
```

In Templates können Kommentare mit Hilfe von REM-Tags realisiert werden.

[Matthias Regensburger (2007)] [oAW (2011)] [OAW (2008)] [Rumpe (2005)] [Thomas Stahl (2005)] [Roland Petrasch (2006)] [b+m Informatik AG (2011)] [Matteo Bordin (2005)]

4.3.3. Code-Attribute

XDoclet ist ein Open-Source-Projekt, mit dessen Unterstützung Kommentare aus Java-Programmen ausgelesen und in ausführbarem Code übersetzt werden [Reiberg (2006)]. Diese Technik ist unter dem Namen Code-Attribute bekannt. Ursprünglich stammen sie aus dem Javaumfeld mit JavaDoc. Die erweiterbare Architektur von JavaDoc erlaubt das Anhängen von eigenen Tags und Code-Generatoren zur automatischen Generierung von HTML-Dokumentationen. Bei dieser Technik ist ebenfalls kein Metamodell notwendig [Thomas Stahl (2005)].

4.3.4. Cartridge und Metamodell

Eine sehr bekannte Technik ist die Technik der sogenannten Cartridges. Cartridges enthalten Transformationsregeln und Code-Templates. Darüber lassen sich durch Modelle als Eingabe Code für beliebige Plattformen generieren. Um einen praktischen Bezug zu

erstellen, wird auf AndroMDA [AndroMDA (2011)], der diese Technik verfolgt, eingegangen.

AndroMDA (ausgesp. Andromeda) ist ein erweiterbarer Framework Generator. In dieses Werkzeug sind Cartridges integriert, mit Hilfe deren Quellcode generiert werden können. Die nachfolgende Abbildung 4.4 veranschaulicht die Transformationen des AndroMDA.

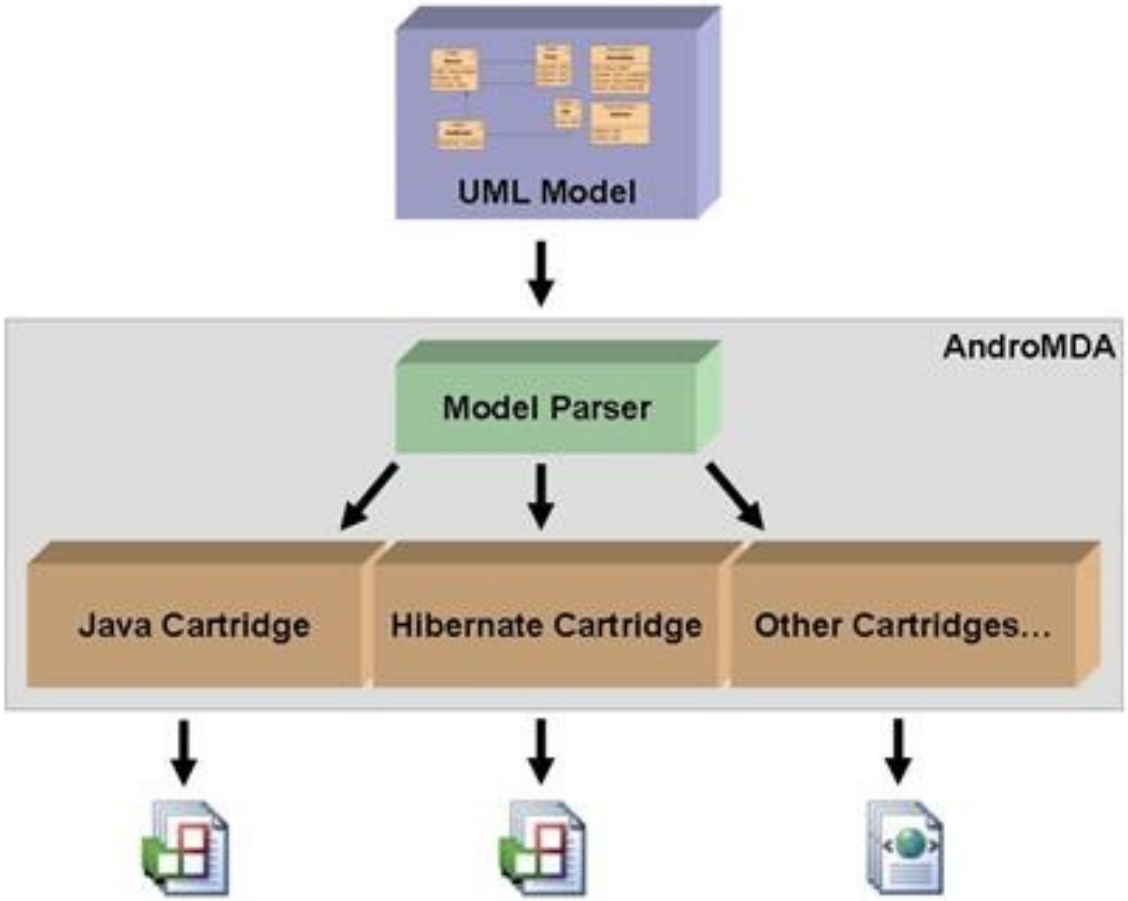


Abbildung 4.4.: Architektur des AndroMDA

AndroMDA knüpft ebenso an das Paradigma der Model Driven Architecture (MDA) an. Er verwandelt UML-Modelle in Code für bestimmte Plattformen. Mit Hilfe von Cartridges definiert AndroMDA Transformationen für Plattformen wie z.B. Spring, JSF, Hibernate, Struts und andere. Ein Cartridge generiert nicht das Gesamtsystem, sondern einzelne Aspekte, zudem können Cartridges auch kombiniert verwendet werden, um unterschiedliche Bereiche generieren zu lassen. Templates ähneln Cartridges. In den Templates sind auch die Abbildungsregeln für die Codegenerierung enthalten, jedoch werden Templates vom Entwickler selber geschrieben. Cartridges sind ein fester vordefinierter Bestandteil der AndroM-

DA Architektur, sie agieren wie Bausteine und können angepluggt werden. AndroMDA hat ein UML-Metamodell und ist somit kompatibel mit anderen Modellierungwerkzeugen, die das selbe UML-Metamodell verwenden (Output: XMI). In der oAW-Plattform können im Gegensatz zu AndroMDA viele Metamodelle verwendet werden; die Voraussetzung dafür ist, dass das Metamodell in ECORE-Format vorliegt. Das Format für Metamodelle in EMF oder Eclipse wird als ECORE bezeichnet [EMF (2011b)]. AndroMDA ist eine Plattform, die viele Möglichkeiten bietet, auch weitere Möglichkeiten der Codegenerierung. Der Autor ist hierbei auf das Wesentliche eingegangen. Nachfolgend sind die Literaturquellen angezeigt, auf die der interessierte Leser zur Vertiefung an dieser Stelle verwiesen wird:

[Breno Lisi Romano und Mourão (2010)] [AndroMDA (2011)] [Rumpe (2005)] [Thomas Stahl (2005)] [Thomas Stahl (2007)] [Roland Petrasch (2006)]

Nachfolgend werden die beiden Plattformen (oAW und AndroMDA) für die Codegenerierung anhand der Qualitätsmerkmale zusammengefasst und bewertet. Bewertet werden lediglich Plattformen mit MDA-Ansätzen, daher werden „Template und Filtering“ und Code-Attribute nicht in der Tabelle aufgeführt. Auf die Qualitätsmerkmale wurde bereits im Abschnitt 4.1.6 eingegangen. Eine fehlende BPMN 2.0 Unterstützung gilt auch hier als Ausschlusskriterium.

Kriterien/MDA-Tools	oAW	AndroMDA
Dokumentation	++	+
Stabilität	++	++
BPMN 2.0 Unterstützung	++	- -
Entwicklungsstadium	++	++
Bedienbarkeit	++	+

Tabelle 4.2.: MDA-Plattform Evaluation

Wie aus der Tabelle ersichtlich, erfüllt AndroMDA nicht das Akzeptanzkriterium. AndroMDA ist ausgerichtet auf UML-Modelle. Ein Hinzufügen von anderen Metamodellen, z.B. einem BPMN-Metamodell, ist nicht möglich. Hinzu kommt, dass die oAW-Plattform bessere Ergebnisse in der Dokumentation und der Bedienbarkeit erzielt.

5. Systemdesign

In diesem Kapitel sollen Designüberlegungen über die Architektur des Systems (B2JSG) angestellt werden. Dabei sind die aufgestellten Anforderungen 3 und die gewonnenen Erkenntnisse über die Modellierungs- und Codegenerierungswerkzeuge 4 nicht zu vernachlässigen. Ebenso sind die Verteilung der Komponenten und die Schnittstellen des Systems Bestandteil dieses Kapitels.

Die oAW-Plattform erwies sich zur Unterstützung der Codegenerierung im Hinblick auf die aufgestellten Bewertungskriterien 4.2 als am besten geeignet. Ein integraler Bestandteil für die Codegenerierung mittels der oAW-Plattform sind das Modell, das Metamodell und die entsprechenden Templates. Unter der Voraussetzung, dass der Autor das BPMN-Diagramm (Modell) von BizAgi 4.1.4, verwendet, sind weiterhin ein Metamodell für BPMN 2.0 und die dazugehörigen Templates erforderlich. Die oAW-Plattform stellt bereits für die Generierung von Java-Sourcecode aus dem „UML-Modell“ Funktionalitäten und Hilfsmittel sowie ein Metamodell und die entsprechenden Templates, zur Verfügung. Die Möglichkeit, aus einem BPMN-Model Java-Sourcecode zu generieren, wird nicht unterstützt. Infolgedessen wird kein BPMN 2.0 Metamodell und die dazugehörigen Templates bereitgestellt. Im Anbetracht dessen sieht der Autor die in Abschnitt 2.2 beschriebene Anwendungsmöglichkeit:

Die Transformation von „BPMN 2.0“ zu Java-Sourcecode (Model-Code Transformation = Translationist) auf direktem Weg (ohne Zwischenschritte).

5.0.5. Der Translationisten Ansatz

Die Transformation von BPMN 2.0 zu Java (Model-Code Transformation) ist realisierbar, allerdings sind ein Metamodell der BPMN 2.0 und die jeweiligen Templates erforderlich. Der Autor stellte die Überlegung an, ob er selbst ein Metamodell für BPMN 2.0 entwickeln soll oder ein bereits existierendes BPMN 2.0 Metamodell dafür verwenden kann. Wie bereits erwähnt, erlaubt die oAW-Plattform das Hinzufügen eigener Metamodelle, solange die Metamodelle als Ecore-Format vorliegen, und die Implementierung eigener Templates mit Hilfe der Templatesprache Xpand und der Objektorientierten Sprache Java durchgeführt wird. Im nächsten Abschnitt wird die Architektur vorgestellt und auf Lösungsmöglichkeiten der beiden

wichtigen Bestandteile (Metamodell und Templates) der Modelle zur Code Transformation eingegangen

5.1. Architektur

In diesem Abschnitt wird zunächst ein Überblick über die Architektur 5.1 des zu entwickelnden Systems (B2JSG) gewährt. Nachfolgend wird die Architektur auf einer hohen Abstraktionsebene zum Zweck der Nachvollziehbarkeit kurz vorgestellt. Anschließend folgt eine detaillierte Erläuterung der Komponenten des zugrunde liegenden Systems.

Wie bereits angemerkt, basiert die Architektur auf den Anforderungen, die in Kapitel 3 vorgestellt wurden. Zudem ist sie von der oAW-Plattform stark geprägt. Das Gesamtsystem besteht aus zwei Teilsystemen: einem externen BPMN 2.0 Modellierungssystem (Externes Modellierungstool) und dem eigentlichen Codegenerator (B2JSG). Die beiden Teilsysteme spiegeln sich in der Architektur wider, wobei der Codegenerator den wesentlichen Kern dieser Arbeit darstellt. Das externe Modellierungssystem stellt den Input für den Codegenerator (B2JSG) bereit. Der Input ist ein BPMN-Diagramm (Modell). Der Codegenerator greift darauf zu, falls notwendig, findet eine Übersetzung des Modells in ein zum verwendeten BPMN-Metamodell kompatibles Modell (XMI) statt. Anschließend kann unter Verwendung des Modells, Metamodells und Templates die Codegenerierung in Java-Sourcecode durchgeführt werden. Als nächstes wird auf die einzelnen Komponenten detaillierter eingegangen.

5.1.1. Komponentenbeschreibungen

Nachfolgend werden alle Komponenten, die für die Architekturerstellung benötigt werden, beschrieben. Die angebotenen und verwendeten Schnittstellen werden in der Architektur des Autors durch den Buchstaben „I“, gefolgt von einem Namen, zur Unterscheidbarkeit der Schnittstellen gekennzeichnet.

5.1.2. Externes Modellierungstool

Ein mögliches Modellierungstool wäre z.B. der im letzten Kapitel vorgestellte „BizAgi“ 4.1.4 Modeler, worauf der B2JSG basiert. BizAgi hat sich als ein geeignetes Modellierungstool herauskristallisiert. Unter Verwendung dieses Tools kann ein BPMN-Diagramm modelliert

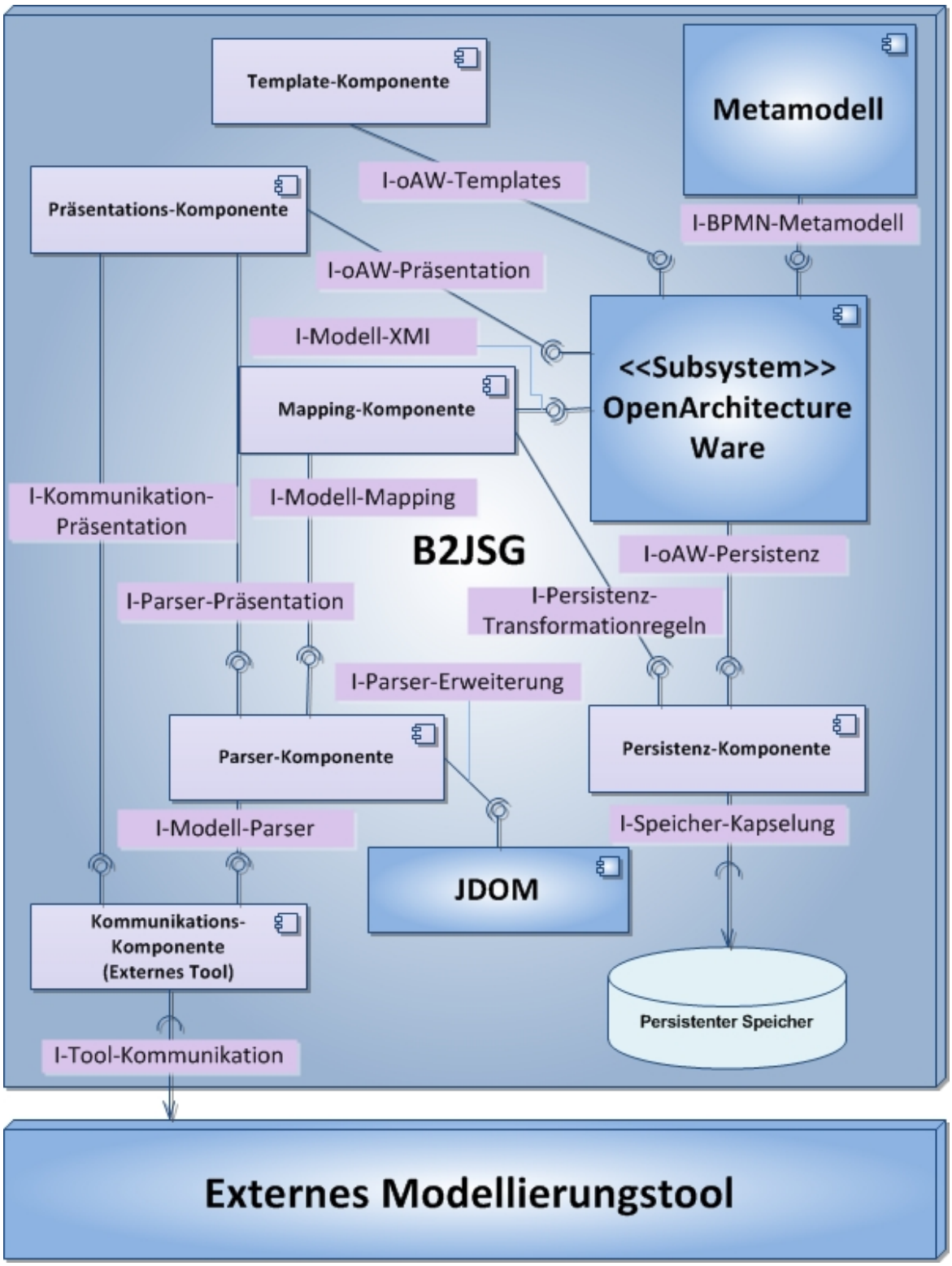


Abbildung 5.1.: Konzept und Architektur

werden. Dieses kommuniziert über das Interface `I-Tool-Kommunikation` mit dem B2JSG. Hierfür kommen ausschließlich Modellierungs-Tools in Frage, die BPMN 2.0 unterstützen und einen brauchbaren Output (Modell) bereitstellen. D.h., dass der Output (Modell) mit geringem Aufwand reproduzierbar sein sollte. Es ist nicht hilfreich, wenn die Tools z.B. JPG als Output liefern. Aus dem Output sollen später Java-Sourcecode mit den entsprechenden Klassen und Abhängigkeiten generiert werden können.

5.1.3. Kommunikations-Komponente

Die Kommunikations-Komponente kapselt die Kommunikation von B2JSG nach außen. Sie definiert und implementiert die Schnittstelle für die Kommunikation zwischen dem B2JSG und dem Modellierungstool über das Interface `I-Tool-Kommunikation`. Sie stellt ebenso den Zugriff der „Parser-Komponente“ auf die Daten des externen Modellierungstools sicher. Über das Interface `I-Kommunikation-Präsentation` der Präsentations-Komponente kann angesteuert werden, dass der B2JSG den Input (Modell) von außen abruft.

5.1.4. Parser-Komponente

Diese Komponente stützt sich über die Schnittstelle `I-Parser-Erweiterung` auf die JDOM-Funktionalitäten und verbindet sie mit den eigenen Funktionen. Sie liest das BPMN-Diagramm (Modell) über die Schnittstelle `I-Modell-Parser` ein und stellt die darin enthaltenen Informationen durch die Schnittstelle `I-Modell-Mapping` der Mapping-Komponente zur Verfügung. Der Parser wird über die Präsentations-Komponente initiiert. Zu diesem Zweck wird die Schnittstelle `I-Parser-Präsentation` verwendet. An dieser Stelle ist anzumerken, dass diese Komponente und die Mapping-Komponente nicht aktiv werden, solange das Modell in dem vom Metamodell verwendeten Format vorliegt.

5.1.5. JDOM

JDOM API 5.2, mit der aktuellen Version 1.1, ist eine Open Source-Software für die Programmiersprache Java. In JDOM werden XML-Dateien als ein DOM-Dokument in Form einer Baumstruktur eingelesen. Auf die Elemente eines Dokuments kann durch die Iteration von oben nach unten zugegriffen werden. Die Zugriffsmethoden von JDOM ermöglichen

dem Entwickler, einfacher auf die Elemente und Unterelemente einer in XML-Format vorliegenden Datei zuzugreifen [JDOM (2011)] [Dieter Eickstädt (2004)]. Anhand der Schnittstelle `I-Parser-Erweiterung` stellt JDOM seinen kompletten Funktionsumfang der Parser-Komponente zur Verfügung.

5.1.6. Mapping-Komponente

Eine zentrale Rolle dieser Komponente ist die Übersetzung des eingelesenen Modells über die von der Parser-Komponente bereitgestellte Schnittstelle `I-Modell-Mapping` in ein XMI konformes Format. Andernfalls würde die oAW-Plattform dieses Format nicht „verstehen“. Die Mapping-Komponente greift außerdem über die Schnittstelle `I-Persistenz-Transformationsregeln` auf die Persistenz-Komponente zu, um die persistenten Transformationsregeln abzurufen. Das Resultat der Transformation wird über die Schnittstelle `I-Modell-XMI` der oAW-Plattform zur Verfügung gestellt. Diese Komponente, abhängig vom Modellierungstool, ist immer austauschbar.

5.1.7. Präsentations-Komponente

Die Präsentations-Komponente stellt die User-Interface (Benutzer-Schnittstelle) dar. Da es sich um einen Java-Codegenerator handelt, erfolgen nicht allzu viele Benutzerinteraktionen, die den Programmfluss stark beeinflussen. Dennoch ist es wichtig, sich auf die in Anforderungen 3 definierten Anwendungsfälle zu stützen. Insofern sollen nach der Aufforderung zur Generierung des Java-Sourcecodes in der Präsentations-Komponente, das Java-Projekt und der Sourcecode einsehbar sein. Die Logik zur Darstellung des gesamten Projektes befindet sich in der Präsentations-Komponente. Sie greift über die Schnittstelle `I-Kommunikations-Präsentation` auf die Kommunikations-Komponente, über `I-Parser-Präsentation` auf die Parser-Komponente und über `I-oAW-Präsentation` auf die oAW-Plattform zu.

5.1.8. Persistenz-Komponente

Diese Komponente wurde eingeführt, um Abhängigkeiten zu dem persistenten Speicher zu kapseln. Die gesamten Zugriffe erfolgen nicht direkt auf den persistenten Speicher, sondern über die Schnittstellen `I-Persistenz-Transformationsregeln` und `I-oAW-Persistenz` der Persistenz-Komponente. Lediglich der Persistenz-Komponente

wird über die Schnittstelle `I-Speicher-Kapselung` der direkte Zugriff auf den persistenten Speicher gewährt. Auf diese Weise wird das Prinzip der losen Koppelung [Sommerville (2007)] verfolgt.

5.1.9. Persistenter Speicher

