Institut für Rechnergestützte Ingenieursysteme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 149

Automatische Modellerzeugung aus klassifizierten und modellbasierten Anforderungen

Lan Jiang

Studiengang:	Informatik
Prüfer/in:	Univ-Prof. Hon-Prof. Dr. Dieter Roller
Betreuer/in:	Dipl. -Inf. Akram Chamakh
Begonnen am:	18. Juni 2014
Beendet am:	18. Dezember 2014
CR-Nummer:	D.2.1, H.2.8, I.2.5, I.2.7

Abstract

The elicitation and analysis of the requirements is a key phase of a technical project. Incorrect requirements for complex software products can lead to a serious consequence. A widespread procedure is the requirement elicitation from natural language texts. On the one hand, the procedure can offer a good understandability of natural language. On the other hand, the ambiguities of natural language are not to avoid. Requirements analysis using natural language texts is carried out in some areas, where model-based approaches are used. Therefore the connection between requirements analysis and design is not really possible.

In this study, a system UML Model Generator (UMG) was designed to generate models from classified and model-based requirements. With the models the requirements can be represented not only textually but also graphically. This improves the quality of the requirements analysis and enables a transition from requirements analysis to requirements design. The ambiguity of natural languages will also be avoided.
The UML Model Generator (UMG) system offers the functionality of the classification based on an existing Requirement Extractor and Classifier (REC) system from the thsis [Zwi13], which extracts and classifies requirements from technical specifications. The UMG system can extract functional requirements from a textual document and generate their models.
A data model was designed for UMG, which represents the extracted requirements in format of UML Model Generator XML (UMGX). Based on the data model a model-based requirements document can be generated, which includes all essential information about the generation of the models. A UMGX document contains use cases and their basic flows. As a basic concept of the UMG system, the data model allows any changes and extensions to the requirements and their models.

Kurzfassung

Die Erhebung und Analyse der Anforderungen ist eine Schlüsselphase eines technischen Projekts. Schlechte oder fehlerhafte Anforderungen an komplizierte Softwareprodukte können zu einer schwerwiegenden Konsequenz führen.
Ein verbreitetes Verfahren ist die Erhebung der Anforderungen aus natürlich sprachlichen Texten. Einerseits kann das Verfahren die gute Verständlichkeit natürlicher Sprache bieten, andererseits lassen sich dabei die Mehrdeutigkeiten natürlicher Sprache nicht vermeiden. Außerdem wird in manchen Bereichen, bei denen modellbasierte Ansätze zur Anwendung kommen, die Anforderungsanalyse unter Verwendung von natürlich sprachlichen Texten durchgeführt. Dadurch ist die Verbindung zwischen Anforderungsanalyse und Entwurf nicht so gut möglich.

In dieser Arbeit wurde ein System UML Model Generator (UMG) entworfen, das Modelle aus klassifizierten und modellbasierten Anforderungen generiert. Mit den Modellen können die Anforderungen nicht nur textuell sondern auch grafisch dargestellt werden. Die Qualität der Anforderungsanalyse wird dadurch verbessert und ein Übergang von der Anforderungsanalyse hin zum Entwurf wird ebenfalls ermöglicht und Mehrdeutigkeiten werden vermieden.
Das UMG System bietet die Funktionalität der Klassifizierung, die auf einem vorhandenen REC System aus der Diplomarbeit [Zwi13] basiert, welches Anforderungen aus technischen Spezifikationen extrahiert und klassifiziert. Zur Erzeugung der Modelle extrahiert UMG die funktionalen Anforderungen aus einem texutellen Dokument, welche Funktionen eines zu entwickelnden Systems beschreiben können.
Für UMG wurde ein Datenmodell entworfen, das die extrahierten Anforderungen in Format von UML Model Generator XML (UMGX) darstellt. Darauf aufbauend kann ein modellbasiertes Anforderungsdokument generiert, in dem alle essentiellen Informationen zur Erzeugung der Modelle von Anforderungen beinhaltet sind. Ein UMGX Dokument enthält Anwendungsfälle und gegebenenfalls zu diesen die Standardabläufe. Als ein grundsätzliches Konzept von UMG ermöglicht das Datenmodell beliebige Änderungen und Erweiterungen zu den Anforderungen und deren Modelle.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

1. Einleitung

Seit letzten Jahrzehnten spielt in fast allen Industriebereichen Software eine wichtige Rolle bei der Realisierung innovativer Funktionen und Services. Wegen der zunehmenden Einflüsse auf eine erfolgreiche Entwicklung der Produkte und Systeme lassen sich die Software sorgfältig behandeln und immer mehr Wert auf sich legen. Die Systeme, deren Eigenschaften durch Software verwirklicht werden, werden als softwarebasierte Systeme bezeichnet. Zu solchen Systemen gehören eingebettete Systeme und Informationssysteme.

Die Entwicklung softwarebasierter Systeme wird durch eine Reihe von Herausforderungen erschwert. [Poh08] hat darauf hingewiesen, dass sich die Herausforderungen aus zunehmender Komplexität, erhöhtem Kostendruck, reduzierter Entwicklungszeit sowie höheren Qualitätsansprüche zusammensetzen. Die Konsequenz dazu liegt jedoch an der Schwierigkeit, eine erfolgreiche Software zu realisieren. Zur Bewältigung der Herausforderungen bei der Softwareentwicklung wurde Requirements Engineering als Voraussetzung gesetzt.
In der deutschen Forschungsumfeld wurde der englische Begriff *Requirements Engineering* am meisten als Anforderungsanalyse oder -spezifikation bezeichnet. Allerdings beinhaltet *Requirements Engineering* im weiteren Sinn nicht nur den Bereich der Anforderungsanalyse sondern auch Anforderungsdefinition und -managment. Im engeren Sinn wird *Requirements Engineering* trotz seines weiten Umfangs dazu verwendet, die Tätigkeiten am Beginn eines Systemprojekts zu charakterisieren. Um die Umstände der Verwendung des Begriffs zu vermeiden, wird *Requirements Engineering* in dieser Arbeit als Anforderungsanalyse und -spezifikation charakterisiert.

In der Abbildung 1.1 wurde die Quote für erfolgreiche Projekts zwischen 1994 und 2009 gezeigt, dass der Anteil der erfolgreichen Projekte bis dem Jahr 2006 kontinuierlich gestiegen ist und im Jahr 2009 jedoch abgefallen ist. Die Gründe für den Abwärtstrend liegen den Experten nach an der mangelnden Analyse der Anforderungen, die von den Stakeholdern erhoben wurden. Weiterhin ist der Einfluss einer ausführlichen Anforderungsanalyse in der Abbildung 1.2 zu sehen, dass der höhere Aufwand für Requirements Engineering zu einer relativ niedrigen Kostenüberschreitung führen kann. Deswegen sind Requirements-Engineering sowie die Anforderungsanalyse eine Schlüsselphase in der Entwicklung softwaregestützter Systeme.

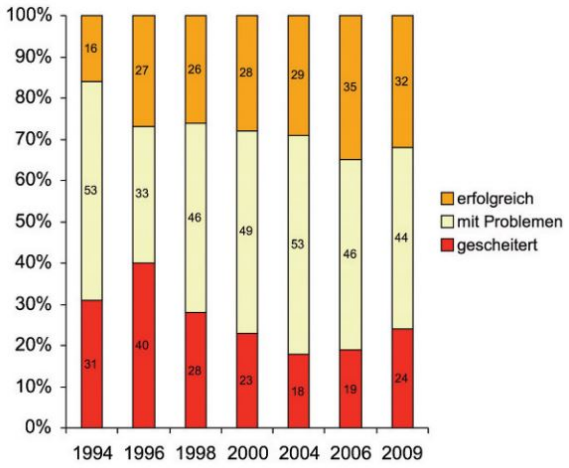

Abbildung 1.1.: Projekterfolgsquote [Sta09]

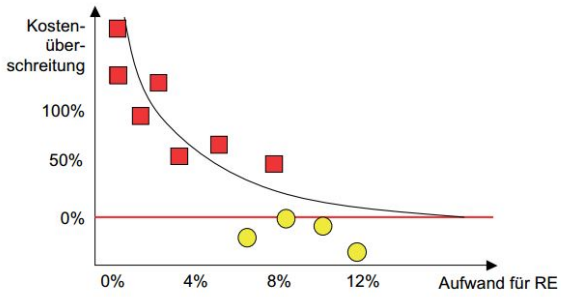

Abbildung 1.2.: Aufwand für Kostenüberschreitung und RE [Sta09]

Zur optimalen Erfüllung der Schlüsselphase werden heutzutage die Anforderungen an die zu entwickelnden Systeme in der Industrie größtenteils rein textuell formuliert. Die natürliche Sprache ermöglicht den Analysten eine gute Verständnis über die Anforderungen, welche die Basis der Entwicklungsprozesse eines Produkts oder Systems festlegen. Obwohl die textuellen Anforderungen leicht zu begreifen und zu nachvollziehen sind, verbergen sich trotzdem einige Probleme hinter den Textdokumenten, die den Prozess der Erhebung, Analyse, Spezifikation für Anforderungen sogar deren Weiterverarbeitungen negativ beeinflussen können.

Verschiedene Analysten verwenden bei der Erfassung der textuellen Anforderungsdokumenten unterschiedliche Begriffen und Formulierungen, die Doppelbedeutungen besitzen oder Unklarheit der Kommunikationen zwischen Beteiligten verursachen können. Außerdem sind rein textuelle Formulierungen schwer automatisch zu bearbeiten, dabei werden die Änderungen von Anforderungen oder die Verknüpfung mit Entwurf und Implementierung nicht optimal realisiert. Bei einem Anforderungsdokument mit großer Menge von Informationen und komplexen Sachverhalte verliert

man oft schnell den Überblick über das ganze System.

Neben den natürlichsprachigen Anforderungsdokumentationen können die Dokumentationen zusätzlich modellbasiert stattfinden. Die modellbasierte Entwicklung hat in manchen Bereichen vor allem in den eingebetteten Systemen immer mehr Aufmerksamkeiten bekommen. Die Verwendung von graphischen Modellierungssprachen bei der Anforderungsermittlung kann den Medienbruch zur Anwendungsentwicklung vermeiden und alle Projektbeteiligten sprechen die gleiche Sprache, in dem Fall die Modellierungssprache.
Durch die Modelle werden die Anforderungen an Systeme oder Produkte in einer einheitlichen Form dargestellt, die Zusammenhänge und Wechselbeziehungen zwischen allen Anforderungen werden dementsprechend graphisch illustriert. Aus den Modellen kann Code eventuell für die Entwicklungsphase generiert werden, der die Implementierung bei der Weitverarbeitung unterstützen kann.

Das Ziel dieser Arbeit ist es, textuelle Anforderungen automatisiert in die entsprechende modellbasierte Anforderungen umzuwandeln, damit die Modelle aus den erhaltenen Anforderungen als Ergebnis für die Weiterverarbeitung der Projektentwicklung zur Verfügung stehen können.

Die automatisierte Erzeugung der Modelle wird durch ein System namens *UML Model Generator (UMG)* von der Arbeit aufgrund der objektorientierten Methode realisiert. Zur Erfüllung der Voraussetzung der Umwandlung, dass nur funktionale Anforderungen modellbasiert dargestellt werden können, sollen die textuellen Anforderungen zunächst klassifiziert werden. Deswegen ist eine automatisierte Klassifikation für die zu analysierenden Anforderungen wichtig.

Für das Teilsystem, das die Funktionalität der Klassifizierung enthält, basiert diese Arbeit auf dem REC System von der Diplomarbeit [Zwi13]. In [Zwi13] wurde das REC System zur Klassifizierung der Anforderungen in Textform entwickelt, das ein verteilungsfreies Verfahren verwendet und die Klassifikation aufgrund der klaren Klassentrennung ermöglicht. Die dafür entwickelte Datenstruktur können auch domänenspezifischen Ontologien verwalten.
Zur Erweiterung und Verbesserung des REC Systems werden dem neuen UMG System in dieser Arbeit zusätzliche Features hinzugefügt, damit die Klassifizierung und Anforderungsumwandlung problemlos miteinander anpassen können.

Zur Modellerzeugung der Anforderungen werden Anwendungsfalldiagramme und Sequenzdiagramme durch das UMG System visualisiert. Ein Anwendungsfalldiagramm präsentiert die funktionalen Anforderungen in Form von Anwendungsfällen. Ein Sequenzdiagramm präsentiert den Standardablauf eines Anwendungsfalls, der aus einer funktionalen Anforderung generiert wird.
Der Kern des UMG Systems ist ein Datenmodell, das die Verwaltung der generierten Modelle ermöglichen kann. Für das Datenmodell bietet das UMG System ein Dateiformat *UML Model Generator XML (UMGX)*. In einem UMGX Dokument können alle Informationen des Datenmodells vollständig gespeichert und behalten.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Hier werden die Grundlagen dieser Arbeit beschrieben. In dem Kapitel werden die Klassifikation der Anforderungen, das Konzept Model Driven Requirements Engineering (MDRE) sowie die Modellierungssprache und die maschinellen Sprachverarbeitung beinhaltet vorgestellt.

Kapitel 3 – Vewandete Arbeiten und genutzte Technologien: Die verwandten Arbeiten werden hier betrachtet, dazu gehören die verwendeten Technologien, die für die Realisierung des UMG Systems verwendet werden.

Kapitel 4 – Lösungsansatz: Hier werden ein gewählter Lösungsansatz und ein alternativer Lösungsansatz vorgestellt. Eine Ausgangslage wird am Anfang des Kapitels vorgestellt.

Kapitel 5 – Implementierung: Vorstellung der Implementierung des UMG Systems. Dazu gehören eine Datenstruktur des Systems und der Programmablauf bei der Implementierung.

Kapitel 6 – Evaluierung: Die Ergebnisse, die das UMG System liefert, werden durch Evaluierungen behandelt.

Kapitel 7 – Zusammenfassung und Ausblick Die vorliegende Arbeit wird zusammengefasst und ein Ausblick über künftige Arbeiten wird vorgestellt.

2. Grundlagen

In diesem Kapitel werden die Grundlagen der Arbeit erläutert. Zuerst wird eine grundsätzliche Erklärung zu der Klassifikation der Anforderung vorgestellt. Danach werden das Konzept „MDRE“ einschließlich der modellbasierten Anforderungen und deren Vorteile erklärt. Darauf folgt eine allgemeine Beschreibung der Modellierungssprache „Unified Modeling Language (UML)“ . Schließend wird die maschinellen Sprachverarbeitung erläutert, die ebenfalls als Natural Language Processing (NLP) bezeichnet wird.

2.1. Klassifikation der Anforderungen

Eine Anforderung ist eine Aussage über die zu erfüllenden Eigenschaften eines Produkts oder Systems. Der Standard IEEE 610.12-1990 definiert den Begriff „Anforderung“ formal und präzise, die folgende deutsche Übersetzung der Definition stammt von [Poh08].
Eine Anforderung ist:

1. Eine Bedingung oder Eigenschaft, die ein System oder eine Person benötigt, um ein Problem zu lösen oder ein Ziel zu erreichen.
2. Eine Bedingung oder Eigenschaft, die ein System oder eine Systemkomponente aufweisen muss, um einen Vertrag zu erfüllen oder einem Standard, einer Spezifikation oder einem anderen formell auferlegten Dokument zu genügen.
3. Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft wie in 1 oder 2 definiert.