Der persistente Speicher sorgt für die dauerhafte Speicherung aller benötigten Daten. Es sind beispielsweise die Abbildungsregeln der Mapping-Komponente hinterlegt und ebenso der generierte Java-Sourcecode. Dabei erfolgt die Speicherung der Daten auf dem lokalen Datenträger. Es ist nicht auszuschließen, dass an dieser Stelle ein Datenbankmanagementsystem (DBMS) zum Einsatz kommt. Der Zugriff auf den persistenten Speicher erfolgt nicht direkt, sondern über die Schnittstelle `I-Speicher-Kapselung` der Persistenz-Komponente. Als Speichermedium werden XMI-, XPT und Java-Dateien verwendet. XMI dient der Speicherung der BPMN-Diagramme (Modell), Java-Dateien sind das Ergebnis der Generierung. XPT sind die Templates, die mit der Templatesprache Xpand erstellt werden.

5.1.10. Template-Komponente

Ein fundamentaler Bestandteil der oAW-Plattform sind unter anderem die Templates. Die Templates werden über die Schnittstelle `I-oAW-Templates` der oAW-Plattform bereitgestellt. Nach der Instanziierung des Modells können die Templates auf die Modellelemente angewendet werden. Im Wesentlichen umfassen die Templates und somit die Template-Komponente, den zu generierenden Code zuzüglich der Abbildungsregeln. Ein wichtiger Bestandteil dieser Masterarbeit sind die Templates, da sie den Kern der Abbildung der BPMN 2.0 Spezifikationselemente zur Java-Sourcecode darstellen. Aus diesem Grund wird im nächsten Kapitel 6 auf die Abbildungen der Elemente detaillierter eingegangen.

5.1.11. Metamodell

Wie zuvor mehrfach angemerkt, liefert das Modellierungstool z.B. BizAgi 4.1.4 das BPMN-Diagramm (Modell), basierend auf BPMN 2.0. Was der Verfasser dieser Arbeit zurzeit benötigt, ist das Metamodell. Das Metamodell muss beispielsweise kompatibel zum Modell von BizAgi sein; das Parsen und ein Mapping sind notwendig. BizAgi nutzt ein BPMN 2.0 Metamodell, dieses ist jedoch nicht in ECORE-Format verfügbar. Die oAW-Plattform benötigt

das Metamodell im ECORE-Format.

Daraus ergeben sich zwei Alternativen für den Autor:

- Entwicklung eines eigenen Metamodells in ECORE.
- Verwendung eines beliebigen in ECORE-Format vorliegendes Metamodells, zugleich das Parsen und Mappen.

Die durch den Autor entwickelte Architektur lehnt sich an die Architektur der oAW-Plattform an. Sie ist so konzipiert, dass ein beliebiges Metamodell (ECORE Modell) verwendet werden kann.

Ende 2010 lieferte die SAP ein BPMN-Modellierungstool (SAP Process Composer), das auf der Eclipse-Plattform basiert. Das Metamodell (ECORE-Modell) dieses Tools wurde mittels „Eclipse Modeling Framework“ (EMF) erstellt, da das EMF viele nützliche APIs und Funktionen für die Erstellung von Metamodellen bietet. BPMN 2.0 enthält bereits ein Metamodell. Allerdings hat es das sogenannte CMOF (Common Meta Object Facility)-Format, das nicht direkt in Eclipse verwendet werden kann. Daher hat SAP das BPMN 2.0 Metamodell auf EMF implementiert [EMF (2011a)] [Hille-Doering (2010)]. Aufgrund des von SAP zur Verfügung gestellten Process Composer kann der Autor auf die Entwicklung einen Metamodells verzichten, da die Entwicklung eines komplexen Metamodells sowie das von SAP entwickelte Metamodell den Rahmen dieser Masterarbeit sprengen würde. Obwohl das BPMN-Modellierungstool der SAP noch in der Entwicklungsphase ist, bietet es ein komplexes Metamodell und einen textuellen Modellierungseditor. Lediglich die Templates müssen implementiert werden. Demnach können BPMN-Diagramme, die mit dem Modellierungseditor der SAP erstellt wurden, ohne Mapping (ohne Einsatz der Parser umd Mapping-Komponente) direkt von der oAW-Plattform verwendet werden. Dabei muss allerdings die oAW-Plattform das Metamodell der SAP über die Schnittstelle `I-BPMN-Metamodell` nutzen. Bei Verwendung externer Modellierungstools z.B. BizAgi muss dann eine Übersetzung (Mapping) des BPMN-Diagramms in das von der oAW-Plattform akzeptierten XMI-Format stattfinden.

5.1.12. OpenArchitectureWare (oAW)

Im vorherigen Kapitel in Abschnitt 4.3 wurde bereits auf die Funktionen und Technologien, die sich hinter der oAW-Plattform verbergen, eingegangen. Die oAW ist ein wichtiger Bestandteil der B2JSG Architektur und spielt eine zentrale Rolle bei der Codegenerierung. Sie ist über Plugins in die Eclipse-Plattform integrierbar, ebenso kann sie unabhängig von

der Eclipse-Plattform verwendet werden. Die Schnittstelle zwischen dem Modell, dem Metamodell und den Templates repräsentiert in der oAW-Plattform die XML-Konfigurationsdatei „generator.oaw“.

Nach der Konfiguration der generator.oaw greift die oAW-Plattform über die Schnittstelle `I-Modell-XMI` auf das BPMN-Diagramm (Modell z.B. XMI-Format) zu, das bereits kompatibel zum verwendeten BPMN-Metamodell ist. Das Modell wird mit dem Parser (nicht zu verwechseln mit der Parser-Komponente des B2JSG) der oAW-Plattform ausgelesen. Auf das Metamodell wird über die Schnittstelle `I-BPMN-Metamodell` zugegriffen. Die oAW-Plattform bildet eine vereinigte Instanz (model) aus dem Modell und dem entsprechenden Metamodell. Nach der Instanzierung können über die Schnittstelle `I-oAW-Templates` die Templates auf die Modellelemente angewendet werden. Somit wird der Sourcecode generiert. Das Resultat (der generierte Java-Sourcecode) wird über die Schnittstelle `I-oAW-Persistenz` an die Persistenz-Komponente geliefert, die wiederum den persistenen Speicher für die dauerhafte Persistierung nutzt. Die Schnittstelle `I-oAW-Präsentation` definiert die Schnittstelle zur Präsentations-Komponente und somit die Repräsentation für den Benutzer.

5.2. Verwendete Technologien

In der Entwicklung des Java-Codegenerators (B2JSG) sind eine Reihe von Technologien und Werkzeugen zum Einsatz gekommen. Sie haben es dem Autor ermöglicht, einen Prototypen zu entwickeln. Die prototypische Entwicklung beschränkt sich ausschließlich auf „Open Source“- und Freeware-Produkte. Der Autor hat sich in der Designentscheidung für die Verwendung des Codegenerierungswerkzeugs oAW ausgesprochen. Infolgedessen sind die Auswahlkriterien für die verwendeten Technologien (Entwicklungsumgebung und Programmiersprache) abhängig von der oAW-Plattform.

Entwicklungsumgebung
Wie bereits in Kapitel 4, Abschnitt 4.3.2 erwähnt, wurde durch die starke Eclipse-Integration der oAW-Kern seit der Version vier nach Eclipse portiert. Hinsichtlich der Ausrichtung auf Eclipse ist es komfortabler, die Entwicklungsumgebung Eclipse zu benutzen. Diese Plattform hat sich über viele Jahre in der Open Source Gemeinde bewährt. Seitdem im Jahr 2001 die Elipse-Plattform von der IBM, Object Technology International (OTI) und weiteren acht Unternehmen ins Leben gerufen wurde, hat die Eclipse-Plattform eine Vielzahl von Anhängern gewonnen. Wie am Beispiel der oAW-Plattform zu sehen ist, eignet sich die Eclipse-Plattform zum einen für die Tool-Integration (Plugin-Konzept), zum anderen ist sie

plattformunabhängig und in jedem Betriebssystem ausführbar.

Programmiersprache
In der oAW-Plattform wird die Template-Sprache Xpand für die Entwicklung von Templates verwendet. Die Ausdruckskraft dieser Sprache reicht allerdings nicht immer aus. Daher bietet die oAW-Plattform Java Extentions (Erweiterungen), die in Java-Klassen ausgegliedert werden. Diese Klassen können dann über Expression aufgerufen werden, um in den Templates Java-Sourcecode zu verwenden. Um die Vorteile von Extentions nutzen zu können, ist es naheliegend, die Programmiersprache Java zu verwenden. Des Weiteren sprechen viele weitere Gründe für die Verwendung der Programmiersprache, u.a. das unterstützte objektorientierte Konzept, die weltweite Verbreitung und die Plattformunabhängigkeit.

[Eclipse (2011)] [Sherry Shavor (2004)] [Oracle (2011)]

Parser
Für die Entwicklung des Java-Sourcecode-Generators (B2JSG) wird ein XML-Parser benötigt. Er muss die folgenden Voraussetzungen erfüllen:

- Java-Kompatibilität
- Integrierbarkeit in die Eclipse-Entwicklungsumgebung
- Einfache Bedienbarkeit
- Schmal, unkompliziert zu handeln

Für den Autor ergeben sich zwei Alternativen: Simple API for XML (SAX) und JDOM 5.1.5. SAX ist schneller als JDOM und ist einfacher bedienbar. Allerdings baut JDOM einen DOM-Tree im Arbeitsspeicher auf, bei dem jederzeit auf beliebige Knoten zugegriffen werden kann [Darwin (2005)]. SAX dagegen bietet diese Technik nicht, insofern wird der JDOM vom Autor favorisiert. Zudem existieren viele Beispiele und Dokumentationen, da JDOM sehr oft eingesetzt wird.

6. Abbildung von BPMN 2.0 in Java

Dieses Kapitel befasst sich mit dem konkreten Abbilden von BPMN 2.0 Elementen in Java. Zu diesem Zweck wird auf jedes BPMN 2.0 Element der ausgewählten Spezifikationsteilmenge eingegangen. Die Abbildungsbeschreibungen der BPMN 2.0 Elemente werden nach deren Basiskategorien 2.2 strukturiert. Jede Abbildungsbeschreibung umfasst ein Abbildungskonzept, die Realisierung und abschließend, falls erforderlich, Anmerkungen. Das Abbildungskonzept beschäftigt sich mit der konkreten Konzeptidee bezüglich der Abbildung des BPMN 2.0 Elements in Java und behandelt separat den Kontrollfluss 6.1.1.1 und die Verhaltenslogik 6.1.1.2, auf die im Basiskonzept und Basisklassen genauer eingegangen wird. Die Realisierung erläutert die Umsetzung der Konzeptidee und ist unterteilt in zwei Bereiche. Der erste Bereich konzentriert sich auf die Darstellung des generierten Java-Sourcecodes, also das Ergebnis einer Generierung, und der zweite Bereich auf die Templates, mit deren Hilfe der Sourcecode generiert bzw. abgebildet wird. Die Anmerkungen enthalten Probleme, die aufgetreten sind, oder andere zusätzliche Informationen bezüglich des Abbildungsvorgangs des entsprechenden BPMN 2.0 Elements.
Bevor auf die einzelnen Abbildungsbeschreibungen eingegangen wird, folgt ein Basiskonzept der Abbildungen von BPMN 2.0 Modellen in Java und die daraus folgenden Basisklassen, die unabhängig vom BPMN 2.0 Modell generiert werden.

6.1. Basiskonzept und Basisklassen

Das Basiskonzept umfasst die grundlegende Überlegung über das Abbilden von BPMN 2.0 Modellen in Java und stellt die Grundstruktur dar. Standardmäßig werden einige Klassen benötigt, die unabhängig vom BPMN 2.0 generiert werden. Diese entsprechen den nachfolgend vorgestellten Basisklassen.

6.1.1. Basiskonzept

Es existieren nur sehr wenige Ansätze, die sich mit dem Abbilden von BPMN 2.0 Modellen bzw. BPMN 2.0 Elementen in Java beschäftigen, und die wenigen erhältlichen Informationen behandeln diese Thematik nur oberflächlich. Das Paper von Ramin Nasiri [Ramin Nasiri (2009)] beschäftigt sich lediglich mit der Abbildung von zwei BPMN-Elementen (AND und OR). Die Publikation von Aaron Calafato [Calafato (2010)] umfasst die Diskussion über einige wenige BPMN-Elemente, die vom Autor kritisch betrachtet werden. Des Weiteren sieht der Autor einen interessanten Ansatz in den Zustands-Patterns von Erich Gamma [Erich Gamma (2004)]. Unter Verwendung von Zustands-Patterns können z.B. Zustandsautomaten einfach in Java abgebildet und möglicherweise für Geschäftsprozesse übernommen werden. In jeder Abbildungsbeschreibung wird auf die vorhandenen Ansätze eingegangen und diese vom Autor kritisch betrachtet. Einige Beiträge liefern brauchbare Ansätze, andere werden vom Autor verworfen und durch eigene Alternativen ersetzt. In viele Abbildungsbeschreibungen fließen lediglich Überlegungen des Autors ein, da diesbezüglich keine Ansätze zu finden waren.
Der Autor separiert das Abbildungskonzept in zwei Bereiche: Kontrollfluss und Verhaltenslogik. Da ein Geschäftsprozess eine Abfolge von Aktivitäten umfasst, soll sich der Bereich Kontrollfluss mit dem Abbilden des Kontrollflusses beschäftigen. Die weiteren möglichen Abbildungsaspekte werden in der Verhaltenslogik abgehandelt.

6.1.1.1. Kontrollfluss

Im Ansatz von Erich Gamma (Abfolge von Zuständen in Java abbilden) und der Abfolge von Aktivitäten in BPMN sieht der Autor eine Analogie. Mit Hilfe von Zustands-Patterns können verschiedene Verhalten als Klassen abgebildet werden und ein Objekt kann nacheinander das jeweilige Verhalten annehmen. Infolgedessen findet eine Abfolge von Zustandsänderungen statt. Laut Gamma kann das zustandsabhängige Verhalten eines Objekts in einer Klasse gekapselt werden. Die zentrale Idee dahinter ist die, für jeden möglichen Zustand eine eigene Klasse zu erzeugen. Jede Klasse repräsentiert das Verhalten eines Objekts in diesem Zustand. Ändert sich der Zustand eines Objekts, ändern sich sein Verhalten und seine Klasse. Dieses Verfahren (Verhalten) ist als objektorientiertes Verhaltensmuster oder State-Pattern bekannt. Das kleine Beispiel veranschaulicht das objektbasierte Verhaltensmuster 6.1.

Es existiert eine Klasse Zustand, die das abstrakte Verhalten von Zuständen repräsentiert. Jeder konkrete Zustand (Start, Bearbeiten, Ende) wird ebenfalls als eine Klasse abgebildet und erbt von der Oberklasse Zustand. Das vererbte Verhalten wird in der konkreten Zustandsklasse spezifiziert. Ein weiterer neuer Zustand kann, durch das Erben der Oberklasse Zustand, trivial hinzugefügt werden. Die Klasse Controller verwaltet den aktuellen

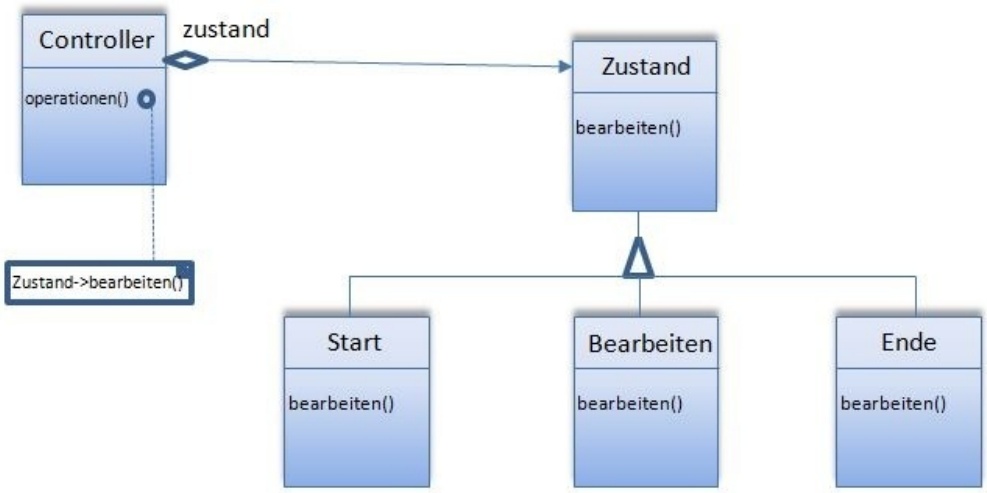


Abbildung 6.1.: Zustands-Pattern laut Ehrich Gamma

Zustand, die den aktuellen Bearbeitungsstatus (Start, Bearbeiten, Ende) repräsentiert. `operationen()` sind Methoden des Controllers, die unter andrem den aktuellen Zustand verwalten und den nächsten Zustand mittels der Methode `bearbeiten()` aufrufen. Somit werden nach und nach die Zustände geändert bzw. eine Abfolge von Zustandsänderungen wird durchgeführt. Eine Idee des Autors sieht vor, diese Abfolge auf die Abfolge innerhalb eines Geschäftsprozess zu übertragen. Ein Geschäftsprozess umfasst die Ausführung unterschiedlicher Tätigkeiten, die in einer definierten Reihenfolge durchgeführt werden, und besitzt aus diesem Grund einen Kontrollfluss. Gemäß dieser Überlegungen wird der Kontrollfluss vom Autor unter Verwendung von Zustands-Patterns abgebildet.

6.1.1.2. Verhaltenslogik

Das Abbilden eines BPMN 2.0 Modells beschränkt sich nicht auf den Kontrollfluss. Es sollen noch weitere Aspekte in Java abgebildet werden. Dazu erfolgt eine Anreicherung des zu generierenden Java-Sourcecodes mit zusätzlichen Informationen. Die weiteren Aspekte umfassen u.a. die Datenbank, Kommunikation und Logik.

Die Datenbankaspekte sollen, soweit wie möglich, ebenfalls mit abgebildet und automatisch generiert werden. Der Autor stellt zu dieser Bestrebung mehrere Fragen bzw. Anforderungen: Es soll möglich sein, unterschiedliche Datenbanken an die generierte Java-Anwendung

aufzusetzen. Des Weiteren soll es einfach sein, Objekte in der Datenbank abzuspeichern. Es besteht die Möglichkeit, mit Hilfe von OR-Mapping auf einfache Weise Objekte direkt in der Datenbank abzuspeichern. Ein OR-Mapper ermöglicht das bidirektionale Abbilden von Objekten (z.B. Java-Objekten) zu relationalen Datenbanktabellen [Abts (2010)]. Es existieren zwei OR-Mapping Tools, die für die generierte Java-Anwendung in Frage kommen; Hibernate [Community (2011)] und Ujorm [Ujorm (2011)]. Beide Frameworks bieten eine ausreichende Unterstützung bezüglich des OR-Mapping. Jedoch ist Hibernate ein sehr ausgereiftes und bewährtes Framework, welches häufig zum Einsatz kommt und weit verbreitet ist. Des Weiteren unterstützt Hibernate eine Vielzahl unterschiedlicher Datenbanksysteme, sodass der Austausch oder Einsatz vieler Datenbanken für die generierte Java-Anwendung möglich ist. Unter Berücksichtigung dieses Informationsstands hat sich der Autor für Hibernate entschieden.

Ein weiterer wichtiger Aspekt betrifft die unternehmensübergreifende Kommunikation innerhalb des Geschäftsprozesses. Die Choreography und die Collaboration müssen ebenfalls für die zu generierende Java-Anwendung abgebildet werden. Das BPMN 2.0 Modell ist zu abstrakt, um detaillierte Informationen über den konkreten Informationsfluss zwischen den Pools zu liefern. In der Collaboration können Kunden und die Unternehmen auf unterschiedliche Art und Weise kommunizieren, wie z.B. über einen Telefonanruf oder andere Kommunikationsschnittstellen; dasselbe gilt für die Choreography. In der Choreography kann die Kommunikation zu unterschiedlichen Systemen erfolgen. Der Autor strebt trotz dieser Diskrepanzen einen Lösungsweg an und entscheidet sich dafür, einen Kommunikationsweg zu wählen, der zum einen plattformunabhängig und zum anderen sprachunabhängig ist. Aufgrund dessen fällt die Entscheidung auf eine Kommunikation auf einer höheren Abstraktionsebene, wie z.B. Web Services. Für die Umsetzung von Web Services stehen mehrere Möglichkeiten zur Verfügung. Im Java-Umfeld kristallisierte sich Axis2 [Apache (2011a)] und JAX-WS [JAX-WS 2.0 (2011)] heraus. Beide Alternativen bieten eine umfangreiche Unterstützung in der Erstellung von Web Services. Da JAX-WS bereits in Java 1.6 enthalten ist, favorisiert der Autor JAX-WS.

6.1.2. Basisklassen

Für die Umsetzung des Basiskonzepts werden einige Java-Klassen benötigt, die unabhängig vom BPMN 2.0 Modell sind.
Für das Abbilden des Kontrollflusses werden vier Basisklassen verwendet (`State`, `Controller`, `ControllerFactory` und `Start`), welche sehr stark vom Entwurfsmuster Zustands-Pattern geprägt sind. Diese werden nachfolgend erläutert. Im Anschluss

daran werden die weiteren Basisklassen vorgestellt, die relevant für die Abbildung der Verhaltenslogik sind.

6.1.2.1. Kontrollfluss

Jede generierte Java-Anwendung besitzt einen Kontrollfluss. Die Basisklasse `Controller` repräsentiert diesen Kontrollfluss. Der `Controller` ist dafür zuständig, nacheinander den Geschäftsprozess zu durchlaufen. Voran zu schicken ist, dass jeder Punkt des Kontrollflusses (z.B. Event, Activity etc.) als eine Klasse mit einer Startmethode dargestellt wird. Die Startmethode steuert den inneren Ablauf des Kontrollflusspunktes an. Damit der `Controller` nacheinander die Startmethode aufrufen kann, implementiert jeder Kontrollflusspunkt das Interface `State`. `State` umfasst zwei Methoden `State work();` und `State getNextStep();`. Die `work`-Methode repräsentiert die Startmethode, die vom `Controller` aufgerufen wird, um den internen Ablauf des Kontrollflusspunktes auszuführen. Die Vervollständigung dieser Methode ist Aufgabe des Entwicklers. Am Ende der work-Methode wird als Rückgabeparameter die Methode `getNextStep` aufgerufen.

Diese Methode gibt den nächsten Kontrollflusspunkt zurück, der von der `work`-Methode weiter zum `Controller` gereicht wird. Die `getNextStep`-Methode einschließlich deren Inhalt wird vom Generator erstellt und nicht manuell vom Entwickler, es sei denn, es handelt sich bei dem Kontrollflusspunkt um ein Exclusive Gateway (Näheres dazu findet sich in der Exclusive Gateway Beschreibung). Es existiert noch eine weitere Basisklasse für den Kontrollfluss: die `ControllerFactory`, die als Singleton implementiert wurde. Die `ControllerFactory` ist zuständig für die Instanzierung eines `Controller`s. Jede generierte Java-Anwendung kann aus mehreren `Controller`n bestehen, falls parallele Abläufe im Geschäftprozess vorhanden sind. Die parallelen Abläufe werden später in Parallel Gateway 6.2.4 behandelt. Vorweg anzumerken ist, dass ein `Controller` ein Thread repräsentiert. Jede parallele Verzweigung lässt einen weiteren Thread starten, indem eine neue `Controller`-Instanz über die `ControllerFactory` erzeugt und im Anschluss darauf gestartet wird. Zusammenfassend stellt sich dies so dar: Zur Laufzeit existieren so viele Controller-Instanzen, wie parallele Abläufe durchgeführt werden. Das nachfolgende Beispiel 6.2 verdeutlich, wann und wie viele Controller-Instanzen erzeugt werden. Die Klasse `Start` steuert den Ablauf des Geschäftprozesses an, indem die erste `Controller`-Instanz mit Hilfe der `ControllerFactory` erzeugt und das StartEvent als erster Kontrollflusspunkt übergeben wird.

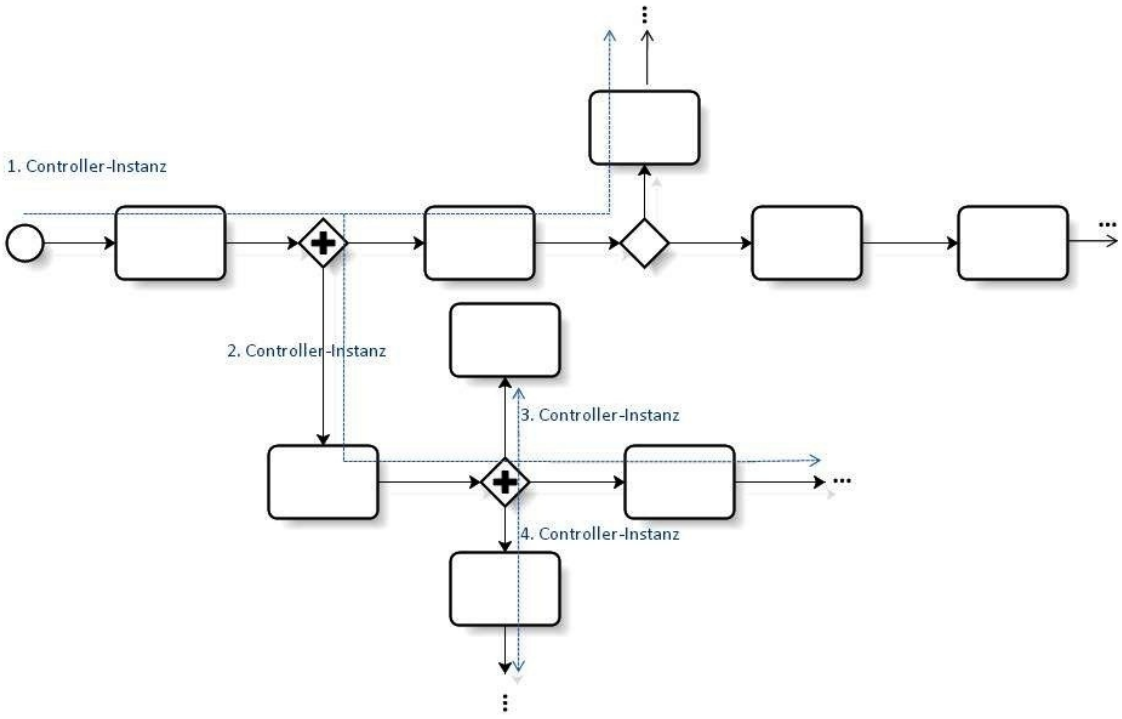


Abbildung 6.2.: Parallel ausgeführte Controller-Instanzen

6.1.2.2. Verhaltenslogik

Wie bereits im Basiskonzept erwähnt, kommt Hibernate zum Einsatz und wird wie folgt eingesetzt: Zu jedem Objekt, das in der Datenbank abgespeichert werden soll, wird eine dazugehörige DAO (Data Access Objects) Klasse erzeugt mit dem entsprechendem Interface. Die DAO-Klasse enthält Operationsmethoden für den Zugriff auf die Datenbank bezüglich des entsprechenden Objekts. Um ein Beispiel zu geben: Das Objekt der Klasse Person soll abgespeichert werden, dazu wird eine Klasse `PersonDAOImpl` und das entsprechende Interface `PersonDAO` erzeugt. Die Klasse `PersonDAOImpl` enthält beispielsweise Methoden zum Speichern, Suchen oder Löschen des Person-Objekts. Da fast jede DAO-Klasse Standard-Funktionen bietet, wie Save etc., wird eine abstrakte Klasse erzeugt, von der alle generierten DAOImpl Klassen erben. Diese Klasse ist eine der Basisklassen mit dem Namen `AbstactDAOImpl` und umfasst einige Standard Zugriffsmethoden. Die zweite Basisklasse ist das entsprechende Interface `AbstractDAO`; alle `DAOInterface` erben von dieser.
Die übergreifende Unternehmenskommunikation wird anhand von Web Service unter Verwendung von JAX-WS realisiert. JAX-WS ermöglicht eine einfache Definition von Web Services über Annotations. Jede generierte Java-Anwendung enthält eine Basisklasse, die diese Methoden sammelt und als Webservice darstellt. Der Name dieser Basisklasse setzt sich aus dem Präfix des Poolnamens mit der Endung „WebService“ zusammen.

6.2. Flow Objects

Sechs „Flow Objects“ 2.1.5.2 sind Bestandteil der Spezifikationsteilmenge, auf die nachfolgend eingegangen und deren Abbildungsmöglichkeiten diskutiert werden.

6.2.1. Event (Start, Intermediate und End)

Nachfolgend wird auf das BPMN 2.0 Element Event eingegangen und ein Abbildungskonzept erarbeitet, gefolgt von der Realisierung.

6.2.1.1. Konzept

Kontrollfluss
Jedes Event ist eine Aktivität im Kontrollfluss und repräsentiert einen Kontrollflusspunkt;

dieser wird in Java als eine Klasse abgebildet. Sie enthält mindestens eine Methode, die aufgerufen wird, falls das Event angesteuert wird. Diese Überlegung ist basierend auf dem Zustands-Pattern eingeflossen. Das End-Event beendet den Geschäftsprozess und somit den Kontrollfluss. Es werden keine weiteren Aktivitäten daraus aufgerufen.

Verhaltenslogik
Jedes Event aus der Spezifikationsteilmenge, mit Ausnahme des End-Event, beinhaltet eine Message. Die Message repräsentiert eine Nachricht, die übertragen wird. In den seltensten Fällen wird ein primitiver Datentyp übertragen. Infolgedessen kann jede Message als ein komplexer Datentyp angesehen und in Java als eine Klasse bzw. Entity abgebildet werden. Sollte es sich trotzdem um einen primitiven Datentypen handeln, kann die Klasse nachhaltig einfach gelöscht werden. Es ist sehr realistisch, dass das übertragene Message Objekt persistent in der Datenbank gehalten werden soll. Aus diesem Grund werden für jedes Message Objekt die entsprechenden DAO-Klassen erzeugt.
Des Weiteren wird in Events eine weitere Methode generiert, die den Arbeitsvorgang des Events beinhalten soll.

6.2.1.2. Realisierung

Generierter Java-Sourcecode
Wie bereits im Konzept erwähnt, wird ein Start-, Intermediate- und End-Event als eine Klasse abgebildet. Jede Klasse implementiert das Interface `State`, da es sich um einen Kontrollflusspunkt handelt. Demzufolge werden beide Methoden des Interfaces implementiert. Der Klassenname setzt sich zusammen aus dem Präfix „Start“, „Intermediate“ oder „End“ zusammengesetzt mit dem Namen des Events.

Templates zur Java-Sourcecode Generierung
Sobald ein Event im BPMN 2.0 Modell erkannt wird, wird dafür eine Klasse erstellt. Die Klasse enthält die `work`-Methode, die eine weitere Methode aufruft. Bei der weiteren Methode handelt es sich um die Methode, die den internen Ablauf enthält. Am Ende der `work`-Methode wird die Methode `getNextStep` aufgerufen, die den nächsten Kontrollpunkt zurück liefert. Die `getNextStep`-Methode erzeugt eine Instanz der Klasse, die als nächstes aufgerufen wird. Anhand der ausgehenden Sequence-Flows aus dem Event kann der nächste Kontrollpunkt aus dem Modell ermittelt werden.

6.2.2. Activity

Nachfolgend wird auf die BPMN 2.0 Element Activity eingegangen und ein Abbildungskonzept erarbeitet, gefolgt von der Realisierung und weiteren Anmerkungen.

6.2.2.1. Konzept

Kontrollfluss
In der Literatur von Aaron Calafato [Calafato (2010)] wird eine Aktivität (Activity) als eine Java-Methode abgebildet. Der Autor verfolgt teilweise diesen Ansatz. Er ist der Auffassung, dass eine Aktivity nicht lediglich in einer Methode abbildbar ist. Sollten alle Aktivities jeweils als eine Methode immer in der selben Klasse abgebildet werden, würde der ganze Geschäftsprozess innerhalb einer Klasse mit einer hohen Anzahl an Methoden verlaufen. Zum einen ist dies ein sehr unübersichtlicher Ansatz, und zum anderen verfolgt er eher den prozeduralen und nicht den objektorientierten Ansatz. Aus den genannten Gründen wird jede Aktivität als eine separate Java-Klasse abgebildet und analog [Calafato (2010)] enthält diese Klasse eine Methode (`work`), über die die Activity angesteuert wird.

Verhaltenslogik
Wie bereits erwähnt, repräsentiert eine Activity eine Aktivität. Aufgrund dessen entscheidet der Autor, dass die Klasse eine zusätzliche Methode enthalten soll, die verantwortlich für die Durchführung dieser Aktivität ist. In der Regel setzt sich der Name einer Activity aus einem Nomen und einem Verb zusammen (z.B. Dokumente bearbeiten). Infolgedessen trägt die Klasse den Namen des Nomens und die Methoden, die für die Aktivitäten-Ausführung steht, logischerweise den Namen des Verbs.

6.2.2.2. Realisierung

Generierter Java-Sourcecode
Äquivalent zu den Events implementiert jede Activity-Klasse das Interface `State`, da es sich ebenfalls um einen Kontrollflusspunkt handelt. Die `work` Methode enthält lediglich den Aufruf der Methode, die den internen Arbeitsschritt enthält und am Ende den Aufruf der `getNextStep`-Methode die den nächsten Kontrollflusspunkt zurückliefert.
Nachfolgend ein Beispiel 6.3 einer generierten Java-Klasse, basierend auf einem BPMN 2.0 Activity Element.

Listing 6.1: Java-Code: Activity