Im Sinne dieser Definition sind Anforderungen sowohl Wünsche und Ziele (Definition 1) von Benutzern als auch Bedingungen und Eigenschaften des zu entwickelnden Systems (Definition 2). Laut der Erläuterung zu der Definition 3 in [Poh08] werden auch eine dokumentierte Form von Bedingungen und Eigenschaften eines Systems als Anforderung bezeichnet. Jedoch wird eine dokumentierte Form in vielen technischen Spezifikationen nicht als Anforderungen bezeichnet, sondern als Anforderungsartefakt.
In der Technik sind Anforderungen für ein Entwicklungsprojekt von zentraler Bedeutung, sie werden in der Anforderungserhebung aufgenommen, analysiert, spezifiziert und verifiziert. Verschiedene Systeme erwarten verschiedene Funktionalitäten, mit denen die Anforderungen übereinstimmen müssen.

Um die Eigenschaften und Funktionalitäten des zu entwickelnden Systems als Anforderungen möglichst präzise und widerspruchsfrei zu erheben, sollen die Anforderungen in Klassen geteilt werden.

Die klassifizierten Anforderungen, die unterschiedliche Aspekte eines Systems beschreiben und dessen Eigenschaften oder Funktionalitäten einschränken, können in der weiteren Verarbeitung effizient verwendet werden, so dass die Entwicklung des Systems ständig unter der Kontrolle der Anforderungsanalyse steht.

[Poh08] hat Anforderungen in drei Klassen unterschieden, die Abbildung 2.1 fasst die Klassifikation der Anforderungen zusammen:

1. Funktionale Anforderungen
2. Qualitätsanforderungen
3. Rahmenbedingungen

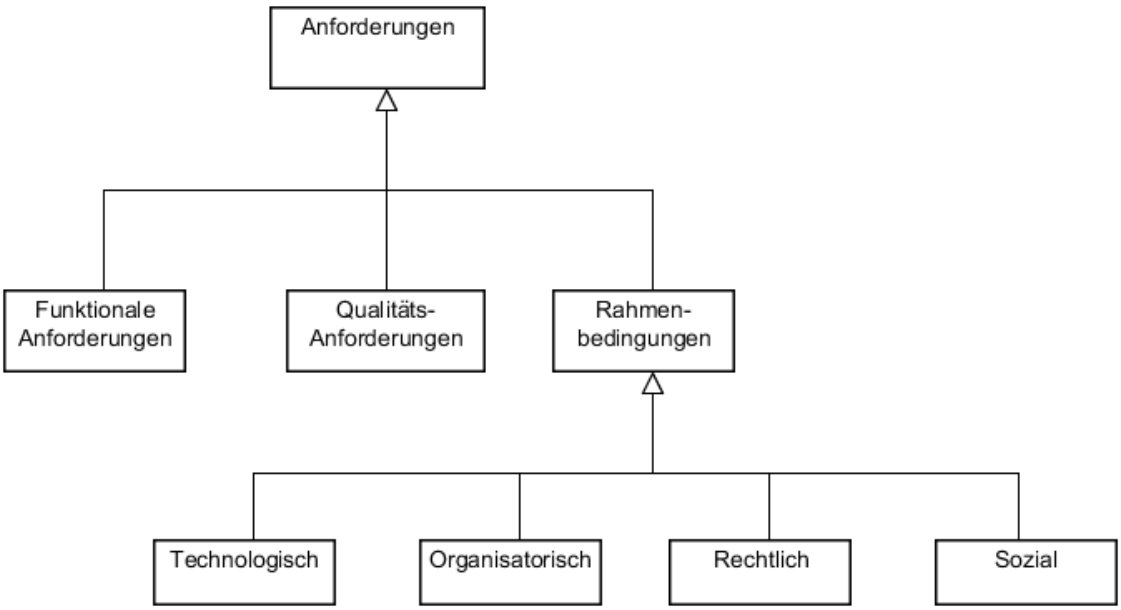

Abbildung 2.1.: Klassen der Anforderungen [Poh08]

2.1.1. Funktionale Anforderungen

In Anlehnung an [Som11] spezifizieren funktionale Anforderungen die Kernfunktionalitäten eines zu entwickelnden Softwaresystems oder deinen bereitzustellenden Service. Als Benutzeranforderung kann eine funktionale Anforderung allgemein beschrieben sein, während als Bestandteile einer Spezifikation eine funktionale Anforderung die Eingabe, Ausgabe sowie Ausnahmen im Detail beschreibt. Traditionell werden funktionale Anforderungen laut [Poh08] weiter in drei Perspektiven unterteilt. Der Ziele der Anforderungen gemäß bilden Daten, Funktionen und Verhalten die drei komplementären Perspektiven zur Beschreibung der funktionalen Anforderungen.

Dabei beziehen sich die Datenperspektive auf die statische Struktur der Daten. Datentyp, Attributen und Beziehungen zwischen den Datentypen gehören zu der statischen Struktur.
Die Funktionsperspektive betrachtet die Manipulation der Daten durch die Funktionen des Systems. Davon ausgehend hängt die Funktionsperspektive teilweise von der Datenperspektiven ab, indem die Daten als Eingabe durch Funktionen in Ausgabe transformiert werden.
Im Gegensatz zu der Datenperspektive ist die Verhaltensperspektive eine dynamische Struktur, die das Verhalten des Systems beschreibt. Bei der Perspektive werden die Reaktion des Systems auf externe Stimuli in Form von Zuständen betrachtet.

Die drei Perspektiven von der funktionalen Anforderungen werden in der Regel textuell innerhalb einer Anforderung dokumentiert. Modellbasiert werden sie jedoch unter Verwendung geeigneter Modellierungssprachen dokumentiert.

2.1.2. Nicht-funktionale Anforderungen

Neben der funktionalen Anforderungen sind noch Qualitätsanforderungen und Rahmenbedingungen von [Poh08] definiert. Am verbreitetsten ist jedoch die Unterteilung in funktionale und nicht-funktionale Anforderungen. Eine nicht-funktionale Anforderung stellt die Qualitätseigenschaften eines Systems dar, wobei die Qualitätsanforderungen und Rahmenbedingungen, die von [Poh08] separat unterschieden sind, zu dem Begriff „Qualitätseigenschaften“gehören. Daher sind die zwei Arten der Anforderungen in nicht-funktionalen Anforderungen beinhaltet. Zuerst werden die Qualitätsanforderungen erklärt.

Qualitätsanforderungen

Qualitätsanforderungen beschreiben die Qualitätsmerkmale eines Systems und definieren eine qualitative Funktion des gesamten Systems, einer Systemkomponenten oder einer Funktion. Die funktionalen Anforderungen können von Qualitätsanforderungen ergänzt werden.
[RR06] und [RG01] haben die Qualitätsanforderungen in typische Arten gegliedert. Die Tabelle 2.1 zählt die typischen Gliederungen der Qualitätsanforderungen in eine externe und eine interne Sichtweise. Aufgrund der beiden Sichtweisen beziehen sich Qualitätsanforderungen auf Entwicklung und Benutzer. Die Anforderungen aus Entwicklungssicht werden vor allem von der internen Sichtweise beschrieben. Hingegen spiegelt die externe Sichtweise die Anforderungen von Benutzern wider.

Externe Sicht	**Interne Sicht**
Performanz	Testbarkeit
Sicherheit	Wartbarkeit
Benutzbarkeit	Portierbarkeit

Tabelle 2.1.: Gliederungen der Qualitätsanforderungen [Ebe12]

Rahmenbedingungen

Die zweite Kategorie von nicht-funktionalen Anforderungen sind sogenannte Rahmenbedingungen, die in mancher Literatur (z.B. [Ebe12]) auch als Randbedingen bezeichnet werden. Eine Rahmenbedingung, die eine technische oder organisatorische Anforderung ist, beschreibt die Restriktionen an ein System. Rahmenbedingungen dienen zu Einschränkungen im Hinblick auf die Umsetzung von funktionalen Anforderungen und Qualitätsanforderungen. Die einschränkende Wirkung von Rahmenbedingungen können in extremen Fall die Anforderungen entweder auf keine Weise beschränken oder komplett entfernen.
Die typischen Arten der Rahmenbedingungen sind in der Abbildung 2.1 beschrieben, neben den vier Arten gibt es noch ökonomische Rahmenbedingungen, politische Rahmenbedingungen oder zeitliche Rahmenbedingungen. Die Beispiele für jede Art sind in der folgenden Tabelle 2.2 zu sehen.

Arten der Rahmenbedingungen	**Beispiele**
Technische Rahmenbedingungen	Materialeigenschaften, Klima
Organisatorische Rahmenbedingungen	Unternehmensorganisation, Geschäftsprozesse
Rechtliche Rahmenbedingungen	Normen, Vorschriften
Soziale Rahmenbedingungen	Interessensgruppen
Ökonomische Rahmenbedingungen	Budgetvorgaben, Amortisationszeiten
Politische Rahmenbedingungen	Wahl der Kooperationspartner
Zeitliche Rahmenbedingungen	frühester Beginn, Deadlines

Tabelle 2.2.: Gliederungen der Rahmenbedingungen [Ang14]

2.2. Model Driven Requirements Engineering

MDRE, welches für Model Driven Requirements Engineering steht, gehört mit Model Driven Architecture (MDA) zu der Methodologie Model Driven Engineering (MDE).
Unter MDE versteht man modellgetriebene Softwareentwicklung, die eine Vorgehensweise bei der Entwicklung eines Softwaresystems bezeichnet, bei der das System durchgängig mit Hilfe von Modellen beschrieben wird. Das Model Driven Engineering wird verwendet zu dem Ziel, die Entwicklung von Software zu beschleunigen und die Qualität der Software zu erhöhen.

In [Zwi13] wurde erläutert, dass MDA ein Ansatz zu der modellgetriebenen Softwareentwicklung ist und auf einer klaren Trennung von Geschäfts- und Anwendungslogik von der Plattformtechnologie beruht. MDA unterteilt ein Gesamtmodell in mehrere Schichten: Computation Independent

Model (CIM), Platform Independent Model (PIM) und Platform Specific Model (PSM). CIM ist unabhängig vom formalen Modell und beschreibt das System in umgangssprachliche Form. In dem Modell wird die Anforderungsanalyse durchgeführt. PIM bezieht sich auf plattformunabhängiges Modell für Geschäftsprozesse. Die Funktionalität des Systems wird im PIM beschrieben, die unabhängig von der Zielplattform sind. Mithilfe von Plattforminformationen werden die beschriebenen Funktionalitäten in das PSM transformiert.

Unter dem Begriff MDRE versteht man modellgetriebene Anforderungsanalyse, die auf MDA aufbauend ein spezieller Fall vom MDE ist. Modellgetriebene Anforderungen können auch als modellbasierte Anforderungen bezeichnet, weil dabei Anforderungen durch die Verwendung der Modellierungssprache dokumentiert werden. Somit können die Modelle generiert und weiter verwendet werden. In den folgenden Abschnitten werden zunächst die modellbasierten Anforderungen erläutert, danach werden die Vorteile der modellbasierten Anforderungen präsentiert.

2.2.1. Modellbasierten Anforderungen

Modellbasierte Anforderung sind anhand der Erläuterung in [Sch14] die Anforderungen, die in einem Modell implizit enthalten ist, so dass sich diese aus der graphischen Darstellung ableiten lassen können. Das Stichwort „Modell“ weist darauf hin, dass die Anforderungen durch die Verwendung der Modelle auf eine graphische Weise beschrieben oder dokumentiert werden können. Im Gegensatz zu der textuellen Anforderungen, die in natürlichen Sprachen dokumentiert werden, besitzen modellbasierte Anforderungen mehr Anschaulichkeit. Der Einsatz eines geeigneten Modells spielt dabei eine wichtige Rolle.

Ein Modell, das das wichtigste Mittel der modellbasierten Anforderungen ist, kann als ein abstrahierendes Abbild einer existierenden oder fiktiven Realität verstanden werden[Poh08]. In vielen Bereichen sind Modelle seit langer Zeit im Einsatz gelangen. Neben physikalischen Modellen werden Modelle auch graphisch präsentiert. Im Vergleich zu der textuellen Dokumentation sind graphische Modelle anschaulicher und leichter zu verstehen.
Bei Software handelt es sich um einen abstrakten Gegenstand. Deswegen kommen graphische Modelle in der Softwareentwicklung zum Einsatz. Die graphischen Modelle können spezifische Perspektiven des Systems anschaulich visualisieren und die Komplexität der Betrachtung reduzieren[Poh08]. Solche Modelle werden in der Softwareentwicklung als konzeptuelles Modell bezeichnet.

Anforderungsmodelle sind konzeptuelle Modellen, die Anforderungen dokumentieren. Bei der Anforderungsanalyse werden Modelle als hauptsächliches Ergebnis angesehen. Das Ergebnis, das durch ein Modell geliefert wird, besitzt in der Regel einen Formalismus. Der Formalismus besteht aus einer Mengen von Konzepten in einer speziellen Notation. Die folgende Abbildung 2.2 illustriert den Zusammenhang zwischen System, Modell und Formalismus.

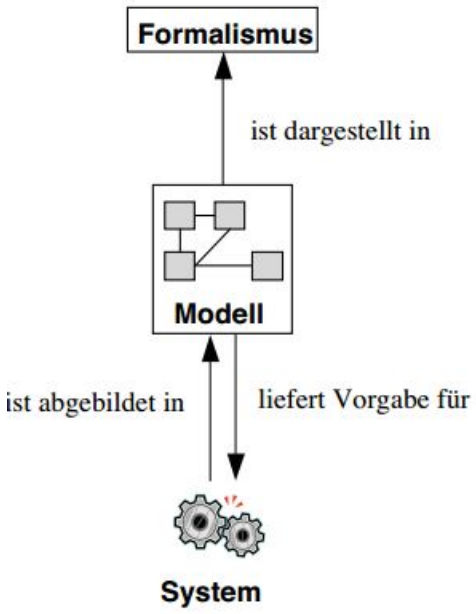

Abbildung 2.2.: Zusammenhang zwischen System, Modell und Formalismus [Pat10]

Ein konzeptuelles Modell wird vor allem eingesetzt, um ein Verständnis zu erhalten, wenn die Anforderungen eines Systems komplex, unübersichtlich und ungeeignet sind. Eine richtige Modellbildung kann deswegen zu dem angemessenen Entwurf und der einwandfreien Implementierung beitragen.

Die Voraussetzung für die Dokumentation und Interpretation von Anforderungen in Anforderungsmodellen ist die Kenntnis der dem Anforderungsmodell zugrunde liegenden Modellierungssprache. [Poh08] erläutert, dass eine konzeptuelle Modellierungssprache selbst durch ein konzeptuelles Modell definiert werden kann. Ein Modell, das eine Modellierungssprache definieren kann, wird als „Metamodell“ bezeichnet. Wiederum kann das Metamodell unter Verwendung einer Modellierungssprache definiert werden, die als „Meta-Sprache“bezeichnet wird. Analog können weitere Meta-Metamodelle durch Meta-Sprachen definiert werden, die theoretisch unendlich viele Metaebenen generieren können.

Die Modellierungssprache UML, die in Kapitel 2.3 vorgestellt wird, ist eine verbreitet verwendete Modellierungssprache. Zu dieser Modellierungssprache hat die Object Management Group (OMG) eine 4-Ebenen-Architektur (siehe Abbildung 2.3) für die Metamodelle eingeführt.