```
package process_verschiffungsunternehmen.service;

import process_verschiffungsunternehmen.service.*;
import process_verschiffungsunternehmen.ladestelle.*;
import process_verschiffungsunternehmen.spedition.*;
import process_verschiffungsunternehmen.State;
import process_verschiffungsunternehmen.dao.source.*;
/**********************************************************
 *
 * @Autor            Arazm Hosieny
 *
 **********************************************************/
public class DokumenteVorbereiten implements State {

        public State work() {

                        vorbereiten();

                return getNextStep();
        }

        @Override
        public State getNextStep() {
                System.out.println("nächster␣Schritt:Dokumente␣schicken");
                return new DokumenteSchicken();
        }

        public void vorbereiten() {
                System.out.println("vorbereiten");
                // TODO: Vom Entwickler zu vervollständigen
        }
}
```

Templates zur Java-Sourcecode Generierung

Die generierte Klasse wirkt trivial, jedoch ist der dazugehörige Template-Code Komplexer Natur. Die Iterationen über das XMI-Modell, um alle notwendigen Informationen aus dem Modell zu lesen, gestaltet sich nicht so einfach durchzuführen. Ein kleiner Ausschnitt des Templates für die Codegenierung einer Activity soll dies verdeutlichen. Auf die Interpretation des Codes wird bewusst verzichtet. Das grundlegende Template-Konzept wurde bereits in den existierenden Ansätzen 4.3.2 kurz eingeführt.

Listing 6.2: Template-Code: Activity

```
«IMPORT bpmn2»
«EXTENSION template::MyExtensions»

«DEFINE classCreaterTask FOR Task»
```

```
«LET ((bpmn2::Process)eContainer) AS projekt-»

«FILE projekt.name.toLowerCase()+"/"+getLaneName(this.lanes.first().name).toLowerCase()+
"/"+insertCharacter(this.name)+".java"»
        package «((bpmn2::Process)eContainer).name.toLowerCase()».
        «getLaneName(this.lanes.first().name).toLowerCase()»;

«FOREACH projekt.laneSets.lanes AS lane»
import «projekt.name.toLowerCase()».«lane.name.toLowerCase()».*;
«ENDFOREACH»
import «projekt.name.toLowerCase()».State;
import «projekt.name.toLowerCase()».dao.source.*;
/*****************************************************
*
* @Autor            Arazm Hosieny
*
*****************************************************/

public class «insertCharacter(this.name)» implements State{
 «REM»Hier wird die Methode work = Hauptmehtode der Klasse erstellt«ENDREM»
 «REM» (ist im Interface State definiert)«ENDREM»
 «EXPAND DefaultClassGenerator::workMethodStart FOR this-»
 «REM»Methodenaufruf«ENDREM»
 «filterMethod(this.name)»();
 «REM»Über alle MessageFlows iterieren und schauen,«ENDREM»
 «REM»ob dieser Task eine ausgehendes MessageFlow hat«ENDREM»
 «LET (bpmn2::Definitions)this.eRootContainer AS rootDef-»
  «FOREACH rootDef.rootElements AS rootElement-»
   «FOREACH rootElement.eAllContents AS innerElement-»
    «IF innerElement.metaType.name.compareTo("bpmn2::MessageFlow") == 0-»
     «LET (bpmn2::MessageFlow)innerElement AS messageFlow-»
      «LET messageFlow.sourceRef AS aSourceRef-»
       «IF aSourceRef.metaType.name.compareTo("bpmn2::Task") == 0-»
        «LET (bpmn2::Task)aSourceRef AS aTask-»
          «REM»Prüfen ob Task ein ausgehendes MessageFlow hat«ENDREM»
          «LET this.id AS aThisTaskId-»
            «IF aTask.id.compareTo(aThisTaskId) == 0-»
                        .
                        .
                        .
                        .
```

6.2.2.3. Anmerkung

Problematiken

Ein wesentliches Problem ergab sich im Zusammenhang mit der Namensgebung. Wie bereits erwähnt, erhält der Klassenname das Nomen der Activity und die enthaltene Methode

das Verb der Activity. Trägt eine Activity den Namen Dokument verschicken und eine weitere Dokument bearbeiten, werden zwei Klassen mit dem Namen Dokument erzeugt, was in Java nicht erlaubt ist. Ein möglicher Lösungsweg, der Java konform ist, bietet sich aus der Zusammensetzung des Klassennamens, aus dem Verb und dem Nomen. Das Resultat aus diesem Beispiel wäre eine Klasse mit dem Namen DokumentVerschicken und eine mit dem Namen DokumentBearbeiten. Es ist lediglich ein kleines Problem in der Namenskonversion aufgetreten. Wird die Problematik weiter betrachtet, können viele weitere Problematiken in diesem Bereich auftreten. Aufgrund dessen, dass die BPMN 2.0 Spezifikation keine Namensgebung definiert, können bei unterschiedlichen Modellen viele weitere derartige Probleme auftreten. Es existieren mehrere Möglichkeiten, dieser Problematik entgegen zu wirken, jedoch würde die Auseinandersetzung mit diesem Themengebiet den Rahmen der Masterarbeit sprengen. Ein einfacher Lösungsweg sieht vor, alle Wörter des Activity-Namens miteinander zu verbinden und den ersten Buchstaben jedes Wortes groß zu schreiben.

6.2.3. Exclusive Gateway

Nachfolgend wird auf das BPMN 2.0 Element Exclusive Gateway eingegangen und ein Abbildungskonzept erarbeitet, gefolgt von der Realisierung und weiteren Anmerkungen.

6.2.3.1. Konzept

Kontrollfluss

Das Exclusive Gateway stellt bei mehr als einem ausgehendem Fluss eine Entscheidung und bei mehr als einem eingehendem Fluss eine Zusammenführung dar.
Die Entscheidung bedeutet lediglich, dass eine Activity oder ein Event genau eine andere Activity oder ein anderes Event aus den Alternativen aufruft, worauf das Exclusive Gateway hinweist. Durch die abstrakten (fehlenden) Informationen in BPMN 2.0 ist die Generierung eines vollständigen Mechanismus, der die Entscheidung der Alternativen durchführt, nicht möglich. Aus diesem Grund wird eine Methode für den Entscheidungsweg generiert, die nicht vollständig ist. Im Gegensatz zum Ansatz von Ramin Nasiri wird lediglich eine Methode generiert. Innerhalb der Methode werden alle möglichen Alternativzweige in Kommentaren aufgeführt und eine Alternative als „default“ gewählt. Bei der Vervollständigung der Anwendung muss der Entwickler, die Entscheidungslogik in die Methode einbauen. Die Literatur von Ramin Nasiri [Ramin Nasiri (2009)] bildet jede alternative Verzweigung des „Exclusive Gateway“ als eine separate Methode ab. Jede Methode überprüft für die Verzweigung, ob die Bedingung erfüllt ist und dieser Weg eingeschlagen werden kann. Bei dem Ansatz

von Ramin Nasiri müssen ebenfalls die Bedingungsmethoden vervollständigt werden. Die `getNextStep`-Methode ist immer dafür zuständig, den nächsten Kontrollflusspunkt zu berechnen bzw. zurückzuliefern. Diese Struktur soll beibehalten werden, und infolgedessen wird der Entscheidungsweg in diese eine Methode ausgelagert. Aus diesem Grund verzichtet der Autor auf eine Methode pro Alternativverzweigung (wie Ramin Nasiri). Sollte jedoch die Entscheidungsberechnung komplexer Ausfallen, ist es dem Entwickler überlassen, die Entscheidungsmethode (`getNextStep`) in weitere kleinere Methoden auszulagern.
Die nächste zu klärende Frage ist die Platzierung der Entscheidungsmethode. Wird die Methode in die vorige Activity oder das vorige Event generiert oder eine separate Klasse, die die Entscheidungmethode enthält, erzeugt? Der Autor entscheidet sich für die separate Klasse, um die Entkopplung der Kontrollsteuerung zu der Logik zu erreichen.
Die Zusammenführung ist zuständig für die Vereinigung parallel ausgeführter Abläufe. Auf welche Art und Weise die parallelen Abläufe wieder zusammengeführt werden, ist abhängig von der Art, wie die Abbildung der Parallelität in Java durchgeführt wird. Diese folgt in der Parallel Gateway Abbildungsbeschreibung 6.2.4. Vorweg sei anzumerken, dass mit Hilfe von Threads die parallelen Abläufe abgebildet werden. Demzufolge, ist die Zusammenführung von parallelen Abläufen die Zusammenführung von Threads. Das bedeutet, dass mehrere in ein Exclusive Gateway eingehende Threads zu genau einem Thread zusammengeführt werden.

Verhaltenslogik

—

6.2.3.2. Realisierung

Generierter Java-Sourcecode

Das Exclusive Gateway wird, wie im Konzept festgelegt, als eine Klasse abgebildet, die ebenfalls das Interface State implementiert, da sie ebenfalls einen Kontrollflusspunkt darstellt und zudem die `getNextStep`-Methode zur Entscheidung benötigt. In der `work`-Methode wird lediglich die `getNextStep`-Methode aufgerufen. Die `getNextStep`-Methode repräsentiert die Entscheidungsmethode, die vom Entwickler vervollständigt werden muss. Die `getNextStep`-Methode enthält alle Alternativzweige im Kommentar, und ein default Wert (der letzte mögliche Kontrollflusspunkt) wird gesetzt. Aufgrund dessen, dass alle Alternativzweige als Kommentar vorhanden sind, liegen dem Entwickler alle Alternativen vor, und er muss lediglich die Entscheidungskriterien einbinden. Nachfolgend wird ein Beispiel des generierten Java-Sourcecodes dargestellt

Listing 6.3: Java-Code: Exclusive Gateway

```
package process_verschiffungsunternehmen.service;
package process_verschiffungsunternehmen.ladestelle;
```

```
import process_verschiffungsunternehmen.service.*;
import process_verschiffungsunternehmen.ladestelle.*;
import process_verschiffungsunternehmen.spedition.*;
import process_verschiffungsunternehmen.State;
/*******************************************************
 *
 * @Autor      Arazm Hosieny
 * @Erstellung 08.02.2011
 *
 *******************************************************/
public class ExclusivBeladung implements State {

 public State work() {
  return getNextStep();
 }

 @Override
 public State getNextStep() {

  /*TODO
  Welcher Weg letztentlich eingeschlagen wird, muss vom Entwickler
  implementiert werden. Default wird der letzte Weg genommen, die
  weiteren sind in Kommatar zu betrachten*/

  //return new  ContainerTransportieren();

  return new KundeBenachrichtigen();

 }
}
```

Templates zur Java-Sourcecode Generierung
Das Abbilden der Klasse im Template ist äquivalent zu jedem anderen Kontrollflusspunkt, lediglich die Methode `getNextStep` stellt sich anders dar. Um in der `getNextStep`-Methode alle Alternativ-Kontrollflusspunkte aufzulisten, müssen aus dem Modell alle ausgehenden Sequence-Flows aus dem Exclusive Gateway betrachtet werden. Die ausgehenden Sequence-Flows referenzieren auf alle Alternativen, die in der `getNextStep` Methode aufgelistet werden.

6.2.3.3. Anmerkung

Problematiken
Im Konzept wurde entschieden, das Exclusive Gateway ale eine separate Klasse darzustellen, die die Entscheidungsmethode umfasst. Die andere Alternative bestünde darin, die

Entscheidungsmethode in der vorherigen Activity oder dem vorherigen Event zu platzieren. Die Entscheidung fiel zu Gunsten einer Trennung des Kontrollflusses und der Verhaltenslogik aus. Diese Entscheidung ist legitim, jedoch in Anbetracht dessen, dass die Entscheidungsmethode abhängig ist von der vorherigen Aktivity oder dem vorherigen Event, können Informationen notwendig sein, die in der separaten Klasse nicht mehr zur Verfügung stehen. Der Entwickler müsste infolgedessen weitere nicht vorgesehene Informationen über den Kontrollfluss senden. Eine Möglichkeit bestände darin, direkt bei der Instanziierung des nächsten Kontrollflusspunktes weitere Informationen an das Objekt anzufügen. Eine weitere triviale Alternative ist die, die Entscheidungsmethode doch auf der vorherigen Activity oder dem vorherigen Event zu belassen.

6.2.4. Parallel Gateway

Nachfolgend wird auf das BPMN 2.0 Element Parallel Gateway eingegangen und ein Abbildungskonzept erarbeitet, gefolgt von der Realisierung und weiteren Anmerkungen.

6.2.4.1. Konzept

Kontrollfluss
Die Parallel Gateways beschreiben die Parallelität von Abläufen. Durch dieses Gateway findet eine Verzweigung statt. Der Fluss geht in mindestens zwei parallele Flüsse über. Um die Parallelität in Java abzubilden, stellt sich die Frage, welche Möglichkeiten Java bereits anbietet. Die bekannteste und sehr stark vertretene Lösung für die Realisierung von Parallelitäten in Java sind Threads. Mit Hilfe von Threads können nebenläufige (parallele) Prozesse in Java abgebildet werden. Mehr zum Thema Threads sei in der Literatur von Gosling [James Gosling (2005)] und Flanagan [Flanagan (2005)] nachzulesen. Der Gedanke von Aaron Calafato [Calafato (2010)] zur Abbildung von „Parallel Gateways“ ist die Verwendung von Threads. Dies bestärkt den Autor in seiner Entscheidung, Parallel Gateways ebenfalls als Threads abzubilden. Ab der zweiten Parallelität wird für jeden nebenläufigen Ablauf ein neuer Thread erzeugt und gestartet. Das bedeutet, zwei parallele Abläufe sehen einen neuen Thread vor, drei parallele Abläufe zwei neue Threads, vier parallele Abläufe drei neue Thread usw.. Es bestehen in Java zwei Möglichkeiten, Threads abzubilden, zum einen über das Implementieren des Interfaces `Runnable`, zum anderen über das Erben der Klasse `Thread`, die wiederum ebenfalls das Interface `Runnable` implementiert [James Gosling (2005)]. Beide Varianten sind trivial umsetzbar, aufgrund dessen können beide verwendet werden. Der Autor entscheidet sich für die Interface-Variante.

Identisch mit den Exclusive Gateway stellt sich ebenfalls hier die Frage, an welcher Stelle die Erzeugung und das Starten der Threads erfolgt. Sollten die Erstellung und das Starten der Threads in der vorigen Activity oder dem vorigen Event geschehen, oder soll zu diesem Zweck eine separate Klasse erzeugt werden? Die Begründung, eine separate Klasse zu generieren, beruht auf der Entkopplung der Kontrollflusssteuerung von der Logik.

Verhaltenslogik

—

6.2.4.2. Realisierung

Generierter Java-Sourcecode

Das Parallel Gateway wird als eine separate Klasse abgebildet und implementiert das Interface `State`. Äquivalent zum Exclusive Gateway umfasst die `work`-Methode lediglich den Aufruf der `getNextStep`-Methode. Die `getNextStep`-Methode umfasst die Erzeugung der parallelen Abläufe. Wie im Konzept bereits erwähnt, werden die parallelen Abläufe mit Hilfe von Threads realisiert. Für die Erstellung und das Starten eines Thread wird lediglich eine neue Instanz der Klasse `Controller` erzeugt und der nächste bzw. erste Kontrollpunkt mitgegeben. Werden drei parallele Abläufe ausgeführt, werden zwei neue Instanzen der Klasse `Controller` erzeugt, und einer der drei parallelen Abläufe verwendet den ursprünglichen Controller. Nachfolgend wird ein Beispiel 6.4 für den generierten Java-Sourcecode visualisiert.

Listing 6.4: Java-Code: Parallel Gateway

```
package process_verschiffungsunternehmen.service;

import process_verschiffungsunternehmen.State;
import process_verschiffungsunternehmen.ControllerFactory;
import process_verschiffungsunternehmen.service.*;
import process_verschiffungsunternehmen.ladestelle.*;
import process_verschiffungsunternehmen.spedition.*;

/*******************************************************
 *
 * @Autor            Arazm Hosieny
 * @Erstellung  08.02.2011
 *
 *******************************************************/
public class ParallelGateway implements State {
 public State work() {
                return getNextStep();
        }
```

```
        @Override
        public State getNextStep() {

                // Neue Controller-Instanz (neuen Thread erzeugen und starten)
                ControllerFactory.getInstance().createController(
                                new DokumenteVorbereiten());

                // Neue Controller-Instanz (neuen Thread erzeugen und starten)
                ControllerFactory.getInstance()
                                .createController(new ContainerBeladen());

                // Alte Controller-Instanz weiterführen
                return new Intermediate1Status();
        }
}
```

Templates zur Java-Sourcecode Generierung

Mit Hilfe der ausgehenden SequenceFlows werden die nächsten parallelen Activites oder Events ermittelt. Der nachfolgende Code-Ausschnitt 6.5 aus den Templates stellt das Ermitteln der parallelen Activities und/oder Events dar.

Listing 6.5: Template-Code: Parallel Gateway

```
        .
        .
        .
  «REM»Die Nachfolger bestimmen«ENDREM»
  @Override
  public State getNextStep() {