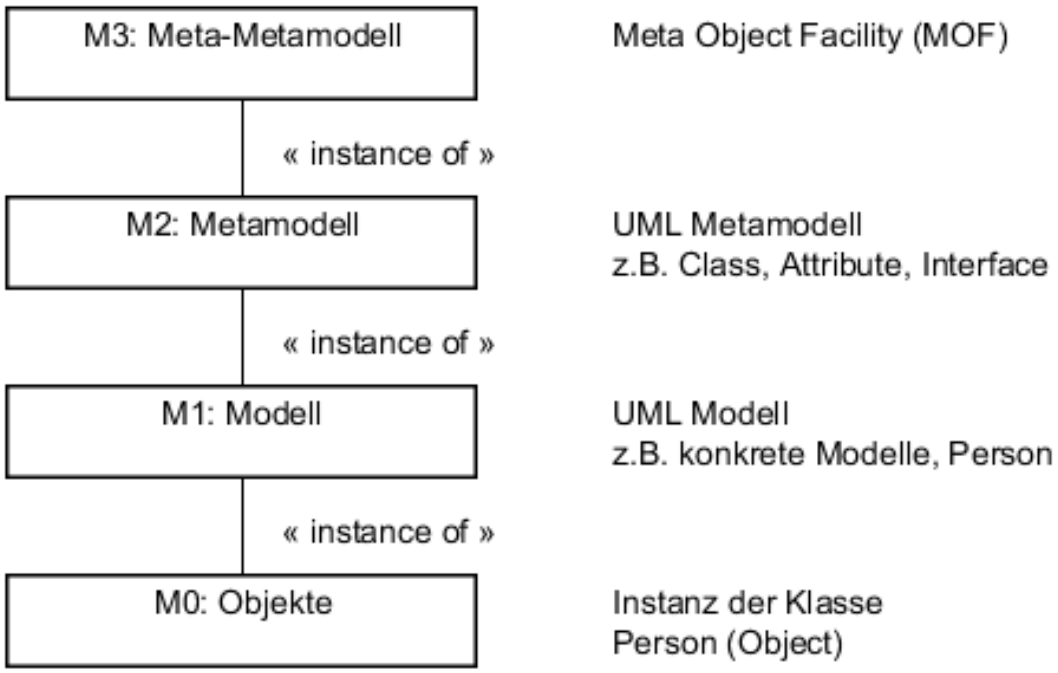

Abbildung 2.3.: Die 4-Ebenen-Architektur der OMG [IT14]

Ausgehend von der Architektur stellt die unterste Ebene M0 die realen Ausprägungen dar, die konkrete Objekte verkörpern. Die Ausprägungen der Ebene werden von der Modellkonstrukture auf der Ebene M1 definiert.
Die Ebene M1 interpretieren die Modelle. Die Anforderungsmodelle werden auf dieser Ebene verwaltet. Die Modellierungssprache dafür sind jedoch durch Metamodelle auf der Ebene M2 definiert.
Auf der Ebene M2 werden die UML-Modelle in Form des UML-Metamodells definiert. In der Regel stellt sich das Klassenmodell das UML-Metamodell dar, durch das die meisten UML-Modelle definiert werden.
Die oberste Ebene M3 definiert die Meta-Metamodelle mit der Meta Object Facility (MOF) und der Definition der Modell- und Sprachkonstrukture, die zur Konstruktion der Meta-Modelle auf der Ebene M2 verwendet werden.

2.2.2. Vorteile modellbasierter Anforderungen

Modelle spielen eine wichtige Rolle bei der Anforderungsanalyse. Entsprechend lassen sich einige Vorteile gegenüber den textuellen Anforderungen charakterisieren.
[Poh08] und [Pat10] haben die folgenden Vorteilen genannt:

1. Informationen in graphischer Darstellung lässt sich im Vergleich zum Text schneller erfassen und besser einprägen. Mit Hilfe von den geeigneten Notationen und Modellelementen können

die Zusammenhängen und Beziehungen zwischen den Anforderungen besser erkannt und erlernt werden. Bei textueller Darstellung eines komplexen Sachverhalts ist es schwer den Überblick zu behalten.

2. Anforderungsmodelle bieten die Möglichkeit zur Bildung diskreter Perspektiven bei der Modellbildung. Ein geeignetes Hilfsmittel ist die Modellierungssprache UML. Mit der standardisierten graphischen Notationen werden die Abläufe des Systems und die Zusammenhänge in einem Diagramm visualisieren. Verschiedene Modellierungssprachen können auch verschiedene Perspektiven ermöglichen.

3. Konzeptuelle Modelle bieten ferner die Möglichkeit zur Abstraktion durch Klassifizierung, Komposition und Generalisierung. Die Komplexität der resultierenden Modelle wird dadurch verringert und die Komplexität des zu entwickelnden Systems könnte optimiert werden.

Obwohl der Einsatz der Modelle offensichtlich die Anforderungsanalyse verbessern kann, dient die Modellbildung zu Präzisierung und Detaillierung der textuellen Anforderung. Eine Kombination der beiden Dokumentationsarten kann die Vorteile nutzen und gleichzeitig die Nachteile weitgehend eliminieren.

2.3. Modellierungssprache

Um eine modellbasierte Anforderung zu realisieren, kommen eine oder mehrere geeignete Modellierungssprachen zum Einsatz. Die Modellierungssprachen dienen dazu, die Modellbildung der Anforderung zu charakterisieren.
Im Laufe der Jahre sind zur Modellierung zahlreiche Modellierungssprachen aufgekommen. Die verbreitetste Sprache darunter ist UML.
UML entstand ursprünglich als eine Vereinigung der Konzepte aus den alten objektorientierten Ansätzen. Als die meist von dem IT-Bereich verwendete Modellierungssprache stellt UML eine einheitliche Notation der Modellkonzepte zur Verfügung. Die zentrale Darstellung von UML sind Diagramme, die als Projektionen auf die verschiedenen Aspekte einer Systemmodellierung aufgefasst werden können[Pat10]. Für die verschiedenen Aspekte eines Modells werden entsprechend verschiedene Diagramme in UML benutzt, die die Beschreibungsmöglichkeiten erfüllen. Die folgende Abbildung 2.4 interpretiert die Beziehung zwischen Diagrammen.

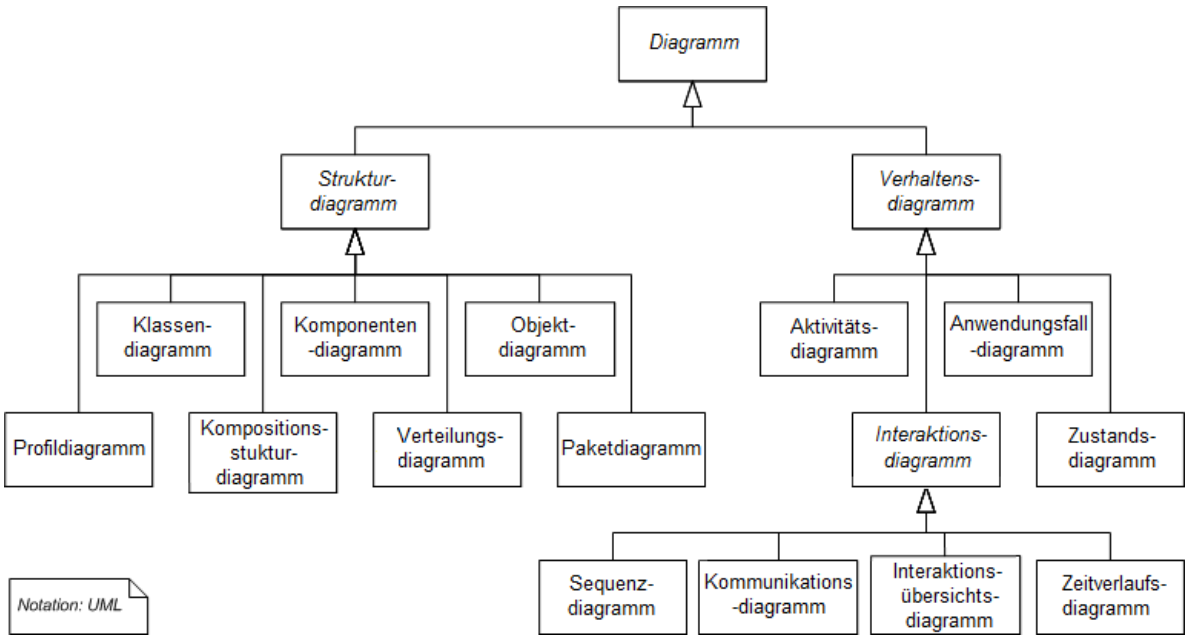

Abbildung 2.4.: Hierarchie zwischen den UML-Diagrammen[OMG]

Die Abbildung illustriert, dass zwei Kategorien von Diagramme in UML aufgeteilt wurden. Einige gehören zu dem Typ der Strukturdiagramme, während die restlichen zu dem Typ der Verhaltensdiagramme gehören.
Strukturdiagramme beschreiben statische Aspekte eines Objekts. Zur Beschreibung werden Klassen, Attribute, Operationen, Beziehungen, Schnittstellen eingesetzt. Unter den Strukturdiagrammen ist das Klassendiagram, welches verschiedene Klassen repräsentiert, das am häufigsten verwendete.

Verhaltensdiagramme beschreiben die Dynamik zwischen Objekten eines Systems mit Hilfe von Interaktionen, Objektkonfigurationen und Zustandshistorien. Zu Verhaltensdiagrammen gehören noch Interaktionsdiagramme, die spezielle Verhaltensdiagramme sind und Interaktionen zwischen verschiedenen Objekten beschreiben.

Ein Anwendungsfalldiagramm ist ein typisches Verhaltensdiagramm, welches das erwartete Verhalten eines Systems darstellt und die Anforderungen an ein System spezifiziert. Anwendungsfalldiagramme setzen sich aus mehreren Anwendungsfällen, mehreren Akteuren und einer Systemgrenze. Ein Anwendungsfall beschreibt alle möglichen Szenarien, die durch die Interaktion zwischen einem Akteur und einem System stattfinden werden. Zum Extrahieren der funktionalen Anforderungen werden Anwendungsfälle meistens verwendet. Funktionale Anforderungen werden durch Anwendungsdiagramme modelliert. Ein Anwendungsfalldiagramm stellt keine Ablaufbeschreibung dar, um einen Ablauf zu modellieren, stehen Aktivitätsdiagramme, Sequenzdiagramme oder Kollaborationsdiagramme zur Verfügung.

Bei Interaktionsdiagrammen handelt es sich um spezielle Verhaltensdiagramme, die für besondere Zwecke der Modellierung eingesetzt werden. Die Interaktion wird als Austausch von Nachrichten

zwischen Objekten beschrieben, die dabei beteiligte Objekte, Reihenfolge der ausgetauschten Nachrichten zwischen den Objekten und weitere Verhaltenselemente definieren kann.
Sequenzdiagramme sind ebenfalls Interaktionsdiagramme und beschreiben den Austausch von Nachrichten zwischen Objekten mittels Lebenslinien in bestimmten Szenarien. Ein Sequenzdiagramm enthält Klassen einschließlich deren Objekten, Nachrichten, die synchronisch oder asynchronisch ausgetauscht werden. Den Eigenschaften der Sequenzdiagramme gemäß können daher Sequenzdiagramme die Standardabläufe der Anwendungsfälle darstellen.

2.4. Maschinelle Sprachverarbeitung

Maschinelle Sprachverarbeitung, bekannt als Natural Language Processing (NLP), ist ein Teilbereich der Computerlinguistik. In diesem Bereich können einerseits natürliche Sprache als Eingabe durch die Verarbeitung eines Computers analysiert werden oder andererseits Texte aus formalen, computerlinguistisch verarbeitbaren Information automatisch generiert werden. Da in dieser Arbeit die Anforderungen in Textform analysiert und modellbasiert verarbeitet werden, steht die Extraktion von Informationen im Mittelpunkt.
[Zwi13] hat zur Analyse der Texte das Saarbrücker Pipelinemodell (siehe Abbildung 2.5) betrachtet und verwendet. Auf Basis der Arbeit von Zwirn werden zwei dieser Schritte des Pipelinemodells eingesetzt: Tokenisierung und Syntaktische Analyse. Tokenisierung dient zu der Segmentierung eines Textes und syntaktische Analyse dient zu der Generierung der Abhängigkeiten zwischen Wörtern eines Textes. Die restlichen Schritten sind in dieser Arbeit nicht relevant, daher werden sie nicht betrachtet.

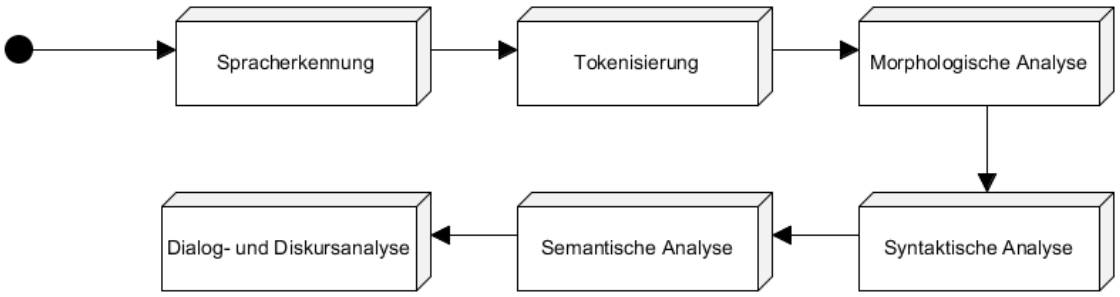

Abbildung 2.5.: Saarbrücker-Pipelinemodell [Zwi13]

2.4.1. Tokenisierung

Tokenisierung bezeichnet die Segmentierung eines Textes in Einheiten der Wortebene. Jedes Wort ist eine Zeichenkette, die auf die Buchstabenketten abgebildet werden. Die Tokenisierung bildet Voraussetzung für die Weiterverarbeitung beispielsweise zur syntaktischen Analyse durch Parser oder zur Extraktion der Informationen.
Token sind als Bausteine eines Textes bezeichnet, die Wörter im klassischen Sinne, Abkürzungen,

Zahlzeichen, Satzzeichen sowie Sonderzeichen repräsentieren können. Die Zerlegung eines Textes geschieht in Token erfolgt, indem sich Leerstellen in einem Text als Übergang zwischen Token interpretieren lassen und die Satzzeichen von Token abtrennt werden.
Jedem Token werden bei der Tokenisierung eine Wortart zugeordnet, um zu zeigen, zu welcher Kategorie die Wörter oder Zeichen gehören. Unter Wortart versteht man „Part-of-Speech (POS)-Tagging“ , die die Klasse von Wörtern einer Sprache aufgrund der Zuordnung nach gemeinsamen grammatischen Merkmalen präsentiert. Die Wortart ist zu unterscheiden von der syntaktischen Funktion eines Wortes wie Normen, Verb, Adverbial, Attribut usw.

Ein Beispiel zur Erläuterung ist wie folgt zu sehen:

The/DT customer/NN can/MD withdraw/VB cash/NN ./.

Der Beispielsatz ist nach der Leerstellung und dem Satzzeichen tokenisiert. Jeder Token sind durch die entsprechende POS kennzeichnet. Dadurch werden „The/DT customer/NN“ und „cash/NN“ als Normen und „withdraw/VB“ als Verb erkannt.
Die Tokenisierung hat jedoch Probleme mit Begriffsklärung von Satzgrenzen, Normalisierung von großgeschriebenen Wörtern und Erkennung von Abkürzungen.

2.4.2. Syntaktische Analyse

Die Syntax ist das Teilgebiet der Computerlinguistik, das sich mit dem Satzbau beschäftigt. Eine syntaktische Komponente gliedert den Satz in seine Bestandteile und identifiziert deren Funktionen. Um die Bedeutung der Bestandteilen eines Eingabesatzes begreifen zu können, müssen die semantische Zusammengehörigkeit der Wörter und Satzteile sowie deren Funktionen in dem gesamten Satz festgelegt werden. Die syntaktische Analyse des Satzes in seine Bestandteile wird als Parsing bezeichnet. Die Komponente, die Parsing durchführt, wird als Parser benannt. Die Ausgabe des Parsers ist die syntaktische Repräsentation des Satzes.