  // Neue Controller-Instanz (neuen Thread erzeugen und starten)
  «FOREACH this.outgoing AS flow-»
    «LET this.outgoing.size AS outSize-»
    «LET flow.targetRef AS target-»
    «REM»Falls nicht letztes Element => Thread erzeugen«ENDREM»
      «IF (this.outgoing.indexOf(flow) != (outSize-1))-»
       ControllerFactory.getInstance().createController(new
       «insertCharacter(target.name)»());
      «ELSE-»
      «REM»Falls letztes Element => alten Thread weiter benutzen«ENDREM»
      «IF (this.outgoing.indexOf(flow) == (outSize-1))-»
       return new  «insertCharacter(target.name)»();
      «ENDIF-»
      «ENDIF-»
   «ENDLET-»
   «ENDLET-»
  «ENDFOREACH-»
   }
  }
        .
```

.
.

6.2.4.3. Anmerkung

Hinweise
Im Gegensatz zu den Exclusive Gateways kann die Erzeugung der Threads separat in eine eigene Klasse ausgelagert werden und nicht in der vorherigen Activity oder dem vorherigen Event vorgenommen werden. Die Klasse Parallel Gateway benötigt keine Codevervollständigung, infolgedessen werden ebenfalls keine Informationen von der vorherigen Activity oder dem vorherigen Event benötigt.

6.3. Connecting Objects

Zwei Connecting Objects sind in der Spezifikationsteilmenge enthalten. Nachfolgend wird deren Abbildung in Java diskutiert.

6.3.1. Sequence Flow

Nachfolgend wird auf das BPMN 2.0 Element Sequence Flow eingegangen und ein Abbildungskonzept erarbeitet, gefolgt von der Realisierung.

6.3.1.1. Konzept

Kontrollfluss
Der „Sequence Flow“ steuert den nächsten Kontrollflusspunkt (Activity, Event, Steuerelement etc.) an. Äquivalent zu Aaron Calafato [Calafato (2010)] wird der „Sequence Flow“ als eine aufrufende Methode abgebildet.

Verhaltenslogik
—

6.3.1.2. Realisierung

Generierter Java-Sourcecode
Bei der aufgerufenen Methode, die den Sequende Flow abbildet, handelt es sich um die `work`-Methode. Wie bereits angemerkt, ist diese Methode standardmäßig in jedem Kontrollflusspunkt enthalten.

Templates zur Java-Sourcecode Generierung
Es existiert kein separates Template für das Abbilden des Sequence Flows. Angesichts dessen, dass Sequence Flows als `work`-Methode abgebildet werden und die `work`-Methode in jedem Kontrollflusspunkt enthalten ist, ist das Abbilden bereits in den anderen Templates enthalten.

6.3.2. Message Flow

Nachfolgend wird auf das BPMN 2.0 Element Message Flow eingegangen und ein Abbildungskonzept erarbeitet, gefolgt von der Realisierung und weiteren Anmerkungen.

6.3.2.1. Konzept

Kontrollfluss
Der Message Flow nimmt die selbe Rolle im Kontrollfluss an wie der Sequence Flow, demzufolge wird äquivalent zum Sequence Flow die Message Flow ebenfalls als eine aufrufende Methode abgebildet.

Verhaltenslogik
Es existieren zwei wesentliche Unterschiede zu den Sequence Flows. Zum einen werden „Message Flows“ verwendet, um über die Pool-Grenzen hinaus zu kommunizieren. Zum anderen werden sogenannte „Messages“ übertragen. Wie bereits in 6.1.1.2 angesprochen, wird die Unternehmensübergreifende Kommunikation mittels Web Services realisiert. Demnach wird die aufgerufene Methode als eine „Web Service“-Methode abgebildet. Die übertrage Nachricht (Message) wird, wie bereits in 6.2.1 erwähnt, als eine Klasse abgebildet. Demzufolge werden Messages als Objekte über die Pool-Grenzen hinaus transferiert. Des Weiteren ist es nahe liegend, dass die übertragenden Message-Objekte auf beiden

Pool-Seiten persistiert werden. Aufgrund dessen sollen auf beiden Poolseiten alle Message-Klassen einschließlich der entsprechenden DAO-Klassen generiert werden.

6.3.2.2. Realisierung

Generierter Java-Sourcecode
Für einen Message Flow werden mehrere Klassen generiert, eine Klasse, die die übertragene Message repräsentiert und zu der Message-Klasse die entsprechende DAOImpl-Klasse und das DAO-Interface. Zusätzlich müssen ebenfalls Web Service Methoden erzeugt werden. Zu bedenken sei, dass die entsprechenden Klassen auf der Sender sowie auf der Empfänger Seite generiert werden müssen.

Templates zur Java-Sourcecode Generierung
In der Template Sprache Xpand ist es nicht möglich, eine bereits generierte Klasse weiter zu modifizieren bzw. um weiteren Sourcecode zu erweitern. Das Erzeugen der unterschiedlichen Klassen ist ein sehr komplexer Vorgang. Der Autor musste auf die Java-Extentions zurückgreifen, um mögliche Schwierigkeiten zu umgehen. Innerhalb des Erzeugungsprozess einer Klasse werden Informationen gesammelt, die notwendig sind, um anschließend die weiteren Klassen zu erzeugen. Die Informationen werden in einer HashMap gesammelt. Somit wird verhindert, dass ein Generierungsprozess einer Klasse unterbrochen wird oder eine bereits generierte Klasse modifiziert werden muss.

6.3.2.3. Anmerkung

Problematiken
Während der Realisierung der Abbildungen für die Message Flows zeigte sich ein Fehler im BPMN 2.0 Metamodell. Innerhalb des Templates wird `MessageFlow.messageRef` aufgerufen, um den Empfänger der Message zu ermitteln. In der Regel gibt dieser Aufruf ein `MessageEventDefinition`-Objekt zurück. Zur Kompilierzeit wird allerdings angezeigt, dass ein `Message`-Objekt zurückgeliefert wird. Zur Laufzeit wird jedoch das richtige Objekt zurückgeliefert, ein `MessageEventDefinition`-Objekt. Dieser Sachverhalt tritt ein, da im Modell tatsächlich der `MessageFlow` als `messageRef` eine `MessageEventDefinition` enthält. Das Metamodell definiert jedoch, dass sich ein `Message`-Objekt an dieser Stelle befinden soll. Um dieser Problematik entgegen zu wirken, sollte das verwendete BPMN 2.0 Metamodell verbessert werden.

Hinweise
Es werden lediglich die Web Services auf der Seite generiert, die diese zur Verfügung stellt. Mit Hilfe von JAX-WS kann im Anschluss automatisch die Gegenseite generiert werden. Somit werden die Web Services ebenfalls für die Gegenseite automatisch generiert, lediglich unter Verwendung von JAX-WS.

6.4. Swimlanes

Die Basiskategorie Swimlanes umfasst zwei BPMN 2.0 Elemente der Spezifikationsteilmenge.

6.4.1. Lanes

Nachfolgend wird auf das BPMN 2.0 Element Lane eingegangen und ein Abbildungskonzept erarbeitet, gefolgt von der Realisierung.

6.4.1.1. Konzept

Kontrollfluss
—

Verhaltenslogik
Lanes repräsentieren die einzelnen Bereiche in einem Unternehmen, die vom Geschäftsprozess betroffen sind. Es stellt sich die Frage, wie diese in Java abgebildet werden kann. Anhand des BPMN 2.0 Modells lässt sich nicht feststellen, inwieweit die einzelnen Bereiche miteinander verbunden sind. Aus diesem Grund kann nicht beurteilt werden, ob ein individuelles Projekt oder alles in einem Projekt zusammengefasst generiert werden soll. Da der zu realisierende Generator lediglich Java-Sourcecode für Kleinunternehmen erzeugen soll, ist die Wahrscheinlichkeit höher, dass alles in einem Projekt gekapselt wird. Der Autor entschied sich dafür, Lanes in separaten Packages abzubilden und alle zusammengehörigen BPMN 2.0 Elemente einer Lane in einem Java-Package zu gruppieren. Infolgedessen besteht die Möglichkeit, engverbundene Unternehmensbereiche verbunden zu lassen und extrem disparate Bereiche separat zu behandelt, indem die Package einfach gekapselt in

ein neues Projekt eingefügt werden kann.

6.4.1.2. Realisierung

Generierter Java-Sourcecode
Bezogen auf das Konzept wird jedes Lane als ein Java-Package abgebildet. Der Name des Package setzt sich aus dem Präfix „process_“ vereint mit dem Lanenamen, der im Modell vorhanden ist, zusammen. Diese Packages umfassen alle Klassen, die aus den BPMN 2.0 Elementen abgebildet wurden, die sich wiederum innerhalb der Lanes befinden. Ein generiertes Java-Projekt umfasst mindestens so viele Packages, wie Lanes innerhalb eines Pools existieren.

Templates zur Java-Sourcecode Generierung
Für jede generierte Klasse ist es notwendig, oben in der Klassendefinition den Packagenamen aufzuführen. Aus diesem Grund muss für jedes BPMN-Element, das als Klasse abgebildet wird, der Lanename aus dem Modell ermittelt werden. Die konkreten BPMN 2.0 Elemente referenzieren nicht direkt auf die Lane, in der sie sich befinden, infolgedessen müssen für die Ermittlung des Lanenamens Iterationen über Referenzen durchgeführt werden.

6.4.2. Pool

Nachfolgend wird auf das BPMN 2.0 Element Pool eingegangen und ein Abbildungskonzept erarbeitet, gefolgt von der Realisierung.

6.4.2.1. Konzept

Kontrollfluss
—

Verhaltenslogik
Pools repräsentieren Unternehmen oder Kunden. Daher werden alle BPMN 2.0 Elemente

eines Pools in ein separates Java Projekt generiert. Somit werden die Unternehmensgrenzen anhand von separaten Projekten realisiert.

6.4.2.2. Realisierung

Generierter Java-Sourcecode
Jeder Pool wird als ein Java-Projekt abgebildet. Der Projektname erhält den Namen des Pools. Das Projekt umfasst alle Packages und Klassen, die abgebildet wurden aus den BPMN 2.0 Elementen, die innerhalb des Pools existieren.

Templates zur Java-Sourcecode Generierung
Das Ermitteln des Pool gestaltete sich einfacher als das Ermitteln der Lanes. Fast jedes BPMN 2.0 Element referenziert direkt auf seinen Pool.

6.5. Zusammenfassung der Abbildungen

Abschließend ist anzumerken, dass der Autor bei der Realisierung nicht auf alle Details eingehen konnte. Beispielsweise wurden bei der Realisierung weitere Java-Extensions in Form von Hilfsklassen verwendet, um den Funktionsumfang von Xpand zu erweitern. Ebenso wurde auf eine vollständige Auflistung aller Problematiken verzichtet, da dieses den Rahmen dieser Masterarbeit sprengen würde. Demzufolge hat sich der Autor in der Realisierung auf das Wesentliche konzentriert.

BPMN 2.0 Element	Abbildung in Java
Event	Klasse; Methode (Kontrollsteuerung); Methode(Arbeitsvorgang)
Activity	Klasse; Methode (Kontrollsteuerung); Methode (Arbeitsvorgang)
Message	Klasse (Entity)
Exclusive Gateway	Klasse; Methode (Entscheidung/Zusammenführung)
Parallel Gateway	Klasse (Erzeugen und Starten von Threads)
Sequence Flow	Methodenaufruf
Message Flow	Webservice; Datenbank-Aspekte
Lane	Package
Pool	Projekt

Tabelle 6.1.: Zusammenfassung der Abbildungen

6.6. Testfälle

Der Autor hat zur Qualitätssicherung während der gesamten Entwicklungsphase das System (B2JSG) kontinuierlichen Tests unterzogen. Dadurch wurde überprüft, ob der generierte Java-Sourcecode korrekt abgebildet wird. Die Grundlage für die Tests waren zum einen der Logistikprozess (BPMN-Modell) 2.3, der in den Grundlagen aufgeführt wurde. Zum anderen wurden am Ende der Realisierungsphase weitere Tests auf anderen BPMN-Diagrammen durchgeführt. Exemplarisch werden an dieser Stelle die generierten Java-Projekte als UML-Diagramme (Abbildungen 6.3 und 6.4) dargestellt, basierend auf dem Logistiksystem. Es sind drei Projekte, die den internen Ablauf des Kunden, des Verschiffungsunternehmens und der Reederei mit der Kommunikation zwischen den Projekten umfassen, als Java-Sourcecode zu erwarten.
Wie erwartet, wird das Projekt Kunde generiert. Hierbei werden ebenfalls alle Basisklassen erzeugt. Sie sind jedoch an dieser Stelle zu vernachlässigen, da im BPMN-Diagramm des Verschiffungsprozesses der interne Ablauf bzgl. des Kunden nicht spezifiziert, sondern als Black-Box dargestellt wurde. Das Projekt Verschiffungsunternehmen besitzt ebenso alle sieben Basisklassen in einem Package (process_verschiffungsunternehmen). Zusätzlich werden vier Message-Klassen und die dazugehörigen DAO-Klassen bzw. Interfaces in zwei Packages (process_verschiffungsunternehmen.dao.source und process_verschiffungsunternehmen.dao.target) gekapselt. Dieses Package (process_verschiffungsunternehmen.dao.source) enthält alle notwendigen Message-Objekte, die über die Prozessgrenzen hinaus verschickt werden, während das zweite Package (process_verschiffungsunternehmen.dao.target) alle ankommenden Massage-Objekte an das Verschiffungsunternehmen umfasst. Die drei Lanes (Service, Ladestelle und Verschiffung) werden ebenfalls als Packages abgebildet. Daher steht in der Abbildung als Suffix der entsprechende Lane-Name (z.B. process_verschiffungsunternehmen.ladestelle). In den Lanes des Verschiffungsunternehmens befanden sich insgesamt zwölf Kontrollflusspunkte (z.B. Activity, Event, Intermediate ect.). Ordnungsgemäß werden in den drei Packages die zwölf entsprechenden Klassen generiert. Die Klassen werden abhängig davon, wo sich der entsprechende Kontrollflusspunkt im Lane befand, den entsprechenden Packages zugeordnet. Der interne Prozess der Reederei weist die gleiche Projekt-, Package- und Klassen-Strukturauf. Ebenso sind hier die Basisklassen, Message- und DAO-Klassen und die Lanes mit den zugehörigen Kontrollflusspunkten als Klassen abgebildet. Die Reederei trägt im BPMN-Diagramm keinen Lane-Namen. Aufgrund dessen wird automatisch als Package-Name der Suffix defaultlane generiert.

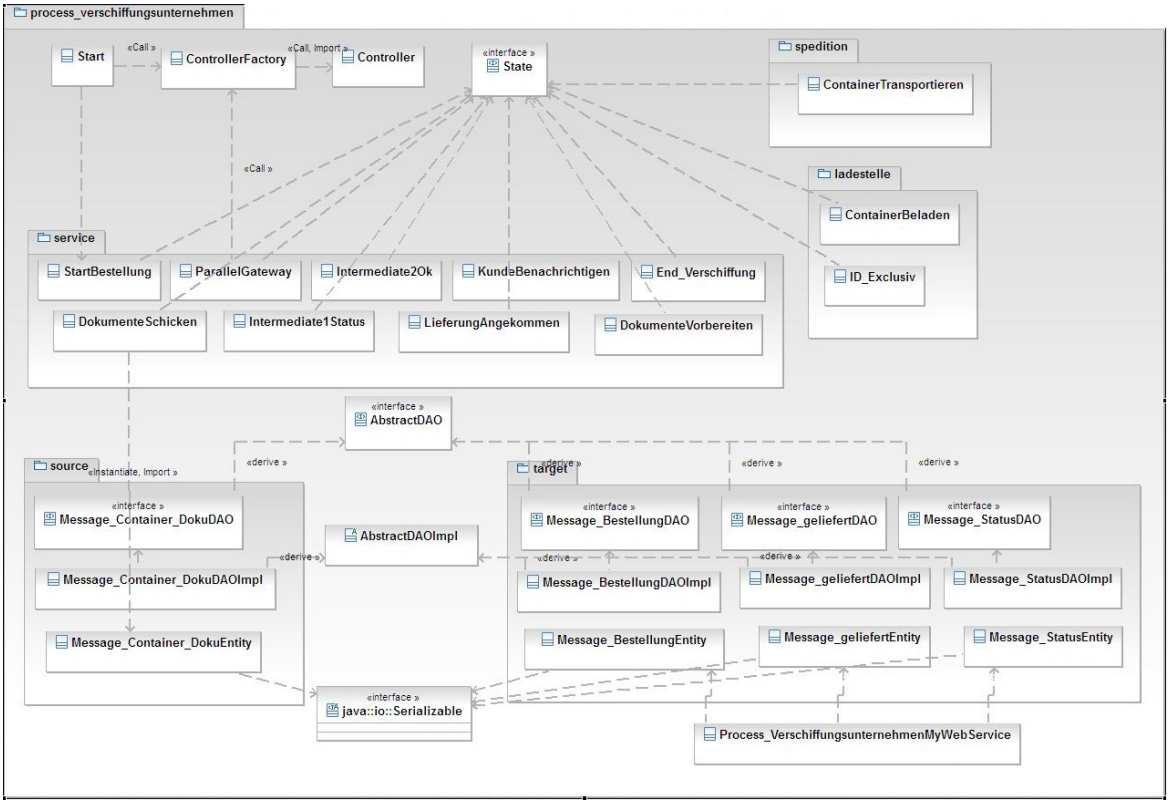


Abbildung 6.3.: Prozess: Verschiffungsunternehmen

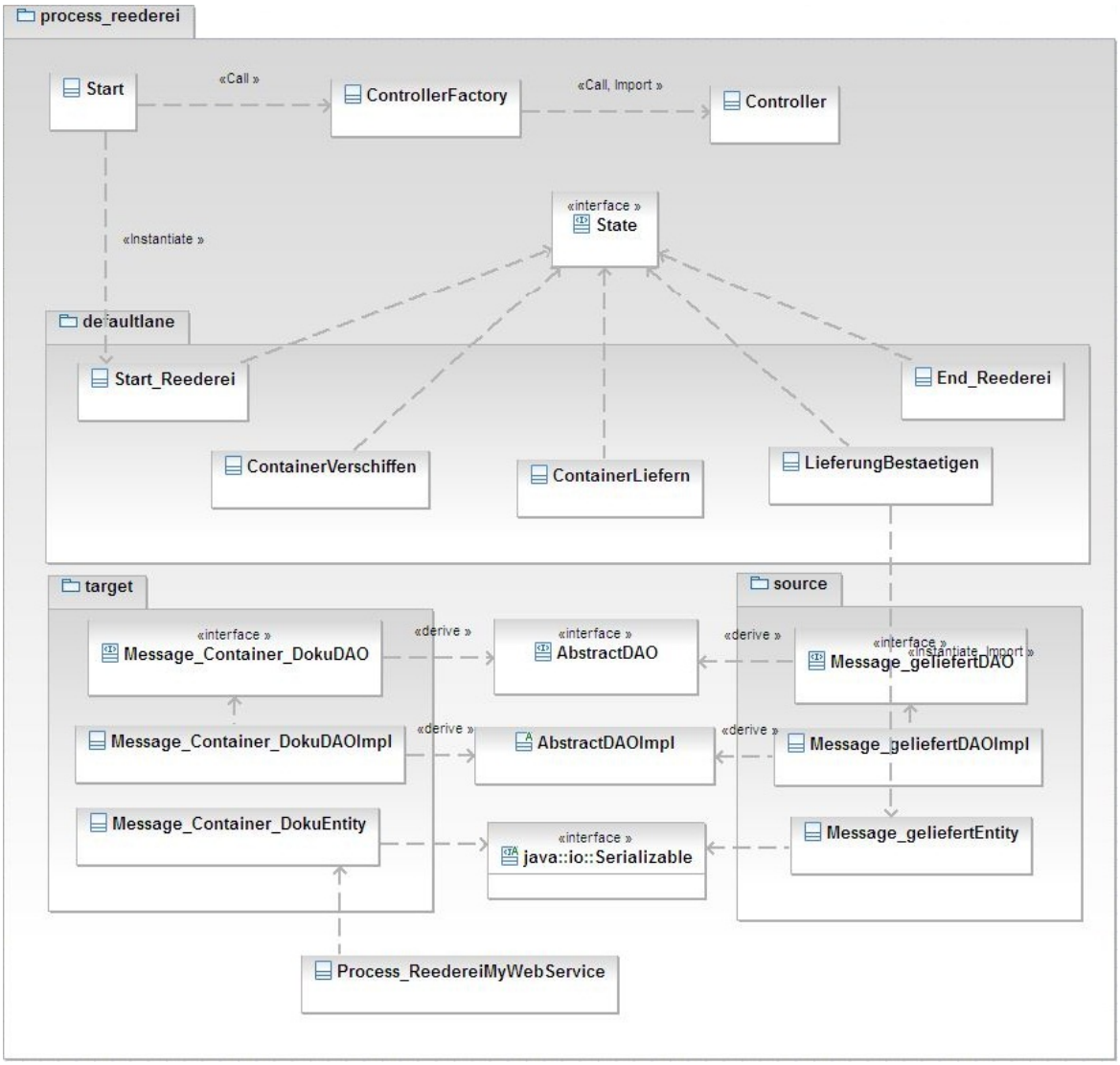


Abbildung 6.4.: Prozess: Reederei

Der zweite Testfall wurde der Literatur von Thomas Allweyer entnommen. Er stellt einen Prozess zur Stellenausschreibung zwischen einer Fach- und Personalabteilung dar. Es werden in diesem Prozess lediglich Elemente der Spezifikationsteilmenge 2.1.5.2 der BPMN 2.0 verwendet.

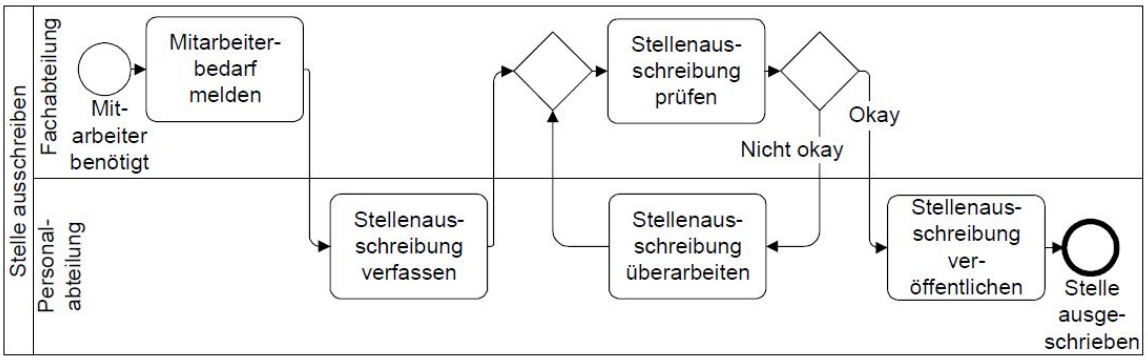


Abbildung 6.5.: Stellenausschreibung [Allweyer (2009b)]

Der Testfall soll zeigen, dass andere BPMN-Diagramme ebenfalls verwendet werden, mit Ausnahme des Logistikprozesses. Für das BPMN-Diagramm 6.5 wird der Java-Sourcecode generiert. Der Java-Sourcode wird vom Autor wieder als UML-Diagramm dargestellt 6.6. In diesem UML-Diagramm sind die generierten Methoden ebenfalls aufgeführt.

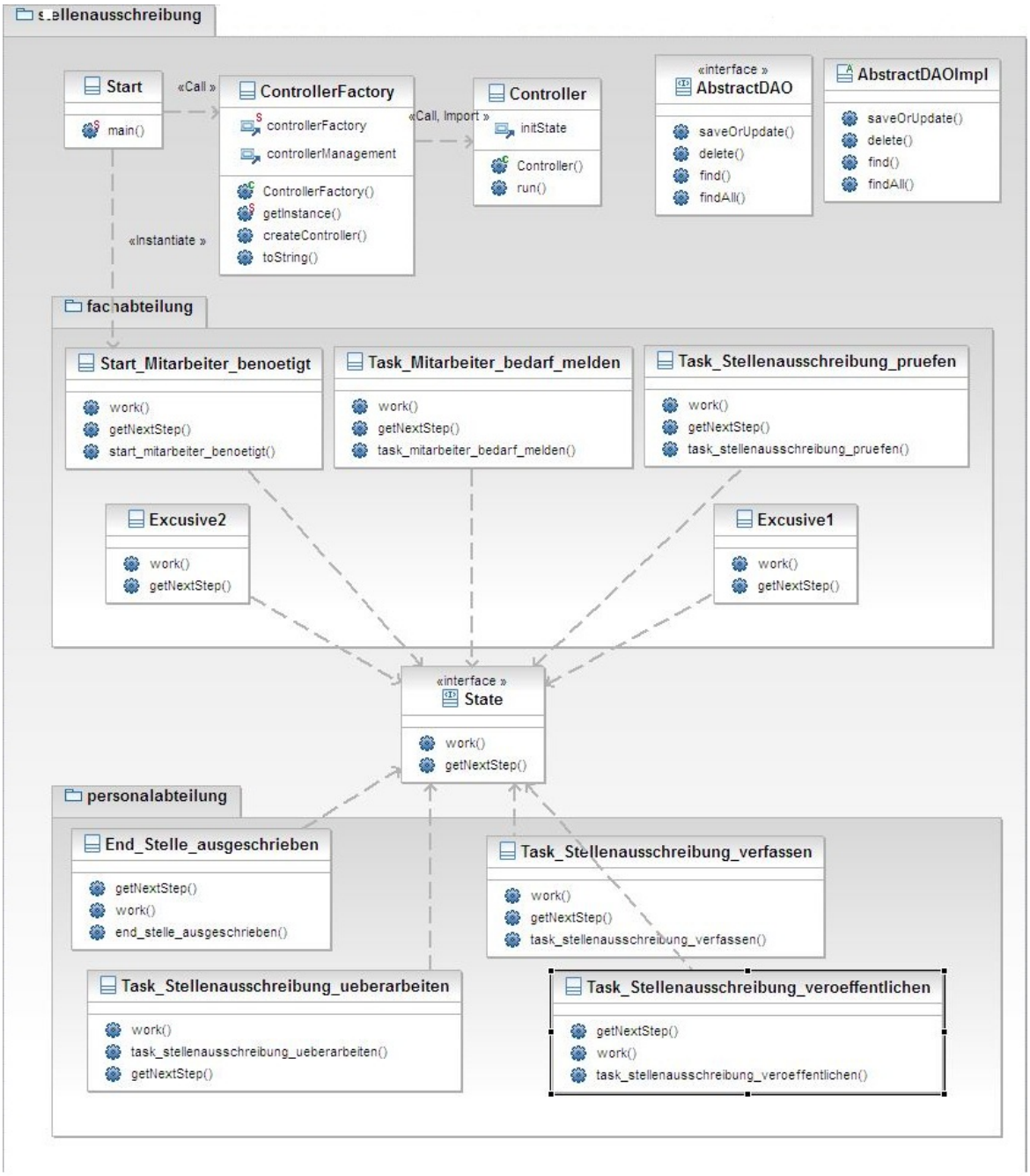


Abbildung 6.6.: Abbildung der Stellenausschreibung als UML-Diagramm

7. Zusammenfassung und Ausblick

Das abschließende Kapitel dieser Masterarbeit fasst alle bisherigen Ergebnisse zusammen und bietet einen Ausblick auf die Weiterentwicklung dieser Arbeit.

7.1. Zusammenfassung

MDA definiert Technologien, die ein Höchstmaß an Flexibilität in der Softwareentwicklung ermöglichen sollen. Dieses Ziel soll erreicht werden unter Verwendung von Ableitungen bzw. der Generierung von Codes aus stabilen Modellen. Der Fokus wird auf die Modellierung anstatt des manuellen Programmierens gerichtet. Die Komplexität soll reduziert und die Wiederverwendbarkeit von Systemen mit den Technologien von MDA erhöht werden. Des Weiteren wird durch die Automatisierung zum einen die manuelle Arbeit reduziert, und zum anderen soll durch die Automatisierung das Einbauen von Fehlern vermindert werden. Mit MDA-Konzepten sollen aus modellierten Geschäftsprozessen Java-Anwendungen erzeugt werden. Für die Modellierung von Geschäftsprozessen eignet sich die bewährte und weitverbreitete Modellierungssprache BPMN. Vor diesem Hintergrund wurden zu Beginn dieser Arbeit Ziele für die Konzeption und Realisierung des automatisierten Java-Codegenerators, basierend auf BPMN 2.0, definiert, um aus BPMN-Modellen automatisch Java-Sourcecode zu erzeugen. Der Nutzen des Generators liegt darin, die vorliegenden Informationen im BPMN 2.0 Modellen effektiv zu nutzen. Dementsprechend soll automatisch so viel Java-Sourcecode generiert werden, wie Informationen aus den BPMN 2.0 Modellen extrahiert werden können. Eine Zielsetzung dabei ist unter anderem eine Erhöhung der Performance in der Entwicklung, der Qualität der Software etc. Das Resultat ist die prototypische Entwicklung eines BPMN-to-Java-Sourcecode-Generators (B2JSG). Die Vorgehensweise dazu wird an dieser Stelle zusammengefasst:

In dieser Arbeit wurden die Themengebiete der BPMN 2.0 und MDA untersucht. Zu Beginn wurde eine kurze Historie über BPMN 2.0 gewährt. Es wurde in diesem Zusammenhang festgestellt, dass einige Autoren (Bruce Silver und Jakob Freund) BPMN unabhängig von der OMG klassifizieren. Die Klassifikation legt den Detaillierungsgrad fest. Diese Erkenntnis konnte der Autor für die Umsetzung des B2JSG verwenden. Je detailreicher das System,

desto präzisier ist der zu generierende Code. Die Grundlagen der BPMN 2.0 wurden beispielhaft mit einem Logistikprozess in Verbindung gebracht. Anhand dieses Beispiels wurde eine Spezifikationsteilmenge der BPMN 2.0 Elemente bestimmt. Zu den Grundlagen gehören ebenfalls die MDA-Konzepte, da einige Codegeneratoren die MDA-Konzepte nutzen.

Anschließend folgte eine Anforderungsanalyse. Es sollte ein System entwickelt werden, das aus einem BPMN 2.0 Modell Java-Sourcecode erzeugt. An dieser Stelle wurde eine Systemidee entwickelt, deren technische und fachliche Anforderungen aufgestellt wurden. Anschließend folgte eine Anforderungsanalyse. Es sollte ein System entwickelt werden, das aus einem BPMN 2.0 Modell Java-Sourcecode erzeugt. An dieser Stelle wurde eine Systemidee entwickelt, deren technische und fachliche Anforderungen aufgestellt wurden. Diese legten die Rahmenbedingungen für die später folgende Konzeption und Realisierung fest.

Der zu entwickelnde B2JSG ist lediglich für die Codegenerierung zuständig. Er benötigt allerdings ein Modellierungstool, mit dessen Unterstützung BPMN-Diagramme erstellt werden. Dabei kann es sich um ein kostenloses Modellierungstool einschließlich Open-Source-Tools handeln. Die Marktanalyse für BPMN-Modellierungstools zeigte, dass die meisten BPMN-Modellierungstools entweder nicht ausgereift sind oder kein BPMN 2.0 unterstützen. Gemäß der vom Autor aufgestellten Kriterien erfüllt BizAgi als einziges Modellierungstool von BPMN-Diagrammen alle Kriterien. Dabei soll BizAgi den Input für den Codegenerator B2JSG liefern. Anschließend folgte eine Marktanalyse und eine Analyse der existierenden BPMN-Codegeneratoren. Die Marktanalyse ergab, dass keine BPMN-to-Java-Codegeneratoren verfügbar sind. Allerdings existieren Ansätze, die sich für die BPMN-Codegenerierung eignen. Im Zusammenhang mit den existierenden Ansätzen wurden zwei Plattformen für die Codegenerierung vorgestellt (AndroMDA 4.3.4 und oAw 4.3.2). Aufgrund der aufgestellten Kriterien fiel die Wahl auf die oAW-Plattform.

Auf Basis der gesammelten Erkenntnisse wurden folglich die Realisierungskonzepte und die Architektur ausgearbeitet. Der Autor entschied sich für die Model zu Code Transformation aufgrund der oAW-Plattform. Entscheidend waren außerdem das SAP-Metamodell.

Der Autor richtete den Fokus bei der Realisierung auf die Kernaufgabe, nämlich auf das Generieren des Java-Sourcecodes bzw. das Abbilden der BPMN 2.0 Elemente in Java. Die Realisierung des Parsens eines externen Modells und die anschließende Übersetzung in das mit dem verwendeten Metamodell kompatible XMI-Format waren nicht der wesentliche Bestandteil dieser Masterarbeit. Der Autor fand lediglich ein BPMN 2.0 Metamodell in ECORE-Format, das von SAP zur Verfügung gestellt wird. Dieses Metamodell von SAP umfasst ebenfalls ein zu dem Metamodell kompatibles Modellierungstool. Das SAP-Modellierungstool bietet lediglich die textuelle Modellierung, BizAgi dagegen die grafische Modellierung. Aufgrund dessen wurde das SAP-Modellierungstool nicht in der Marktanalyse 4.1.6 aufgeführt, da es den Kriterien nicht entspricht. Dieses Tool dient trotzdem als Workaround für die prototypische Entwicklung des B2JSG, um die Realisierung des Parsens und des Mapping-Vorgangs

zu umgehen. Das Metamodell von SAP war mit geringem Aufwand in das OAW-Plattform integrierbar. In Anschluss folgte die Erstellung der geeigneten Architektur, die von der oAW-Plattform geprägt ist.

Die Kernaufgabe dieser Arbeit ist das Generieren des Java-Sourcecodes, dementsprechend das Abbilden der BPMN 2.0 Elemente in Java. Zu diesem Zweck wurde das entsprechende Konzept für diese Abbildungen der Spezifikationsteilmenge erstellt und Diskussionen über diese Thematik durchgeführt. Diese Diskussionen umfassten Konzeptideen des Autors und existierende Ansätze. Die existierenden Ansätze waren nur in einer beschränkten Anzahl vorhanden. Des Weiteren wurde in diesem Zusammenhang auf die Problematiken eingegangen. Wie bereits erwähnt, beschäftigt sich die Realisierung im Wesentlichen mit der Sourcecode-Generierung und infolgedessen mit den Abbildungen der BPMN 2.0 Elemente in Java.

Für die Realisierung gelangten einige Technologien zum Einsatz. Hierbei wurden die favorisierten Technologien für den Codegenerator (B2JSG) vorgestellt. Aufgrund der Auswahl der oAW-Plattform waren teilweise keine Ausweichmöglichkeiten auf andere Technologien möglich. Dadurch sind jedoch keine Nachteile bemerkbar. Die oAW-Plattform wurde in die Eclipse-Entwicklungsplattform integriert. Einige Anforderungen, die in den „Fachlichen Anforderungen“ 3.1.2 der Anforderungsanalyse aufgestellt wurden, wurden dadurch automatisch erfüllt.

In den Testfällen wurden die Fähigkeiten des Prototypen getestet. Es war relevant zu zeigen, dass der zu erwartende Java-Sourcecode generiert wurde, und dass sich des Weiteren der Generator nicht lediglich auf das Fallbeispiel 2.3 beschränkt. Der Generator soll für unterschiedliche BPMN-Diagramme ebenfalls den zu erwartenden Java-Sourcecode generieren.

7.2. Ausblick

Der zu entwickelnde BPMN-to-Java-Codegenerator (B2JSG) ist ein Prototyp. Bezüglich einer Teilmenge von BPMN 2.0 Spezifikationselementen 2.1.5.2 sind seine Fähigkeiten beschränkt. Er befindet sich in einem einsatzbaren Zustand. Jedoch können lediglich für BPMN-Diagramme, die aus Elementen der Spezifikationsteilmenge bestehen, Java-Sourcecodes erzeugt werden. Für die Nutzung des gesamten Funktionsumfangs sind Weiterentwicklungen und Erweiterungen notwendig. Dabei müssen alle Elemente der Spezifikationsmenge in den Templates der oAW-Plattform abgebildet werden. Zudem können weitere Überlegungen angestellt werden, ob nicht zusätzliche Aspekte, basierend auf den BPMN 2.0 Elementen, abgebildet werden.

Das SAP-Modellierungstool 5.1.11 ermöglicht die textuelle Modellierung eines BPMN-Diagramms in BPMN 2.0. Allerdings ist die textuelle Modellierung sehr zeitaufwändig, gegenüber der grafischen Modellierung beispielweise mittels BizAgi 4.1.4. Da ein wesentliches Ziel dieser Masterarbeit die Codegenerierung war, wurden aus zeitlichen Gründen die BPMN-Diagramme mit Hilfe des SAP-Modellierungstools erstellt. Für die Modellierung mit BizAgi wäre die Weiterentwicklung der Mapping-Komponente 5.1.6 notwendig gewesen. Unter Verwendung des SAP-Metamodells ist die Übersetzung des BizAgi-BPMN-Diagramms in XMI erforderlich. Allerdings wurde dies in der Architektur berücksichtigt. Aus zeitlichen Gründen wurde auf die Weiterentwicklung dieser Komponente verzichtet.

Literaturverzeichnis

[Abts 2010] ABTS, Dietmar: *Grundkurs Java, Von den Grundlagen bis zu Datenbank- und Netzanwendungen*. Vieweg+Teubner, 2010. – ISBN ISBN 978-3-8348-1277-3

[Allweyer 2009a] ALLWEYER, Prof. Dr. T.: *Kollaborationen, Choreographien und Konversationen in BPMN 2.0*. 2009

[Allweyer 2008] ALLWEYER, Thomas: *Einführung in den Standard für die Geschäftsprozessmodellierung*. Books on Demand GmbH, 2008. – ISBN 978-3-8370-7004-0

[Allweyer 2009b] ALLWEYER, Thomas: *BPMN 2.0 Business Process Model and Notation, Einführung in den Standard für die Geschäftsprozessmodellierung*. Books on Demand GmbH, 2009. – ISBN ISBN 978-3-8391-2134-4

[AndroMDA 2011] ANDROMDA: *Generate components quickly with AndroMDA*. http://www.andromda.org. 2011

[Anne Gross 2009] ANNE GROSS, Joerg D.: *EPC vs. UML Activity Diagram - Two Experiments Examining their Usefulness for Requirements Engineering*. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05328644. 2009

[Anneke Kleppe 2004] ANNEKE KLEPPE, Wim B.: *EXPLANINED THE MODEL DRIVEN ARCHITECTURE*. ADDISON-AESLEY, 2004. – ISBN ISBN 032119442X