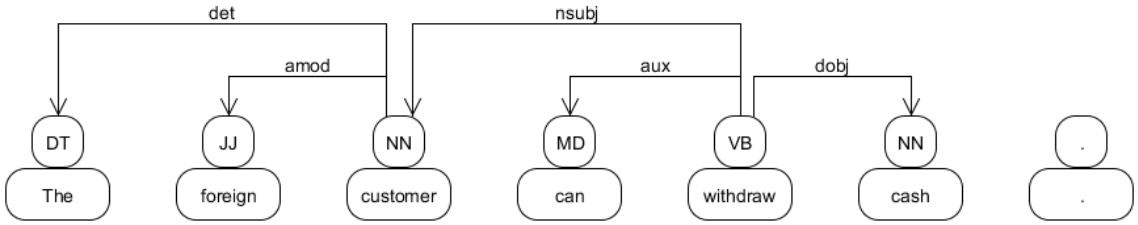

Abbildung 2.6.: Syntaktische Analyse zu dem Satz: The foreign customer can withdraw cash.

Ein Parser ermittelt nicht nur die Informationen der einzelnen Bestandteilen eines Eingabesatzes, sondern auch Informationen einer Wortgruppe, in der die Wörter semantisch zusammengehören.
Der POS-Tag „NP“ , das für „Noun Phrase“ steht, stellt beispielsweise eine zusammengesetzte Wortgruppe dar, die ein Normen bezeichnet. Ein Beispiel für das POS-Tag ist folgend dargestellt:

- The foreign customer:
  NP (DT The) (JJ foreign) (NN customer)

Unter Verwendung des Stanford Parsers ist ein zusammengesetzten Normengruppe zu sehen. Die drei Wörter in dem Beispiel haben drei Wortarten, die jeweils ein Determinator, ein Adjektive, ein Normen bezeichnen. Durch die syntaktische Analyse ist die semantische Beziehungen zwischen den drei Wörtern charakterisiert, deswegen werden sie als ein „NP“ zusammengesetzt. Anhand dieser Eigenschaft kann ein Parser die semantischen Beziehungen und Abhängigkeiten zwischen allen Bestandteilen eines Satzes ermitteln. Die Abbildung 2.6 illustriert die syntaktische Analyse zu dem Satz: The foreign customer can withdraw cash.

3. Vewandete Arbeiten und genutzte Technologien

In diesem Kapitel werden die verwandten Arbeiten sowie die genutzten Technologien zur Entwicklung des Zielsystems vorgestellt. Zunächst werden die verwandtesten Arbeiten zur Erläuterung der Hinweise auf die grundsätzlichen Informationen vorgestellt, wie man die Grundlage verwirklicht. Gefolgt sind die relevanten Technologien, die für die Realisierung des Systems verwendet werden.

3.1. Zielverwandte Arbeiten

Für die Anforderungsklassifikation und Anforderungsumwandlung wurden bereits verschiedene Herangehensweise entwickelt. Die als Referenz verwendete Diplomarbeit [Zwi13] stellt eine Methode für Klassifizierung mithilfe der Ontologie vor, die die Anforderungen anhand bestimmter Schlüsselwörter in Klassen unterscheidet.
In [SLF04] werden Methoden von Rational Unified Process (RUP) genutzt, um die textuellen Anforderungen automatisiert von Anwendungsfällen (Use Case) in Klassenmodelle umzuwandeln. Dabei werden die Spezifikationsdokumente der Anwendungsfälle zur Analyse für die Objekte der Klassenmodelle eingeführt. Für die Modellierung wird *Rational Rose Modeler* von IBM als Werkzeug genutzt.
Auch in [KS08] werden Anwendungsfalldiagramm und Klassenmodell aus natürlichsprachigen Anforderungen generiert, im Vergleich zu anderen Systemen, die das gleiche Ziel haben, ist das System in [KS08] überlegen mit höherer Genauigkeit.

3.1.1. Analyse und Auswertung von gewichteten Anforderungen in technischen Spezifikationen

In der Diplomarbeit [Zwi13] handelt es darum, die textuellen Anforderungen aufgrund einer Ontologie automatisiert in drei Klassen zu unterscheiden und zu klassifizieren.
Die drei Klassen, die in [Zwi13] definiert wurden, lassen sich wie folgend aufzählen:

1. Die Klasse der echten Anforderungen betrifft alle Sätze, bei denen es sich definitiv um Anforderungen handelt.
2. Um eine optionale Anforderung handelt es sich, wenn die Erfüllung der Anforderung eine Wahlmöglichkeit des Entwicklers ist.

3. Als letzte Klassen liegen die schwachen Anforderungen vor. Es sind Anforderungen bei denen nicht klar ist, ob sie optional sind oder nicht.

Alle Dokumente, die sich mit dem System überprüfen lassen, können als „Anforderung“, „optionale Anforderung“ und „schwache Anforderung“ klassifiziert werden.
Laut der Erläuterung untersucht man die Besonderheiten, mit denen ein Satz als Anforderung betrachtet werden kann. Die Klasse der Sätze, die alle Merkmale einer Anforderung besitzen, ist die gesuchte Klasse. Die Klassifikation wurde von Ontologien (W3C OWL) übernommen.
Daraus hat sich folgender Ablauf (Abbildung 3.1) für die Extraktion von Anforderungen ergeben.

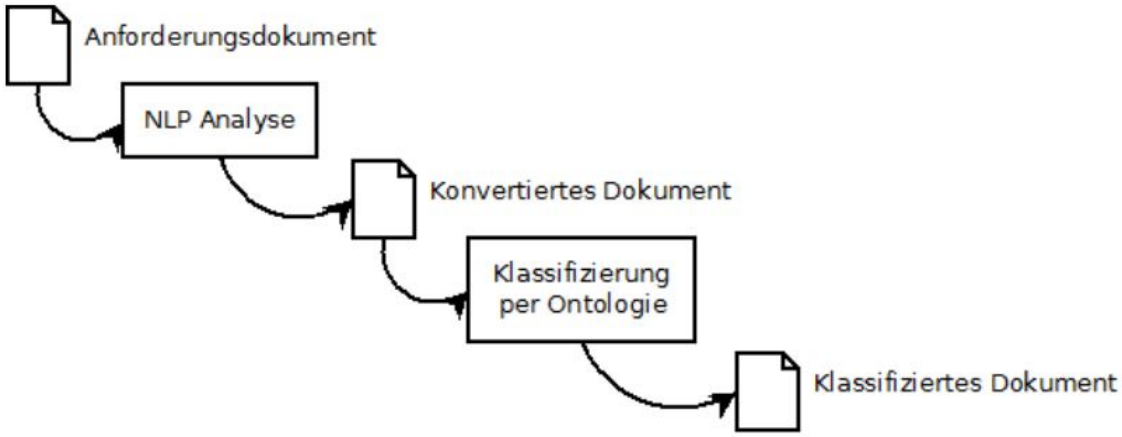

Abbildung 3.1.: Vorgehen bei der Klassifizierung der Anforderungenspezifikation [Zwi13]

Zuerst liefert ein Programm der NLP, in [Zwi13] wurde dazu Stanford CoreNLP verwendet, die Struktur der Sätze und Wörter. Die POS Analyse und die Lemmatisierung der Wörter werden von der Extraktionsontologie verwendet. Schließend können die Anforderungen identifiziert werden. Um dies zu bewerkstelligen, muss identifiziert werden, was eine Anforderung für Eigenschaften hat. Die Anforderungen, die Subjet, Objekt und Verb haben, sind Anforderungskandidaten. Eine Suche nach bestimmten Schlüsselwörtern liefert nun aus den Anforderungskandidaten solche, bei denen es sich um Anforderungen handelt.

3.1.2. UCDA: Use Case Driven Development Assistant Tool for Class Model Generation

In einer Arbeit entwickeln Subramaniam et. al. [SLF04] ein System für die Erzeugung des Klassenmodells mithilfe des *Rational Unified Processes (RUP)*, indem sie eine vollständige, korrekte und eindeutige Spezifikationsdokument von Anwendungsfällen (Use Cases) verwenden. Object Oriented Analysis and Design (OOAD) ist ein verbreitetes Verfahren zur Softwareentwicklung in dem Bereich der Software. Die Objekte und Klassen aus textuellen Anforderungen zu identifizieren zählt zu den wichtigsten Anwendungen von dem OOAD-Verfahren.

In dem folgenden Aktivitätsdiagramm 3.2 erfährt man den Ablauf des Use Case Driven Development Assistant Tool (UCDA)-Systems. Die erste Komponente von dem System UCDA ist ein „Use Case Model Developer “. Die textuellen Anforderungen werden zunächst durch einem Natural Language Parser analysiert und anhand einiger vordefinierten Satzregeln in einfachen Anwendungsfälle zerlegt. Nach einem erfolgreichen Extrahieren wird aus den Anwendungsfällen ein Spezifikationsdokument generiert und nach einer Reihe Bearbeitungen in XML-Format gespeichert. Die Spezifikation dient zu der grundsätzlichen Generierung eines Klassenmodells. Die Besonderheit von dem System liegt daran, dass in den Zwischenphasen neben dem Use-Case-Diagramm noch ein Robustheitdiagramm und eine Kommunikationsdiagramm durch der Anwendung der zweiten Komponente „Class Model Developer “ entstehen. Solche Diagramme validieren, ob ein Randobjekt in dem Klassenmodell notwendig ist.

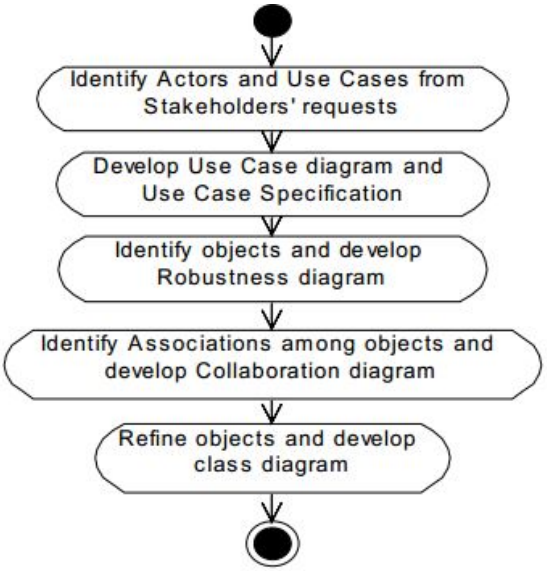

Abbildung 3.2.: Ablauf des UCDA [SLF04]

Zur Modellierung wird der *Rational Rose Modeler* von IBM verwendet. UCDA liest die gespeicherte Spezifikation der Anwendungsfälle in XML-Format ein und wandelt sie in dem *Rational Rose Modeler* mit einer selbstentwickelten Plugin. Dies System verringert den Zeitaufwand bei der OOAD-Aufgabe, außerdem bietet das System den wertvolle Hinweise auf das Gebiet.

3.1.3. Static UML Model Generator from Analysis of Requirements (SUGAR)

Die Arbeit [KS08] entwickelt ein Werkzeug namens *Static UML Model Generator from Analysis of Requirements (SUGAR)* für die Erzeugung der UML-Modelle aufgrund der Analyse von natürlich-sprachigen Anforderungen. Dieses Werkzeug expandiert die Ideen von früheren Werkzeugen wie UCDA [SLF04] und Linguistic assistant for Domain Analysis (LIDA) [OLR01], die hauptsächlich die Lücke zwischen den Phasen von Anforderungsanalyse und Entwurf mithilfe der NLP-Technologien

schließen. Die beiden Werkzeuge können zwar die UML-Diagramme generieren, sie sind jedoch nur semiautomatisiert. Die Schwächen der erwähnten Werkzeuge wurden in [KS08] aufgezählt. UCDA identifiziert nur die Analysemodelle von Klassen und kann lediglich die Modelle mit *Rational Rose Modeler* realisieren, während LIDA nur Subjekte, Objekte sowie Verben identifiziert und somit mehr Nutzerinteraktionen für die Diagrammerzeugung verlangt.
SUGAR folgt dem RUP-Verfahren und verwendet effiziente NLP-Werkzeuge, um die UML-Modelle aus Anforderungen zu generieren. Neben den üblichen Technologien wird bei dem System ein Glossar eingeführt, welches zu den essentiellen Artefakten von RUP zählt. Ein Glossar dient bei dem Softwareentwicklungsprozess dazu, die Begriffe in Einheit zu bringen und die Kommunikationslücke zwischen Beteiligten zu schließen, damit die Anforderungen nicht uneindeutig erfasst werden können.
Die Stärke von SUGAR lässt sich in dem Grad der Automatisierung und der Rückverfolgbarkeit einsehen. Dank den geeigneten NLP-Werkzeugen löst SUGAR das domänenspezifische Problem in dem System, es existieren auch die Möglichkeit, dass bei der Modellerzeugung keine zusätzliche fertiggestellte Werkzeuge zu brauchen.

3.2. Genutzte Technologien

In dieser Arbeit werden verschiedene Technologien verwendet, die jedem als Open-Source online zur Verfügung stehen. Zuerst werden die NLP-Werkzeuge für Verarbeitung der natürlichen Sprache vorgestellt.

3.2.1. Stanford CoreNLP

Auf der Diplomarbeit [Zwi13] aufbauend verwendet das System in dieser Arbeit die Technologien in [Zwi13] weiter, die Datenstruktur für das entsprechende Teilsystem soll auch nicht viel abweichend sein. Zur Analyse und Verarbeitung der natürlichen Sprache bietet „Stanford Natural Language Processing Group “ diverse Werkzeuge und Bibliotheken, die gut zu bedienen sind.

Die Arbeit nutzt deswegen weiterhin Stanford CoreNLP Werkzeugsatz, der eine erweiterbare Pipeline als Kernfunktionalität besitzt. Stanford CoreNLP ist bekannt als eine auf Java basierte Pipeline-Framework für Annotationen, der eine Reihe von üblichen NLP Schritten von Tokenisierung über Lemmatisierung bis Koreferenz anbietet. Die grundsätzliche Architektur (siehe Abbildung 3.3) ist ein Durchführungsablauf und besteht aus zwei Komponenten: Annotator und Annotation. Der Stanford CoreNLP nimmt die Texteingabe in Englischer Sprache und auch Textdateien in chinesischer, spanischer, arabischer oder deutscher Sprache. Mit einer Texteingabe in die „Annotation Object “ fügen die Reihe der Annotators Informationen des Texts in eine Analyse-Pipeline hinzu, die annotierten Sätze werden als Ausgabe geliefert und entweder in XML-Form oder in Textform gespeichert. Dabei werden Wörter in den zu analysierenden Sätzen in ihre Basisform gebracht, ihre POS bestimmt, Namen entdeckt, Daten, Zeiten und numerische Mengen normalisiert und die Struktur des Satzes analysiert.

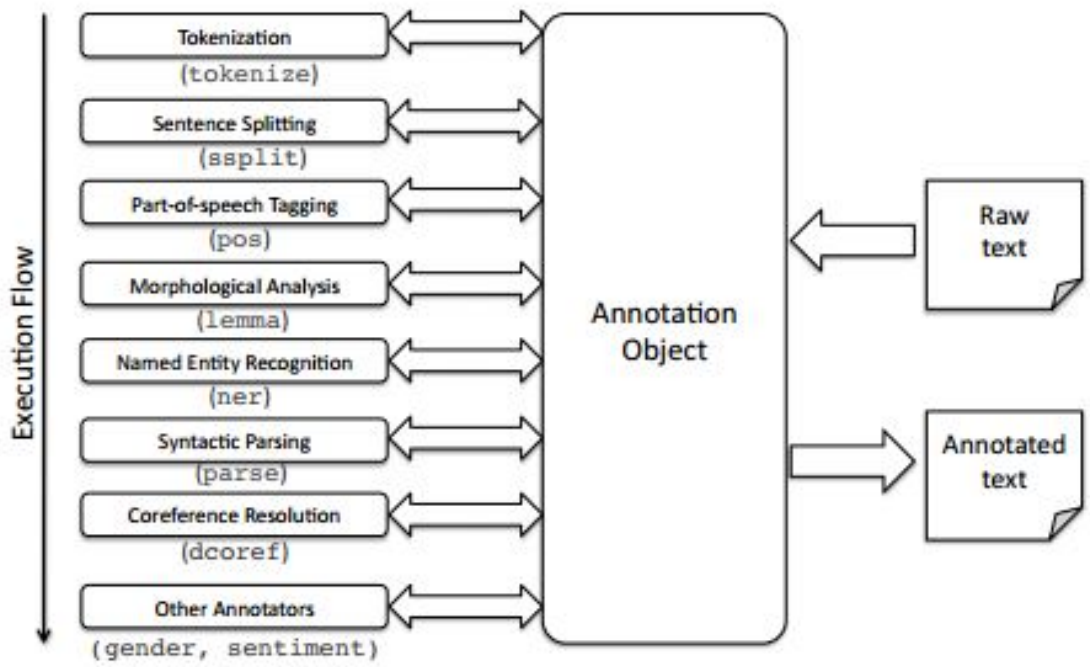

Abbildung 3.3.: Architektur des gesamten Systems von Stanford CoreNLP [MSB14]

Neben den Standardannotators bietet Stanford CoreNLP noch zusätzliche Annotators an, indem Nutzer die Java-Klasse `edu.stanford.nlp.pipeline.Annotator` erweitern und einen Konstruktor mit Signatur `(String, Properties)` definieren.
Wegen der langen Pipeline ist die Komplexität des Stanford CoreNLPs ziemlich hoch, um die Standardannotators komplett zu verwenden wird dementsprechend ein großer Speicherplatz angefordert. Obwohl der Stanford CoreNLP eine gute Leistung liefert, hängt die Laufzeit dafür von dem genutzten Rechner ab. Eine lange Laufzeit kann auch nicht vermieden werden.

3.2.2. Stanford Parser

Der statistische Stanford Parser gehört zu der Stanford Natural Language Processing Group, er ist ein Programm, das die grammatische Struktur von Sätzen analysiert. Dabei werden beispielsweise Wörter analysiert, welcher von ihnen zu einer Gruppe als Redewendungen gehören oder welche Teil der Subjekt-Objekt-Verb (SVO)-Tripel sind.
Das Parser-Paket ist auch in Java implementiert und enthält hoch optimierte Probabilistic Context Free Grammars (PCFG) und „Lexicalized Dependency Parsers “ sowie ein „Lexicalized PCFG Parser“. Die Parsers konzentrieren sich jeweils auf die Beziehungen und Abhängigkeiten zwischen den Wörtern eines Satzes und auf die syntaktische Struktur eines Satzes.

Die Eingabe ist wie bei dem Stanford CoreNLP eine Textdatei, während sich die Ausgabe von dem Parser Abhängigkeiten (Stanford Dependencies) und Strukturbäume (phrase structure trees) ergibt. Die Abhängigkeiten beziehen sich auf „Stanford Typed Dependencies“ , die binäre Abhängigkeiten zwischen Gouverneur und Angehörigen beschreiben und aus denen fünf Abhängigkeitsrepräsentationen generiert werden. In [MM08] sind die Definitionen für etwa 50 „Stanford Typed Dependencies“ (siehe Tabelle im A.1) zu finden, darauf aufbauend werden die Abhängigkeitsrepräsentationen in fünf Klassen unterschieden:

Basic Dieser Typ nutzt die binären Abhängigkeiten um einen Strukturbaum zu bekommen, in dem Typ gibt es keine gekreuzten Abhängigkeiten, welches implizit, dass jedes Wort eines Satzes das Angehörige von einem der anderen Wörtern ist.

Collapsed dependencies Abhängigkeiten, die Präpositionen, Verbindungselemente sowie Informationen zu den Referenten der Relativsätze enthalten, werden zusammengelegt um eine direkte Abhängigkeit zwischen den Wörten zu bekommen.

Collapsed dependencies with propagation of conjunct dependencies Wenn sich eine Konjunktion (and / or) in dem Satz befindet, orientieren sich die Abhängigkeiten bezüglich der Konjunktion an ihre richtige Angehörigen. Da diese Repräsentation lediglich eine Erweiterung für **Collapsed dependencies** ist, wird keine Baumstruktur von der Repräsentation gewährleistet.

Collapsed dependencies preserving a tree structure In diesem Typ werden die Abhängigkeiten entfernt, die keine Baumstruktur behalten können.

Non-collapsed dependencies Die Repräsentation liefert die Basisabhängigkeiten ohne Zusammenlegung oder Übertragung der Konjunktionen und die zusätzlichen Abhängigkeiten, die Baumstruktur brechen.

Der Stanford Parser bietet zwei Klassen für die Erzeugung der Abhängigkeiten. Mit der Klasse `edu.stanford.nlp.parser.lexparser.LexicalizedParser` lässt die Texteingabe sich grammatisch analysieren, die Ergebnisse dafür sind ein Analysebaum in verschiedenen Formen. Die andere Klasse `edu.stanford.nlp.trees.EnglishGrammaticalStructure` kann dazu beitragen, „Stanford Dependencies“ durch einen Analysebaum in „Penn treebank“-Stil [San91] zu realisieren.

4. Lösungsansatz

In Kapitel 2 wurden die Grundlagen dieser Arbeit erläutert, in diesem Kapitel wird ein Konzept eingeführt, das die textuellen Anforderungen automatisiert klassifiziert und in entsprechende modellbasierte Anforderungen transformiert. Zunächst wird die Ausgangslage vorgestellt, warum die Klassifikation und Transformation für die Anforderungen notwendig sind. Danach werden Methoden zur Verarbeitung und Interpretation von Anforderungen interpretiert.

4.1. Die Ausgangslage

In den letzten Kapiteln wurden bereits erwähnt, dass die Anforderungsanalyse sich in einem grundsätzlichen Schritt bei Entwicklung eines hochwertigen Softwaresystems befindet. Lassen die Anforderungen sich richtig analysieren und spezifizieren, kann die Weiterverarbeitung eines Systems effizient fortsetzen.
Eine treffende Dokumentation der Anforderungen dient in den früheren Entwicklungsphasen zu der Gewährleistung einer präzisen Analyse für die Weiterverarbeitung, wobei die Anforderungen oft textuell formuliert werden. Ein Text lässt sich allerdings wegen der Fülle von Formulierungsmöglichkeiten nicht automatisiert verarbeiten. Dies führt dazu, dass die Qualitätsanalyse der Anforderungen und der Übergang von textuellen Anforderungen zur Implementierung auf manuelle Weise erfolgen müssen. Dabei ist es zeitaufwändig und fehleranfällig.
Weiterhin wird heutzutage im Bereich der eingebetteten Systeme immer mehr Wert auf die modellbasierte Entwicklung gelegt, mit einer textuellen Anforderung wird die Modelle des Systems nicht optimal verwirklicht.
Ein System, das die textuelle Anforderung automatisch analysiert sowie die als modellbasierte Anforderung interpretiert, ist wünschenswert. Für das Ziel sind die textuelle Dokumentationen zu überprüfen, ob sie echte Anforderungen sind und zu welchen Kategorien der Anforderungen sie gehören. Nach der Klassifizierung werden aus den analysierten Texten, die mithilfe von angemessenen Kriterien als Anforderungen bestimmt werden, automatisiert ihre entsprechend modellbasierten Anforderungen erzeugt.

4.1.1. Klassifizierung

In dem Kapitel 3.1.1 wurde die Diplomarbeit [Zwi13] als eine Referenzliteratur vorgestellt. Ein REC System wurde von der Diplomarbeit entwickelt, das textuelle Anforderungen mithilfe von Ontologien klassifiziert.

Eine Einschränkung des REC-Systems liegt allerdings an der Ontologie zur Anforderungsunterscheidung, die eine Erweiterung der Basisontologie ist. Der Ontologieausschnitt (Abbildung 4.1) repräsentiert das Modell zur Anforderungsklassifizierung, in dem die bestimmten Verben als Schlüsselwörter zur Unterscheidung betrachtet werden.

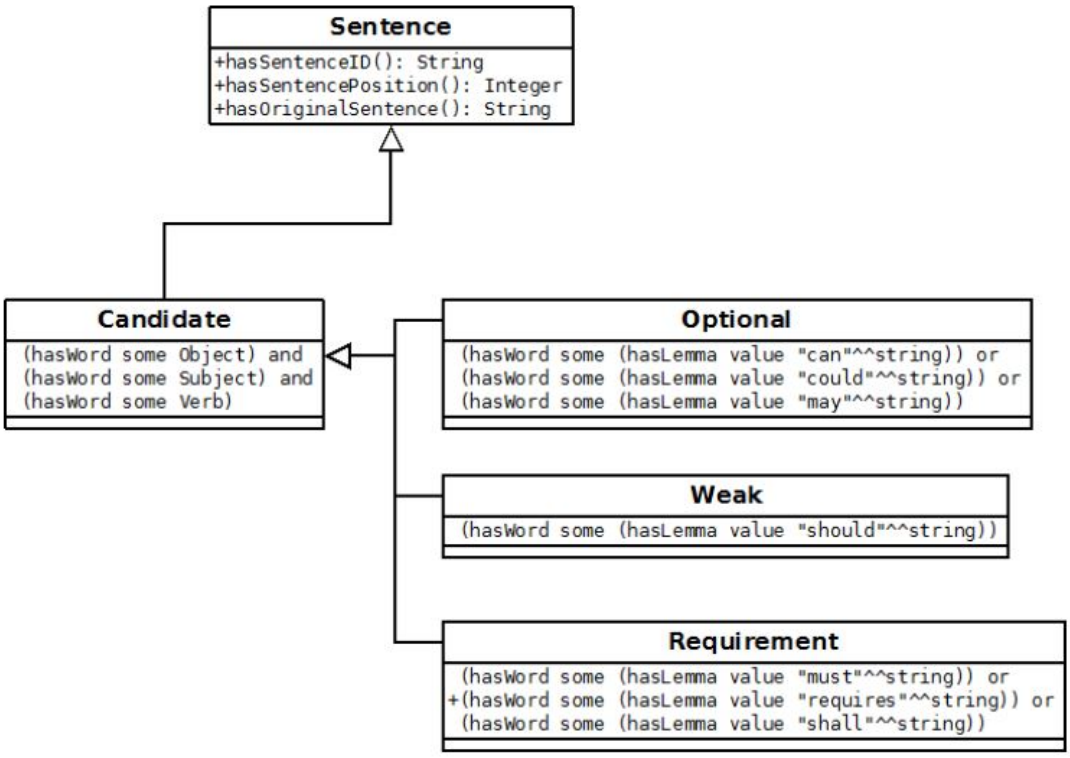

Abbildung 4.1.: Ontologieausschnitt zur Anforderungsunterscheidung [Zwi13]

Das Kriterium über die Anforderungsunterscheidung besteht darin, welche Schlüsselwörter, in dem Fall bestimmten Verben in den zu überprüfenden Sätzen enthalten werden. Wenn die Sätze Verben „can“, „could“ oder „may“ haben, werden die Sätzen als optionale Anforderungen bezeichnet. Die Sätze werden als schwache Anforderungen angesehen, falls sie den Verb „should“ enthält. Die Klasse der richtigen Anforderungen sind die Sätze, die Verben „must“, „requires“ oder „shall“ verwenden.

Vorteilhaft ist das Kriterium einerseits wegen seiner leichten Anwendbarkeit gut zu verstehen, für das keine fachlichen Kenntnisse über Anforderungsanalyse verlangt werden. Andererseits hat das Kriterium auch Einschränkungen, dass Sätze auf diese Weise zu klassifizieren und den drei Gruppen zuordnen.
Beispielsweise wird ein Satz „*Students must submit their assignments individually.*“ aufgrund des Kriteriums als eine richtige Anforderung klassifiziert, weil in dem Satz das SVO-Tripel (S: Students, V: submit, O: assignments) und der entscheidende Verb „must“ erfolgreich erkannt werden können. Laut der Definitionen über Anforderungen im technischen Sinne gehört dieser Beispielsatz jedoch

zu keiner Anforderungsgruppe, das bedeutet, dass der Satz zwar das ganze Kriterium erfüllt, ist trotzdem keine Anforderung. Dies führt offensichtlich zur falschen Klassifizierung einer Anforderung. Umgekehrt werden Sätze, die echte Anforderungen sind und keine Schlüsselwörter enthalten, wegen des Kriteriums falsch behandelt.

Wie in dem Kapitel 2 bereits erwähnt wurde, dass es verschiedene Klassifizierungsarten für die Anforderungen gibt. Die bekannteste und verbreitest verwendete Klassifikation sind „funktionale“ und „nicht-funktionale“ Anforderungen. Unter „nicht-funktionaler“ Anforderung kommen noch weitere Unterteile. Die Ergebnisse, die durch das oben genannte Kriterium generiert werden, können im Normalfall entweder funktionale oder nicht-funktionale Anforderungen sein. Da in dieser Arbeit es um die Erzeugung der Modelle aus einer textuellen Anforderung handelt, stehen funktionale Anforderungen im Mittelpunkt der Betrachtung.

Erweiterung gegenüber dem Kriterium in [Zwi13]

Modelle einer Anforderung können üblicherweise graphisch dargestellt werden. Für objektorientierte Methoden stehen die graphischen Notationen zur Verfügung.

Zu den objektorientierten Methoden wurde es in [Pat10] erklärt, dass die Ideen aus der objektorientierten Programmierung, den abstrakten Datentypen sowie den Zustandsübergangsdiagrammen zusammengesetzt wurden. Dabei führen sie zu einem integrierten Modell der funktionalen Anforderungen, in dem alle relevanten Systemaspekte berücksichtigt sind. Der Definition der funktionalen Anforderungen folge ist es auch davon ausgegangen, dass sich die funktionalen Anforderungen die erforderlichen Funktionalitäten des zu entwickelnden Systems ergeben, dafür kann ein objektorientiertes Modell vertreten.
Im Gegensatz dazu können nicht-funktionale Anforderungen der Meinung von [Pat10] nach allerdings nur textuell oder durch individuelle Erweiterung erfasst werden. Das weist jedoch darauf hin, dass nicht-funktionale Anforderungen ihrer Definition gemäß nicht in modellbasierte Form interpretiert werden können.

Unter Berücksichtigung des Ziels der Arbeit, automatisierte Erzeugung der Modelle zu erfolgen, wird die Methode zur Klassifikation in dieser Arbeit deswegen basierend auf dem Kriterium in [Zwi13] erweitert. Anforderungen werden nicht nur überprüft, ob sie die bestimmten Verben enthalten, sondern auch funktional und nicht-funktional unterschieden. Dabei werden die funktionalen Anforderungen weiter in Anwendungsfälle unterteilt, ihren Anforderungstypen entsprechend werden die verfeinerten funktionalen Anforderungen in Modelle transformiert. Da es schwer ist, Modelle aus nicht-funktionalen Anforderungen zu erzeugen, werden nicht-funktionale Anforderungen nur extrahiert jedoch nicht verwendet.