[Apache 2011a] APACHE: *The Apache Software Foundation, Axis 2*. http://axis.apache.org/axis2/java/core/. 2011

[Apache 2011b] APACHE: *The Apache Velocity Project*. http://velocity.apache.org/. 2011

[ARIS-Community 2010] ARIS-COMMUNITY: *BPMN Process Modeling & Free Modeling Tool*. http://www.ariscommunity.com/aris-express/bpmn-2-free-process-modeling-tool. 2010

[BizAgi 2010] BIZAGI: *BizAgi Documentation Center*. http://wiki.bizagi.com/en/index.php?title=Main_Page. 2010

[BizAgi+ICO 2010] BIZAGI+ICO: *The Spanish government reacts with agility faced with the crisis*. http://www.bizagi.com/docs/BizAgi+ICO.pdf. 2010

[Borland 2011] BORLAND: *TRANSFER SOFTWARE DELIVERY INTO A MANAGED BUSINESS PROCESS*. http://www.borland.com. 2011

[BPMS-Designer] BPMS-DESIGNER: *BPMS is the world's most widely deployed Business Process Management System*. http://www.intalio.com/bpms

[Breno Lisi Romano und Mourão 2010] BRENO LISI ROMANO, Adilson Marques da C. ; MOURÃO, Walter I.: *Applying MDA development approach to a Hydrological Project*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5501481. 2010

[Calafato 2010] CALAFATO, Aaron: *Extending WISE with Contract Management*. http://staff.um.edu.mt/cabe2/supervising/undergraduate/overview/aaron_calafato.pdf. 2010

[Ching-Hong Tsai und Wang 2007] CHING-HONG TSAI, How-Jen L. ; WANG, Feng-Jian: *Constructing a BPM Environment with BPMN**. Uni-Stuttgart - http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4144627. 2007

[CHUN OUYANG 2009] CHUN OUYANG, WIL M. P. VAN DER A.: *From Business Process Models to Process-Oriented Software Systems*. ACM - http://portal.acm.org/citation.cfm?id=1555395&coll=Portal&dl=GUIDE&CFID=108980507&CFTOKEN=71260178&ret=1#Fulltext. 2009

[Community 2011] COMMUNITY, JBoss: *Relational Persistence for Java and .NET*. http://www.hibernate.org/. 2011

[Darwin 2005] DARWIN, Ian F.: *Java Kochbuch*. O'Reilly, 2005. – ISBN ISBN 3-89721-400-8

[Dieter Eickstädt 2004] DIETER EICKSTÄDT, Thomas R.: *J2EE mit Struts & Co., Java-Projekte mit Struts, Tomcat, Jboss und Eclipse*. Markt+Technik, 2004. – ISBN ISBN 3-8272-6680-7

[E.-E. Doberkat 2002] E.-E. DOBERKAT, S. D.: *Einführung in die objektorientierte Programmierung mit Java*. Oldenbourg, 2002. – ISBN ISBN 3-486-25342-5

[Eclipse 2010] ECLIPSE: *Eclipse-Downloads*. http://www.eclipse.org/downloads/. 2010

[Eclipse 2011] ECLIPSE: *Explore the Eclipse universe...* http://www.eclipse.org/. 2011

[Eclipse-EMF 2005] ECLIPSE-EMF: *The Eclipse Modeling Framework (EMF) Overview*. http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.emf.doc/references/overview/EMF.html. 2005

[Eclipse-GEF 2005] ECLIPSE-GEF: *Using GEF with EMF*. http://www.eclipse.org/articles/Article-GEF-EMF/gef-emf.html. 2005

[Eclipse-GMF 2005] ECLIPSE-GMF: *Graphical Modeling Project (GMP)*. http://www.eclipse.org/modeling/gmp/. 2005

[Eclipse-MDT] ECLIPSE-MDT: *Model Development Tools (MDT)*. http://www.eclipse.org/projects/dev_process/development_process_2010.php#6_2_3_Incubation

[Eisenberg 2002] EISENBERG, J. D.: *SVG Essentials: Producing Scalable Vector Graphics with XM*. O' Reilly, 2002. – ISBN 0-596-00223-8

[EMF 2011a] EMF: *Eclipse Modeling Framework Project (EMF)*. http://www.eclipse.org/modeling/emf/. 2011

[EMF 2011b] EMF: *Introducing EMF*. http://ptgmedia.pearsoncmg.com/images/0131425420/samplechapter/budinskych02.pdf. 2011

[Engineering 2011] ENGINEERING, Peak: *Peak Engineering, High Level Software Tools*. http://www.peakengineering.de/Objecteering_UML/JavaDeveloper/javadeveloper.html. 2011

[Erich Gamma 2004] ERICH GAMMA, Ralph J.: *Entwurfsmuster*. Addison-Wesley, 2004. – ISBN ISBN: 978-3-8273-2199-2

[Flanagan 2005] FLANAGAN, David: *JAVA IN A NUTSHELL*. O'Reilly, 2005. – ISBN ISBN 3-89721-332-X

[Frankel 2003] FRANKEL, David S.: *Model Driven Architecture, Applying MDA to Enterprise Computing*. Wiley, 2003. – ISBN ISBN 0-471-31920-1

[Giso Bartels 2007] GISO BARTELS, Marco V.: *Vergleich von BPMN-Modellierwerkzeugen*. Uni-Stuttgart - http://elib.uni-stuttgart.de/opus/volltexte/2007/3385/pdf/FACH_0075.pdf. 2007

[Grass 2010] GRASS, Tim Weilkiens Christian Weiss A.: *Basiswissen Geschäfsprozessmanagement*. dpunkt.verlag, 2010. – ISBN 978-3-89864-647-5

[Group 2010] GROUP, OMG Object M.: *Business Process Model and Notation (BPMN)*. OMG - http://www.omg.org/spec/BPMN/2.0/Beta2/PDF. 2010

[Hille-Doering 2010] HILLE-DOERING, Reiner: *BPMN 2.0 Metamodel Implementation for Eclipse: Get it and Use it*. http://www.sdn.sap.com/irj/sdn/nw-process-modeling. 2010

[Hosieny 2011] HOSIENY, Julia: *Ein Framework zur automatisierten Fehlererkennung und -lokalisierung für Java-Anwendungen*, University of Aplied Sciences (HAW-Hamburg), Masterarbeit, 2011

[b+m Informatik AG 2011] INFORMATIK AG b+m: *b+m business IT management*. http://www.bmiag.de/. 2011

[Ismael Ghalimi 2006] ISMAEL GHALIMI, Hugues M.: *Eclipse BPMN Modeler - Introducing Intalio Designer*. http://www.eclipsecon.org/summiteurope2006/presentations/ESE2006-EclipseModelingSymposium1_BPMS&Eclipse.pdf. 2006

[Jakob Freund 2008] JAKOB FREUND, Klaus G.: *Vom Geschäftsprozess zum Workflow: Ein Leitfaden für die Praxis*. Hanser, 2008. – ISBN ISBN 978-3-446-41482-2

[Jakob Freund 2010] JAKOB FREUND, Thomas H.: *BPMN PRAXISHANDBUCH*. Hanser, 2010. – ISBN ISBN 978-3-446-41768-7

[James Gosling 2005] JAMES GOSLING, Gilad B.: *The Java Language Specification*. Addison-Wesley, 2005. – ISBN ISBN 0-321-24678-0

[Jan Recker 2005] JAN RECKER, Marta Indulska Peter G.: *Do Process Modelling Techniques Get Better, A Comparative Ontological Analysis of BPMN*. CiteSeerX - http://eprints.qut.edu.au/2879/1/Recker_et_al-ACIS2005b.pdf. 2005

[JAX-WS 2.0 2011] JAX-WS 2.0, Oracle C.: *Introducing JAX-WS 2.0 With the Java SE 6 Platform*. http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/. 2011

[JDOM 2011] JDOM: *Build a better mousetrap, and the world will beat a path to your door*. http://www.jdom.org/. 2011

[Kleuker 2009] KLEUKER, Stephen: *Grundkurs Software-Engineering mit UML*. Vieweg+Teubner, 2009. – ISBN 978-3-8348-0391-7

[Limited 2006] LIMITED, Enix C.: *BPM Focus An Independent Evaluation Of BizAgi*. http://www.bizagi.com/docs/BPM%20Focus%20talks%20about%20BizAgi.pdf. 2006

[Matteo Bordin 2005] MATTEO BORDIN, Tullio V.: *Automated Model-based Generation of Ravenscar-compliant Source Code*. http://staff.um.edu.mt/cabe2/supervising/undergraduate/overview/aaron_calafato.pdf. 2005

[Matthias Regensburger 2007] MATTHIAS REGENSBURGER, Alois Knoll Gerhard S.: *Model Based Development of Safety-Critical Systems Using Template Based Code Generation*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4459643&tag=1. 2007

[Müller-Lindenberg 2005] MÜLLER-LINDENBERG, Matthias: *Führung in zeitkritischen und komplexen Projekten*. duv, 2005. – ISBN 978-3-8244-8294-8

[Nasi Tantitharanukul 2009] NASI TANTITHARANUKUL, Prompong S.: *Detecting Deadlock and Multiple Termination in BPMN Model Using Process Automata*. ACM - http://portal.acm.org/results.cfm?coll=Portal&dl=GUIDE&CFID=108980507&CFTOKEN=71260178. 2009

[O. Vogel 2009] O. VOGEL, A. C.: *Software Architektur, Grundlagen - Konzepte - Praxis*. Spektrum, 2009. – ISBN ISBN 978-3-B274-1933-0

[OAW 2008] OAW: *openArchitectureWare User Guide*. http://www.openarchitectureware.org/pub/documentation/4.3.1/openArchitectureWare-4.3.1-Reference.pdf. 2008

[oAW 2011] OAW: *openArchitectureware has moved to the Eclipse Modeling Project*. http://www.openarchitectureware.org. 2011

[OMG 2011] OMG: *OMG Model Driven Architecture*. http://www.omg.org/mda/. 2011

[OMG]

[Oracle 2011] ORACLE: *Oracle und Java*. http://www.oracle.com/de/technologies/java/index.html. 2011

[Peter Roßbach 2003] PETER ROSSBACH, Wolfgang N.: *Model Driven Architecture, Grundlegende Konzepte und Einordnung der Model Driven Architecture (MDA)*. http://it-republik.de/jaxenter/artikel/Model-Driven-Architecture-0408.html?print=1. 2003

[Ramin Nasiri 2009] RAMIN NASIRI, Sahar Yousefi B.: *Development of Error Management in BPMN Using Java Code*. http://portal.acm.org/citation.cfm?id=1725447. 2009

[Reiberg 2006] REIBERG, Daniel: *VERTEILTE ENTERPRISE APPLIKATIONEN AUF BASIS VON J2EE, JBOSS & ECLIPSE*. Hanser, 2006. – ISBN ISBN-13: 978-3-446-40508-0

[Remco M. Dijkman a 2008] REMCO M. DIJKMAN A, c Chun O.: *Semantics and analysis of business process models in BPMN*. ACM - http://portal.acm.org/results.cfm?coll=Portal&dl=GUIDE&CFID=108980507&CFTOKEN=71260178. 2008

[Roland Petrasch 2006] ROLAND PETRASCH, Oliver M.: *Model Driven Architecture*. dpunkt.verlag, 2006. – ISBN ISBN 3-89864-343-3

[Rumpe 2005] RUMPE, Bernhard: *Agile Modellierung mit UML Codegenerierung, Testfälle, Refactoring*. Springer, 2005. – ISBN ISBN 3-540-20905-0

[Schuljak 2010] SCHULJAK, Manfred: *Diplomarbeit Konzeption und prototypische Implementation einer Rich Internet Application zur Modellierung von Human(Workflow)-Gesch¨aftsprozessen*. http://gcc.uni-paderborn.de. 2010

[Sherry Shavor 2004] SHERRY SHAVOR, Scott F.: *Eclipse, Anwendungen und Plug-Ins mit Java entwickeln*. Addison-Wesley, 2004. – ISBN ISBN 3-8273-2125-5

[Silver 2009] SILVER, Bruce: *BPMN Modeling and Style*. Cody-Cassidy Press, 2009. – ISBN 978-0-9823681-0-7

[Sommerville 2007] SOMMERVILLE, Ian: *Software Engineering, Pearson Studium*. Pearson Studium, 2007. – ISBN ISBN-13: 978-3827372574

[Starke 2009] STARKE, Dr. G.: *Effektive Software Architekturen*. Hanser, 2009. – ISBN 978-3-446-42008-3

[Stephen A. White 2004] STEPHEN A. WHITE, IBM C.: *Introduction to BPMN*. bpmn.org - http://www.bpmn.org/Documents/Introduction_to_BPMN.pdf. 2004

[Stephen A. White 2008] STEPHEN A. WHITE, PHD Derek M.: *MN Modeling and Reference Guide: Understanding and Using BPMN*. Future Strategies Inc., 2008. – ISBN 13:978-0-9777527-2-0

[Sven Efftinge 2006] SVEN EFFTINGE, Clemens K.: *OpenArchitectureWare 4.1 Xpand Language Reference*. http://www.openarchitectureware.org/pub/documentation/4.1/r20_xPandReference.pdf. 2006

[Thomas O. Meservy 2005] THOMAS O. MESERVY, Kurt D. F.: *Transforming Software Development: An MDA Road Map*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1510571. 2005

[Thomas Stahl 2005] THOMAS STAHL, Markus V.: *Modellgetriebene Softwareentwicklung*. dpunkt.verlag, 2005. – ISBN ISBN 3-89864-310-7

[Thomas Stahl 2007] THOMAS STAHL, Sven Efftinge Arno H.: *Modellgetriebene Softwareentwicklung*. dpunkt.verlag, 2007. – ISBN ISBN 978-3-89864-448-8

[Thome 2006] THOME, Rainer: *Grundzüge der Wirtschaftsinformatik*. Pearson Studium, 2006. – ISBN ISBN 976-3-8273-7221-5

[Tidwell 2002] TIDWELL, Doug: *XSLT, XML-Dokumente transformieren*. O'Reilly, 2002. – ISBN ISBN 3-89721-292-7

[Ujorm 2011] UJORM: *Ujorm Core, the powerful object architecture*. http://ujoframework.org/. 2011

[UML 2011] UML: *UNIFIED MODELING LANGUAGE*. http://www.omg.org/UML/. 2011

[X. Blanc 2000] X. BLANC, J. Le D.: *A Comparison of the Basic DO Concepts in Standardization*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=874192&userType=inst. 2000

[Y.CHE 2008] Y.CHE, B.Y.REN: *Research on Application of Model-Driven Architecture in the Development Process of Enterprise Information System*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4681067. 2008

[Zöller-Greer 2002] ZÖLLER-GREER, Peter: *Softwareeengineering für Ingenieure und Informatiker*. Vieweg Verlag, 2002. – ISBN 3-528-03939-6

A. Anhang

Im Wurzelverzeichnis der beigefügten DVD befindet sich eine PDF-Datei, diese enthält die Masterarbeit in PDF-Format. Neben alle verwendeten Tools befindet sich auch der Prototyp (B2JSG) auf der DVD. Der Prototyp ist in einem Zip-Archiv Prototyp.zip enthalten. Nach dem Entpacken des Archivs, sind die weiteren Schritte in der enthaltenen setup.txt beschrieben.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 13. Mai 2011	
Ort, Datum	Unterschrift