4.1.2. Transformation und Modellerzeugung

Modelle einer textuell klassifizierten Anforderung werden dem Ziel dieser Arbeit gemäß automatisch durch Transformation erzeugt. Zur Klarstellung der Methodik über Modellierung hat [Pat10] struktu-

rierte und objektorientierte Methoden vorgestellt, die zwei gegenseitige Aspekte bei der Modellierung sind. Die auf der objektorientierten Methode basierte Transformation, die neben der Klassifizierung das zweite Teilsystem in dieser Arbeit ist, wird in folgenden Abschnitten interpretiert.
Die Vorteile der modellbasierten Anforderungen sind bereits in dem Kapitel 2.2.2 vorgestellt. Gemäß den Vorteilen werden die textuelle Anforderungen nach der Klassifikation in geeignete Modellen transformiert.
In [Pat10] wurde erläutert, dass UML den derzeitigen Entwicklungsstand im Bereich objektorientierter Ansätze repräsentiert, das die meisten der früheren Ansätze konzeptionell und in einer einheitlichen Notation vereinigt. UML bietet eine Fülle von Beschreibungsmöglichkeiten, mit denen jeweils verschiedene Aspekte eines Modells dargestellt werden können.

Die Zusammenhänge der verschiedenen UML-Formalismen sind in der Abbildung 4.2 zu verstehen, die hinsichtlich Konsistenz berücksichtigt werden müssen. Von den Zusammenhängen ist es ausgegangen, dass Sequenzdiagramme, die Szenarien erläutern oder Anwendungsfällen spezifizieren, Anwendungsfalldiagramme (SD -(1)- AFD+AFB) und Klassendiagramme (SD -(2)- KD+KB) beeinflussen können. [Pat10] zufolge können Anwendungsfalldiagramme die Grundlagen für Sequenzdiagramme liefern, während Botschaften im Sequenzdiagramm zu Operationen im Klassendiagramm werden. Darüber hinaus liefern Anwendungsfalldiagramme Klassen, Operationen und Beziehungen in Klassendiagrammen. Neben den drei typischen UML-Diagrammen werden die Charakterisierungen anderer Diagramme ausführlich in [Pat10] dargestellt.

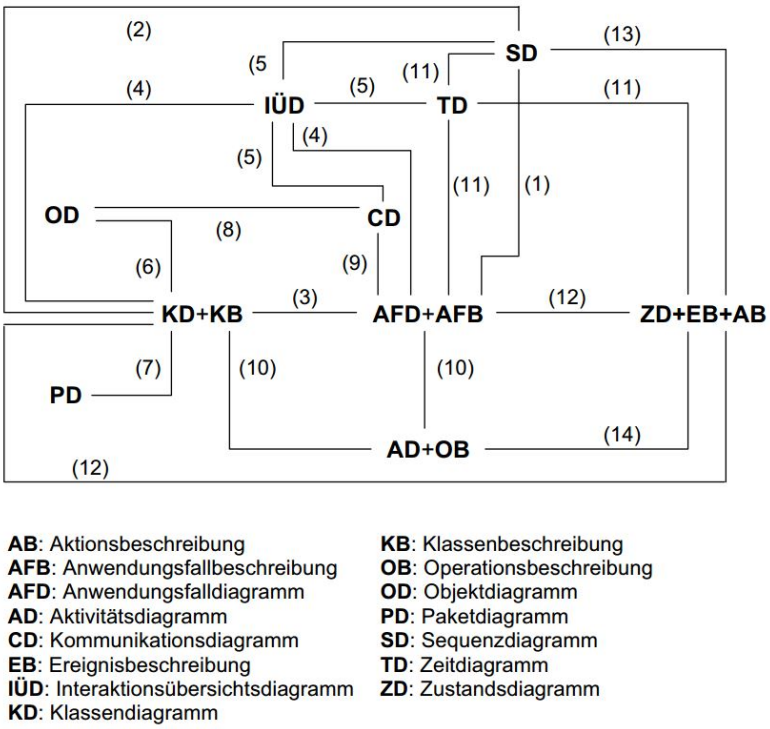

Abbildung 4.2.: Zusammenhänge zwischen den UML-Modellen [Pat10]

Unter Berücksichtigung der Eigenschaften von funktionalen Anforderungen stehen Anwendungsfalldiagramme, Sequenzdiagramme für das anzustrebende Teilsystem der Transformation zur Verfügung. Um das Ziel der Arbeit zu behalten, werden die als Ergebnisse zu liefernden Modelle aus einer modellbasierten Anforderung realisiert. Die Modelle werden so dargestellt, dass Sequenzdiagramme von Anwendungsfalldiagrammen abgeleitet werden, indem ein Dokument über Anwendungsfälle in XML Form die modellbasierte Anforderung vertritt, das gleichzeitig ein Anwendungsfalldiagramm liefern kann und aus dem sich die Modelle der Sequenzdiagrammen ergeben können.
Die Abbildung 4.3 illustriert die unmittelbare Beziehungen zwischen den zwei Modellen. Die Abkürzungen, die für die Begriffen der Modelle stehen, wurden in der Abbildung 4.2 dargestellt.

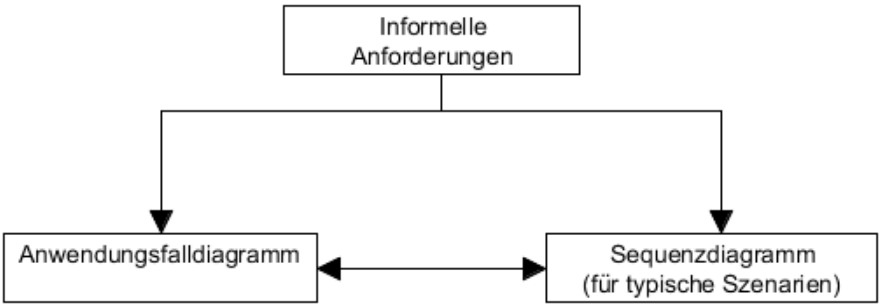

Abbildung 4.3.: Beziehungen zwischen Anwendungsfalldiagramm und Sequenzdiagramm[Pat10]

Den Ablauf der Transformation ergibt sich die Abbildung 4.3. Aus den klassifizierten textuellen Anforderungen können Anwendungsfalldiagramme, Sequenzdiagramme erzeugt. Ausgehend von den informellen Anforderungen werden zunächst Anwendungsfälle spezifiziert und durch Anwendungsfalldiagramme dargestellt. Daraus werden typische Szenarien identifiziert und mithilfe von Sequenzdiagrammen repräsentiert. Aus den informellen Anforderungen oder Sequenzdiagrammen können Anwendungsfälle auch in umgekehrter Reihenfolge beschrieben werden.

4.2. Gewählte Lösung

Die vorherigen Kapitel haben die Grundlagen und Konzepte beschrieben, dass ein System in dieser Arbeit entwickelt wird, das Modelle aus klassifizierten und modellbasierten Anforderungen erzeugt, um computergestützte Weiterverarbeitung zu ermöglichen. Folgend wird der Lösungsansatz zur Realisierung des Systems vorgestellt, in dem zwei Teilsysteme Klassifizierung und Transformation beinhaltet sind. Die Architektur des Systems ist illustriert in der Abbildung 4.4.
Die Eingabe des Systems ist ein textuelles Anforderungsdokument in englischer Sprache, mithilfe des Stanford Parsers wird das ursprüngliche Dokument schrittweise in ein modellbasiertes Anforderungsdokument umgewandelt, aus dem Anwendungsfalldiagramme und Sequenzdiagramme generiert werden. Den Übergang zwischen den zwei Teilsystemen trägt ein Dokument, das klassifizierte Anforderungen enthält.

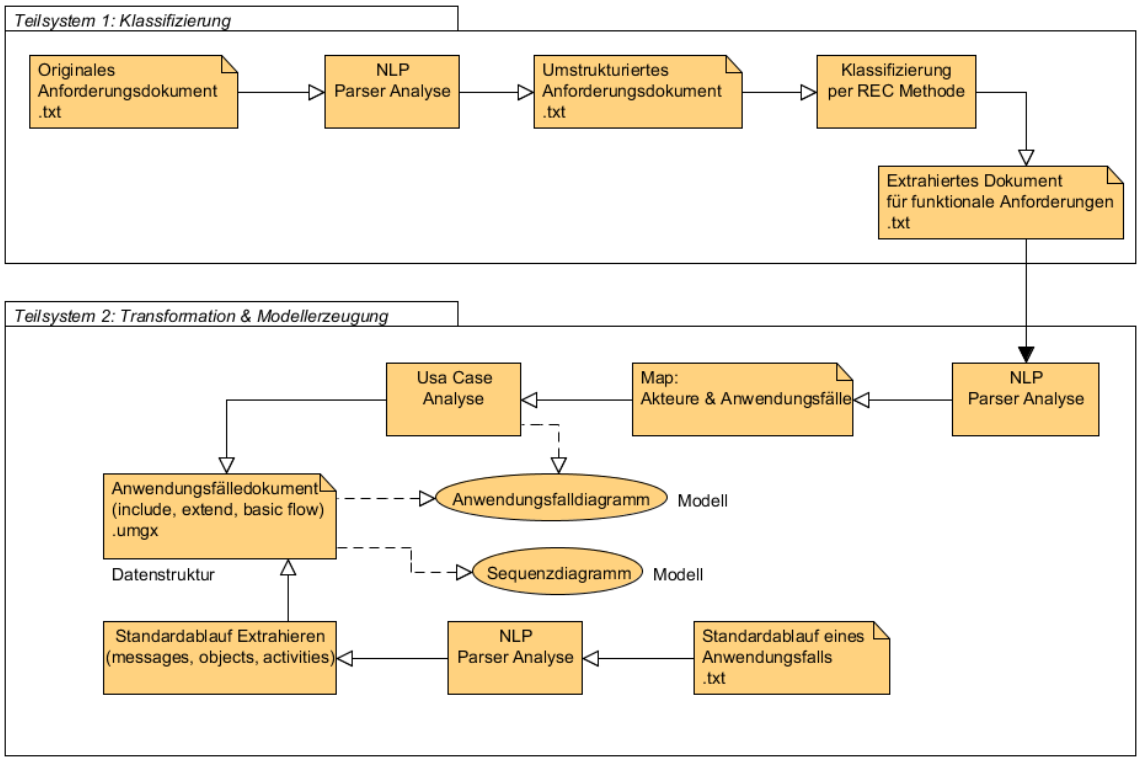

Abbildung 4.4.: Architektur des Systems

4.2.1. Klassifikation

Bei der Klassifizierung sollen die Anforderungen prinzipiell in Klassen unterteilt werden. Komplexe oder zusammengesetzte Sätze in den Anforderungsdokumenten, die keinen einfachen Satzbau haben, sollen in dem ersten Teilsystem zuerst mithilfe des Stanford Parsers in einfache Sätze (Simple Sentence) umgewandelt werden. Dies gilt als ein grundsätzlicher und wichtiger Schritt für das ganze System. Ein einfacher Satz (Simple Sentence), auch genannt als eine unabhängige Klausel, enthält ein Subjekt und ein Verb, er drückt eine vollständige Bedeutung aus. Im Gegensatz zu einem einfachen Satz enthält ein komplexer Satz eine unabhängige Klausel, die durch eine oder mehrere abhängige Klauseln verbunden ist. Ein Beispiel dafür ist wie folgt zu sehen:

- Komplexer Satz: John, who was the CEO of a company, played golf.
- Einfacher Satz: John played golf. John was the CEO of a company.

In dem komplexen Satz hat das Subjekt „John“ eine attributive-Klausel „who was the CEO of a company“, die durch die Umwandlung in ein einzelnen unabhängigen Satz mit dem gleichen Subjekt wird. Aus dem komplexen Satz werden zwei einfache Sätze, die auch die gleiche Bedeutung liefern. Neben komplexen Sätzen sollen zusammengesetzte Sätze in einfache Sätze umgewandelt werden. Solche Sätze haben üblicherweise zwei unabhängige Klausel, die durch eine Konjunktion wie „and“ und

„or“ verknüpft werden.
Für Dokumente, in denen ursprünglich nur einfache Sätze beinhaltet sind, werden die Determinatoren jedes Satzes wie z.B. „the“, „a“, „any“usw.erkannt und entfernt.

Wenn alle Sätze in einem Anforderungsdokument einfache Sätze sind, wird anschließend ein umstrukturiertes Dokument in dem Teilsystem 1 in der Abbildung 4.4 geliefert, mit dem die Klassifizierung anhand Ontologie von dem REC System durchgeführt wird.

Die Anforderungen aus dem umstrukturierten Dokument werden in funktionale und nicht-funktionale Klassen unterschieden. Eine Reihe von Regeln werden dazu verwendet, um die Unterteilung der Klassen auf logische und effiziente Weise zu ermöglichen. Die Ausgabe des Teilsystems 1 ist ein Dokument mit klassifizierten Anforderungen, die aus funktionalen bestehen. Nicht-funktionale Anforderungen werden hingegen nicht weiter verarbeitet.

4.2.2. Transformation und Modellerzeugung

Das Teilsystem der Transformation liefert als Ausgabe ein Datenmodell, aus dem Modelle der Anforderungen erzeugt werden können. Das Datenmodell ist ein modellbasiertes Anforderungsdokument und enthält wichtige Informationen der Anforderungen.

In dem Teilsystem wird ein Anforderungsdokument mit klassifizierten Anforderungen zunächst eingegeben, für das ein Stanford Parser verwendet wird, um Syntaxbäume (Parse Trees) für jeden Satz des Dokuments zu generieren. Die Syntaxbäume tragen zu der Analyse der Wörter in einem Satz bei und enthalten die Wörter als Kinder in einem Baum.

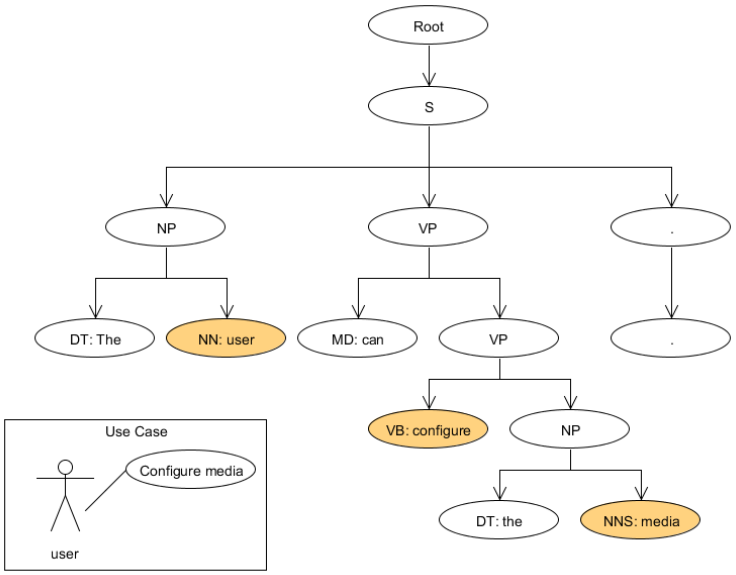

Abbildung 4.5.: Syntaxbaum für den Satz: The user can configure the media.

Ein einfacher Satz „*The user can configure the media.*“ ergibt einen Syntaxbaum (siehe Abbildung 4.5) unter Verwendung eines Stanford Parsers. In dem Syntaxbaum werden Subjekt, Verb und Objekt in orange Farbe markiert, deren POS den Worttyp repräsentieren.
Jeder Syntaxbaum hat eine Wurzel „Root“ , wobei die einzige Wurzel auch nur ein Kind hat. Die Buchstabe „S“steht für das Wort „Sentence“, alle Wörter in einem Satz sind deswegen Kinder von „S“. Ausgehend von „Root“ und „S“ ist es sichergestellt, dass ein Syntaxbaum nur einen Satz repräsentiert, egal ob der Satz ein einfacher oder komplexer Satz ist. Zur Vereinfachung der Analyse sind die Sätze jedoch einfache Sätze.
Mit einfachem Satz sind die Kinder von „S“ üblicherweise „NP“, „VP“ und schließlich ein Satzzeichen wie „.“, „!“ oder „?“. Bei dem obigen Beispiel ist das Satzzeichen ein Punkt zum Repräsentieren einer Aussage.

Aufgrund der Analyse des Syntaxbaums durch Stanford Parser lassen sich für Anwendungsfalldiagramm Akteur (actor) und Anwendungsfälle (use cases) bestimmen. Üblicherweise beziehen sich ein Akteur auf das Subjekt eines Satzes, das ein Verhalten durchführen soll. Für die Anwendungsfälle gelten in einem Satz Verb und Objekt, die das Verhalten des Subjekts beschreiben.
Basierend auf dem Syntaxbaum extrahiert man einen Anwendungsfall (an der linken Ecke der Abbildung 4.5) aus dem obigen Beispielsatz mit dem Akteur „user“ und dem Anwendungsfall (use case) „configure media“.

Zur Speicherung und Weiterverarbeitung der Akteuren und Anwendungsfälle eines Anforderungsdokuments wird eine interne Datenstruktur (siehe Abbildung 4.6) als Zwischenformat des Datenmodells verwendet, in der die Anwendungsfälle (use cases) auf entsprechende Akteuren oder Menge der Akteuren abgebildet sind, damit die Beziehungen zwischen Anwendungsfällen und Akteuren behalten werden.

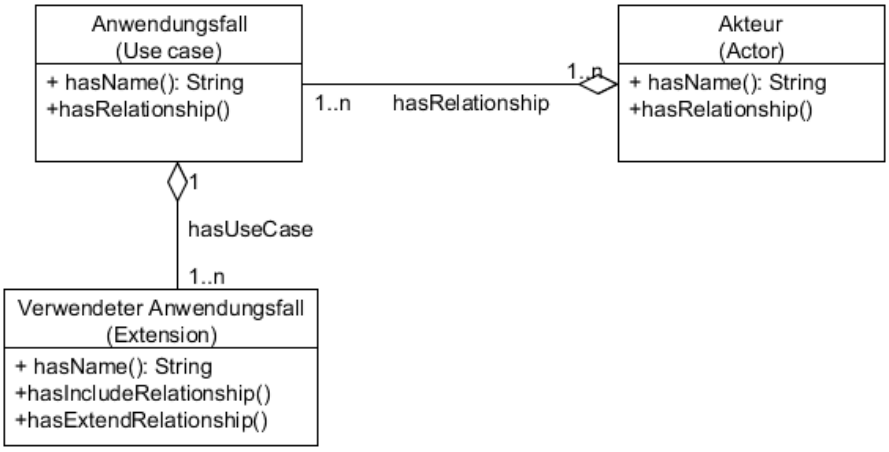

Abbildung 4.6.: Interne Datenstruktur zur Verarbeitung der Anwendungsfälle

Auf jeden Akteur werden mindestens ein Anwendungsfall abgebildet, umgekehrt kann jeder Anwendungsfall auch mehrere Akteuren haben. Die interne Datenstruktur behält alle Beziehungen

innerhalb von einem Anwendungsfall, auch die verwendeten Anwendungsfälle, die Assoziationen wie Importbeziehung (include), Erweiterungsbeziehung (extend) und Generalisierung (Generalization) zu den hauptsächlichen Anwendungsfällen generieren, werden in der Datenstruktur gespeichert.
Aus der internen Datenstruktur wird das Datenmodell als ein modellbasiertes Anforderungsdokument in Form von XML zu realisieren. In dem Dokument werden alle relevanten Anwendungsfälle und Akteuren, die die Anforderungen repräsentieren, als XML Elemente interpretiert. Änderungen zu den Anwendungsfällen inklusive entsprechender Akteuren in einem textuellen Dokument sind ebenfalls in der internen Datenstruktur zu ermöglichen. Dadurch werden die Vollständigkeit und Erweiterbarkeit der Anforderungen behalten.

Anwendungsdiagramme können direkt aus der internen Datenstruktur erzeugt werden. Auch wenn neue verwendete Anwendungsfälle hinzugefügt werden, ist ein neu bearbeitetes Anwendungsdiagramm durch das modellbasiertes Anforderungsdokument möglich zu realisieren.
Zur Erzeugung der Sequenzdiagramme existiert eine Möglichkeit. Ein Sequenzdiagramm liefert die Szenarien eines Anwendungsfalls. Ein Standardablauf eines normalen Anwendungsfalls entspricht den Szenarien für ein Sequenzdiagramm. Daher wird ein Sequenzdiagramm durch die Kombination des modellbasierten Anforderungsdokuments und des Standardablaufs erzeugt. Der Standardablauf kann in einem separaten Dokument enthalten werden. Dabei werden die Szenarien eines Anwendungsfalls in dem Dokument beschrieben und deren Reihenfolge wird eingehalten. Dadurch wird ein Sequenzdiagramm unter Verwendung des modellbasierten Anforderungsdokuments generiert.

4.3. Alternative Lösung

Für das Teilsystem der Klassifizierung wird der vorhandene Lösungsansatz in [Zwi13] in dieser Arbeit dem Ziel gemäß entsprechend erweitert, um die neuen Eigenschaften des anzustrebende Systems zu erfüllen. Eine alternative Lösung zur Klassifizierung wird deswegen nicht in Rücksicht genommen, weil das grundsätzliche Konzept bereits durch die Diplomarbeit festgelegt wurde.

Ein Ansatz zur Transformation und Modellerzeugung bezog sich auf die Anwendung von XML Schema Definition (XSD) und Extensible Stylesheet Language Transformation (XSLT). Klassifizierte Anforderungen in einem XML Dokument könnten in dem Ansatz programmatisch durch ein vordefiniertes XSD für ein spezifisches UML-Diagramm generiert werden.
Ein XSD definiert die Struktur jedes Element von dem UML-Diagramm im Detail, indem ein XML Schema die Eigenschaften eines Elements wie beispielsweise Position, Typ und Anzahl spezifiziert. Mit einem dem XSD entsprechenden XML Dokument wird Vorlagen einer XSLT Datei verwendet, um Scalable Vector Graphics (SVG)-Code aus dem XML Dokument zu generieren, der UML-Diagramm graphisch darstellen könnte.
Unter der Berücksichtigung der Effizienz und dem Zeitaufwand wurde der Ansatz jedoch nicht betrachtet. Abgesehen von der komplizierten und schwer begreifbaren Vordefinition eines XSD ermöglicht dessen Anwendung allerdings nur die Generierung spezifischer UML-Diagramme, welches die Weiterverarbeitung der Anforderungen einschränkt. Das Konzept über SVG könnte jedoch in der künftigen Arbeiten zur erfolgreichen Transformation einer Anforderung beitragen, weil SVG in der Regel interaktive und dynamische graphische Darstellungen aus XML Dokument erzeugen kann.

5. Implementierung

In diesem Kapitel wird die Implementierung des UML Model Generator (UMG) Systems vorgestellt. Von dem allgemeinen Lösungsansatz in Kapitel 4 ausgehend ist die Implementierung in zwei Teilsysteme gegliedert. Der Kern des UMG Systems besteht aus den Komponenten Klassifizierung und Transformation, die als Java-Programm umgesetzt wurden. Zunächst wird die entworfene Datenstruktur des UMGs vorgestellt. Der Programmablauf, bestehend aus den einzelnen Komponenten die jeweils ein Teilsystem implementieren, wird anschließend erläutert.

5.1. Datenstruktur

Die Architektur des Systems wurde bereits in Kapitel 4.2 eingeführt und erklärt. Anhand der Architektur wurde eine entsprechende Datenstruktur (siehe Abbildung 5.1) entworfen, die eine Kernkomponente für das UMG System ist.

5.1.1. Allgemeine Erläuterung der Datenstruktur

Die Datenstruktur repräsentiert die essentiellen Informationen des Systems, welche Beziehungen und Assoziationen zwischen dem Eingabedokument und den Ausgabemodellen entstehen. Neben den Wechselbeziehungen lassen sich die wichtigen Komponenten des UMGs deutlich und sachlich analysieren, wie sie ihren Zweck erfüllen und aus welchen Voraussetzungen realisiert werden können.

Ausgehend von der Datenstruktur ist es klarzustellen, dass die zwei Teilsysteme der Klassifizierung und der Transformation durch ein Dokument mit funktionalen Anforderungen miteinander verbunden sind. Vorteilhaft können nicht nur Modelle aus einem textuellen Dokument basierend auf der Datenstruktur realisiert werden, sondern kann die Daten in jedem Teilsystem auch bearbeitet werden können. Das bedeutet, dass UMG die Ausgaben jedes Teilsystems einzeln speichern kann. Somit können die Teilsysteme jeweils als ein unabhängiges Programm zur Verfügung stehen.

Die grundsätzliche Architektur des Teilsystems zur Klassifikation wurde in dem vergangenen Kapitel 4.2.1 erklärt. Die Eingabe davon ist ein originales textuelles Anforderungsdokument mit einfachen Sätzen, daraus entstehen zwei verarbeiteten Anforderungsdokumente als Resultate, die für verschiedene Zwecke bei der Anforderungsanalyse oder bei anderen Phasen der Produktentwicklung zur Verfügung gestellt werden können. Ein umstrukturiertes Dokument enthält nur einfache Sätze ohne irrelevanten Determinatoren, in dem die Sätze noch nicht in Anforderungsklassen geteilt werden. Der Inhalt des umstrukturierten Dokuments lässt sich die Klassifikationskomponente verarbeiten,

anschließend ist ein Dokument mit funktionalen Anforderungen verfügbar.
Somit kann man zwei einzelne Dokumente erhalten, die jeweils die Eingabe für das zweite Teilsystem verwendet werden können. Beide Dokumente können unter Verwendung des Stanford CoreNLP (siehe Kapitel 3.2.1) in XML Datei gespeichert werden.
Um das Ziel der Arbeit erfolgreich zu erfüllen, ist die Eingabe des zweiten Teilsystems jedoch nur das Dokument mit funktionalen Anforderungen.

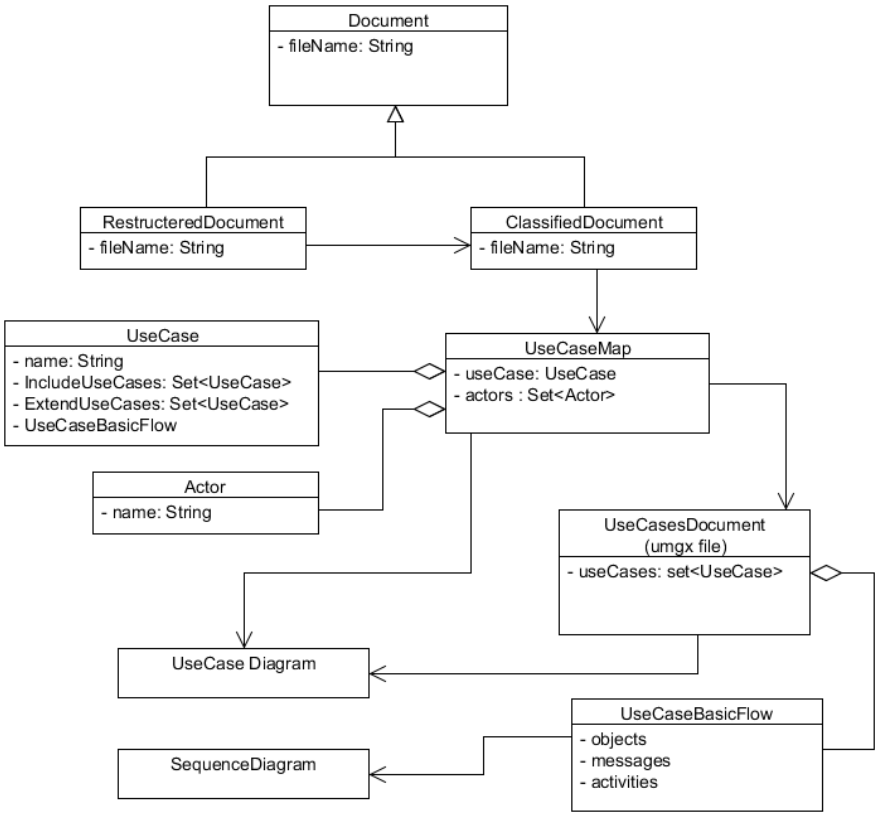

Abbildung 5.1.: Datenstruktur des Systems in Klassendiagramm

In dem Kapitel 4.2.2 wurde die grundsätzliche Methode der Transformation und Modellerzeugung erklärt, wie die Modelle von Anwendungsfalldiagrammen und Sequenzdiagramme aus dem in dem ersten Teilsystem erzeugten Dokument generiert werden.
Das UMG System bietet ein Dateiformat UMGX, das ein XML Format mit unbenanntem Formatnamen ist. Das von UMG generierte Datenmodell wird als modellbasiertes Anforderungsdokument in UMGX gespeichert. Das ganze Teilsystem der Transformation und Modellerzeugung beruht auf dem Datenmodell. In Kapitel 5.1.2 wird das Datenmodell ausführlich erläutert.

5.1.2. Datenmodell

In Kapitel 4.2.2 wurde eine interne Datenstruktur (siehe Abbildung 4.6) zur Verarbeitung der Anwendungsfälle vorgestellt, die ein Zwischenformat des Datenmodells ist. Ein Datenmodell des UMG Systems bezieht sich auf ein modellbasiertes Anforderungsdokument in spezielles Format von UMGX. Darauf aufbauend werden Modelle der Anforderungen eines Dokuments aus dem Teilsystem der Klassifizierung generiert und bearbeitet. Da das Ziel dieser Arbeit in der Modellerzeugung liegt, spielt deswegen das Datenmodell eine essentielle Rolle in dem UMG System. Ohne das Datenmodell ist die Verarbeitung der Anforderungen und deren Modelle nicht möglich.

Unter Berücksichtigung der Wichtigkeit des Datenmodells wird folgend dessen Struktur erläutert. Die Struktur des Datenmodells betrachtet man in der Abbildung 5.2. In dem Datenmodell sind eine Menge von Anwendungsfällen beinhaltet, die ebenfalls als Element in einem entsprechenden UMGX Dokument, das modellbasierte Anforderungen enthält, bezeichnet werden.

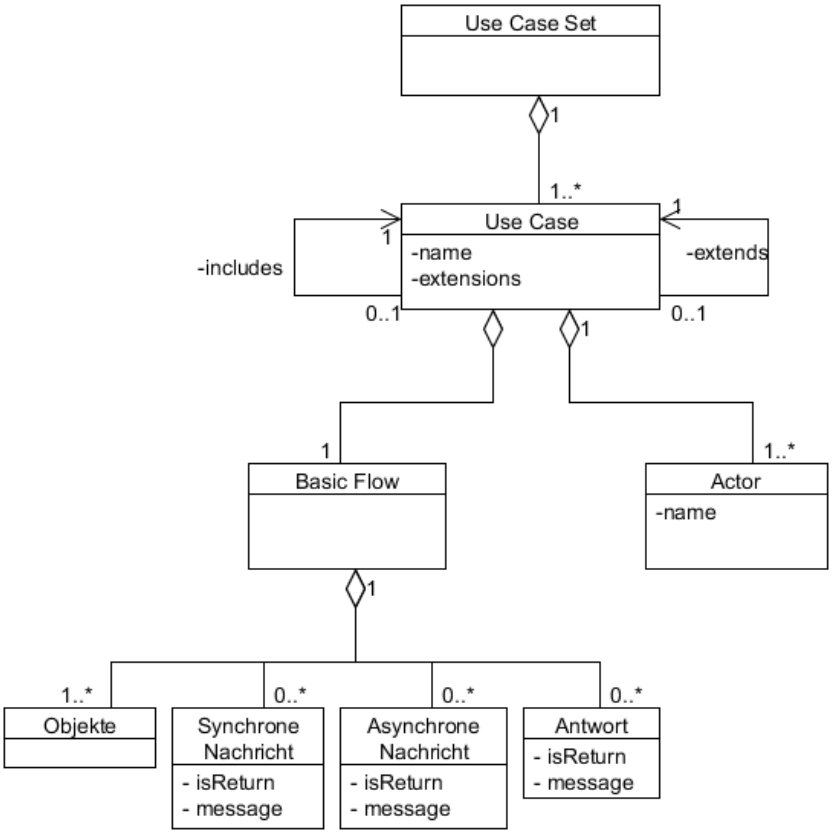

Abbildung 5.2.: Struktur des Datenmodells

Normalerweise ist jeder Anwendungsfall in dem Datenmodell mit einem Akteur assoziative, bei besonderen Fällen kann es auch sein, dass mehrere Akteure mit einem Anwendungsfall verknüpft

sind. Daher gibt es auch die Assoziation zwischen dem Anwendungsfall und einer Menge von Akteuren.
Ein Anwendungsfall haben Erweiterungsbeziehungen mit anderen Anwendungsfällen in dem Datenmodell. Zu den Erweiterungsbeziehungen gehören entweder Include-Beziehung oder Extendbeziehung.

Zusätzlich besitzt ein Anwendungsfall in der Regel verschiedene Szenarien, eine davon ist ein Standardszenario (Basic Flow), die anderen sind alternativen Szenarien. In dem UMG System wird nur Standardszenario eines Anwendungsfalls betrachtet.
Ein Szenario beschreibt eine Folge von Aktivitäten, die von einem Akteur ausgelöst werden und den Anwendungsfall aktivieren. Die Standardszenarien beziehen sich auf die hauptsächlichen Aktivitäten eines Anwendungsfalls, bei denen keine Probleme auftauchen.

Da jeder Anwendungsfall ein Element in einem mit dem Datenmodell übereinstimmten UMGX Dokument ist, sind alle Bestandteile des Anwendungsfalls ebenfalls Unterelemente des UMGX Dokuments.
Wie oben erwähnt hängen die Modellerzeugung von dem Datenmodell ab, sind die Änderungen zu jedem Anwendungsfall auch entsprechend in dem Datenmodell zugreifbar. Als Element wird daher der verarbeitete Anwendungsfall mit seiner Änderungen ebenfalls in dem UMGX Dokument gespeichert. Die Änderungen sind beispielsweise Hinzufügen der neuen Anwendungsfälle, der neuen Akteure zu einem vorhandenen Anwendungsfall oder der Erweiterungen.
Aus dem Datenmodell kann ein Anwendungsfalldiagramm generiert werden. Sequenzdiagramme sind die Visualisierung des Standardablaufs eines Anwendungsfalls. Daher sind Sequenzdiagramme von einem generierten Anwendungsfalldiagramm abgeleitet.
Die Erläuterung des Standardablaufs ist in Kapitel 5.2.4 zu sehen, in dem die Implementierung des Sequenzdiagramms vorgestellt wird.

5.2. Programmablauf

Klassifizierung und Modellerzeugung sind zwei Komponenten des UMG Systems, deren Implementierungen im Detail präsentiert werden.
Bei der ersten Komponente sind zwei Schritte vorgesehen: Zunächst muss mit Stanford Parser (siehe Kapitel 3.2.2) das eingegebene Dokument in ein umstrukturiertes Dokument umwandeln, welches das Format „.xml“ hat. In dem zweiten Schritt wird auch mit einem Stanford Parser das XML Dokument klassifizieren, wobei nur die Sätze, die funktionale Anforderungen sind, in einem neuen Dokument in Form von XML behalten.
Die zweite Komponente hat Funktionalität der Transformation und Modellerzeugung. Bei der Transformation muss das von der ersten Komponente erhaltene Dokument in ein modellbasiertes Anforderungsdokument umwandeln, in dem die funktionale Anforderungen als Anwendungsfälle dargestellt werden. Bei der Modellerzeugung wird ein Anwendungsfalldiagramm aus dem modellbasierten Dokument generiert, schließend wird Sequenzdiagramme für Anwendungsfälle erzeugt.

5.2.1. Teilsystem der Klassifizierung

In dem Teilsystem wird ein textuelles Dokument in englischer Sprache als Eingabe verarbeitet. Die Dokumentenumstrukturierung mithilfe von Stanford Parser entfernt alle Determinatoren jedes Satzes in dem Dokument, damit die Sätze nur eine einfache SVO-Tripel enthalten. Die Klassifizierung hilft dabei, funktionale Anforderungen von dem Dokument zu extrahieren.

Dokumentenumstrukturierung

Die Dokumentenumstrukturierung lässt sich durch den Einsatz des Stanford Parsers realisieren. Zwei Methoden wurden für die Erfüllung des Zwecks entwickelt.
Die erste Methode dient dazu, die Leerzeichen hinter jedem Satz zu überprüfen. Da jedes in UMG System eingegebenes Dokument satzweise durch den Stanford Parser analysiert wird, müssen die Sätze gültig für den Stanford Parser sein. Der Stanford Parser erkennt keinen Satz, hinter dem es kein Leerzeichen gibt. Ein solcher Satz wird mit dem nächsten Satz zusammen als ein Ganzes analysiert. Daher wird eine Vormethode implementiert, die alle Sätze mit einem Leerzeichen hinter dem Satzzeichen ermöglicht. Die verarbeiteten Sätze können in der weiten Implementierungen problemlos analysiert werden.
Die zweite Methode für Dokumentenumstrukturierung ist eine Methode zur Entfernung irrelevanten Determinatoren jedes Satzes. Durch die POS jedes Wortes werden die Determinatoren festgelegt, wenn sie keine richtige und direkte Auswirkung auf die Vollständigkeit eines Satzes haben.
Um die verarbeiteten Sätze zu speichern, kommt ein XML Dokument zum Einsatz.

Klassifizierung

Mit dem umstrukturierten Dokument kann man die Klassifizierung durchführen.
Zuerst werden die Sätze in dem zu klassifizierenden Dokument überprüft, die alle einfachen Sätze sind und SVO-Tripel haben, ob sie sich auf funktionale Anforderungen oder nicht-funktionale Anforderungen beziehen. Dieser Schritt ist ein manueller Schritt für die Anforderungsanalysten. Obwohl die Anforderungen meistens mit Modalverben erfassen werden, kann es jedoch auch möglich sein, dass manche Anforderungen nur normale Aussagen ohne Modalverben sind. Deshalb zählt die Erkennung der Modalverben nicht zu einen optimalen Lösung zur Unterscheidung der funktionalen und nicht-funktionalen Anforderungen.

Die Regeln für die Unterscheidung der Anforderungen liegen vor, dass den Verben der Sätze, die als funktionale Anforderungen betrachtet werden, Modalverben hinzugefügt werden und die Sätze, die als nicht-funktionale Anforderungen betrachtet werden, bleiben unverändert.
Um die Realisierung der Klassifizierung zu vereinfachen, sind die Regeln zu den funktionalen Anforderungen wie folgt zu sehen:

1. Bei den funktionalen Anforderungen bezüglich eines externen Nutzers haben die Verben folgende Darstellung:
   Modalverb „can“ + normales Verb

2. Bei den funktionalen Anforderungen bezüglich eines internen Systems haben die Verben folgende Darstellung:
   Modalverb „must“ + normales Verb
3. Wenn die Verben eines Satzes eine Zusammensetzung (phrase) sind, werden die Verbgruppe durch andere normalen Verben mit der gleichen Bedeutung ersetzt.
4. Beinhalten die funktionalen Anforderungen noch alternativen Anforderungen, fangen die alternativen Anforderungen immer mit dem Wort „If“ an.

Beispiel: Customer can withdraw cash from an account. If the account is invalid, the system must request a new account.
Durch die Regeln werden die funktionalen Anforderungen entsprechend modifiziert, während die nicht-funktionalen Anforderungen immer unverändert bleiben.

Auf dem Kriterium des REC Systems basierend werden die Regeln eingeführt. Ein Vorteil von den Regeln zu den funktionalen Anforderungen liegt vor allem bei der Festlegung der Anwendungsfälle in dem zweiten Teilsystem.
Der Standford Parser analysiert in dem zweiten Teilsystem die Sätze des Dokuments mit extrahieten funktionalen Anforderungen anhand der POS ihrer Subjekten und Verben. In englischer Sprachen können manche Verben in verschiedenen Formen eine unterschiedliche Bedeutung haben. Beispielsweise kann das englische Wort „request“ in dem obigen Beispielsatz bei verschiedenen Kontexten auch ein Normen sein, das die Bedeutung „Anfrage“hat.

In den folgenden Beispielsätzen, die durch Stanford Parser analysiert wurden, ist das Wort „request“ durch zwei POS kennzeichnet:

- The/DT system/NN requests/NNS a/DT new/JJ account/NN ./.
- The/DT system/NN must/MD request/VB a/DT new/JJ account/NN ./.

In dem ersten Satz hat „requests“ die POS „NNS“ . Laut [Atw90] bedeutet die POS „NNS“ plurales Normen, obwohl das Wort „requests“ in dem Satz als ein singularer Verb bezüglich des Wortes „system“ eingesetzt wurde. Stanford Parser ist zwar ein intelligentes Werkzeug, es kann zwischen pluralem Normen und singularem Verb mancher Wörter nicht unterscheiden. Der zweiter Beispielsatz, in dem das Wort „request“ dank dem Modalverb mit richtiger POS kennzeichnet wurde, ist ein geeignete funktionale Anforderung.

Es wurde eine Methode zur Klassifikation entwickelt, indem das umstrukturierte Dokument als Eingabe in der Methode verarbeitet wird.
Durch die Analyse des Stanford Parsers erhalten die Sätze in dem Dokument ihren Syntaxbaum, der in der Abbildung 4.5 des Kapitel 4.2.2 bereits eingeführt und vorgestellt wurde. Jedes Wort eines Satzes ist ein Kind von dem Syntaxbaum, wobei den Wörtern die Gruppen „NP“, „VP“ und Satzzeichen zugeordnet sind.

Um funktionale Anforderungen von dem Dokument zu extrahieren, wird die Gruppe „VP“ betrachtet. Dem Unterbaum „VP“ entlang ist das Modalverb in den Sätzen, die als funktionale Anforderungen betrachtet werden, leicht zu finden. Da die POS aller Modalverben „MD“ ist, ermöglicht es den

Durchlauf bei dem Unterbaum „VP“ , aller Modalverben zu finden. Wegen des einfachen Satzbaus haben die zu extrahierenden Sätze eine einfache SVO-Tripel, dabei ist das Modalverb das erste Kind von dem Unterbaum „VP“ . Für die Sätze, die alternative Anforderungen sind und mit „If“ anfangen, wird das erste Kind ihrer Syntaxbäume überprüft. Somit werden die Modalverben erkannt und die Sätze, die solche Regeln erfüllen, werden in einem klassifizierten Dokument gespeichert. Die restlichen Sätze, die die Regeln nicht erfüllen, werden als nicht-funktionale Anforderungen betrachtet und nicht gespeichert.

5.2.2. Transformation in modellbasiertes Anforderungsdokument

Nach der Klassifizierung wird das Dokument in dem zweiten Teilsystem verwendet, um ein modellbasiertes Anforderungsdokument und darauf aufbauend Modelle zu erzeugen. Das modellbasiertes Dokument, der das Datenmodell des UMG Systems ist, dient zu der hauptsächlichen Verwaltung der Anforderungen und der Modelle. Vier Funktionalitäten wurden für die Komponente entwickelt.

Einführung des Stanford Parsers

Die erste Funktionalität bezieht sich auf den Einsatz des Stanford Parsers. Wie in dem obigen Kapitel angesprochen wurde, ist die Generierung des Syntaxbaums für jeden Satz ein wichtiger Schritt für Transformation. Die Methode nimmt ein Dokument mit funktionalen Anforderungen als Eingabe und liefert die Syntaxbäume für jeden Satz anhand der Vordefinition des Stanford Parsers zurück. Die Ausgabe der Methode ist eine Liste von Syntaxbäumen aller Sätze. Es gibt keine Zwischenspeicherung für die Liste von Syntaxbäumen, weil sie in der nächsten Methode als Eingabe zum Einsatz gelangen wird.

Extrahieren der SVO-Tripel

Bei der zweiten Funktionalität handelt es um das Extrahieren der SVO-Tripel in jedem Satz. Die in dem klassifizierten Dokument beinhalteten Sätze bestehen alle aus Subjekt, Verb und Objekt. Dabei werden für die Tripel jeweils eine Untermethode entwickelt, die das entsprechende und gültige Wort zu extrahieren.

Um die Untermethoden sachlich zu entwickeln, wurden drei Algorithmen dafür definiert. In [RDF07] wurden drei verschiedene Lösungsansätze anhand drei Parser vorgestellt, einer davon hat auch den Baumbank von Stanford Parser verwendet, der den Syntaxbaum liefern kann. Darauf aufbauend behält der Algorithmus in der Methode zum Teil das allgemeine Konzept von dem Algorithmus in [RDF07] und entwickelt zusätzlich noch spezielle Eigenschaften.

In dem folgenden Algorithmus 5.1 handelt es um das Extrahieren des Subjekts und des Verbs in einem Syntaxbaum. Der Algorithmus 5.2 beschreibt die Fallunterscheidungen bei dem Extrahieren des Objekts.
Jeder Algorithmus liefert einen Baum zurück, weil in dem Syntaxbaum jedes Wort auch einen einzelnen kleinen Baum ist. Daher ist die Ausgabe der Methode zum SVO-Tripel Extrahieren eine

Liste von Syntaxbäumen, die Subjekt, Verb und Objekt eines Satzes enthält, wobei der Baum des Objekts hinter an dem Baum des Verbs angehängt wird, so dass Verb und Objekt in einem Baum sind.

Algorithmus 5.1 Algorithmus zum Extrahieren der Subjekt und Verb in einem Syntaxbaum

```
1:  function SubjectExtraction ( Tree )
2:      subject ← first noun found in NPTree
3:      if subject contains modifier then
4:          remove modifier
5:          result = subject - modifier
6:      else
7:          result = subject
        return result
8:
9:  function VerbExtraction ( Tree )
10:     if VPTree not contain VPSubTree then
11:         verb ← first verb found in VPTree
12:         result = verb
13:     else
14:         if VPSubtree not contain VPSubsubTree then
15:             verb ← first verb found in VPSubTree
16:             result = verb
17:         else if VPSubTree contain VPSubsubTree then
18:             verb ← first verb found in VPSubsubTree
19:             result = verb
        return result
20:
```

Algorithmus 5.2 Algorithmus zum Extrahieren des Objekts in einem Syntaxbaum

```
1:  function ObjectExtraction ( Tree )
2:      if VPTree not contain VPSubTree then
3:          if VPTree contain NPSubTree and PPSubTree then
4:              object ← noun and preposition in the both sub trees
5:              result = object
6:          else if VPTree contain only PPSubTree then
7:              object ← all words of PPSubTree
8:              result = object
9:          else
10:             object ← all words of second child in VPTree
11:             result = object
12:     if VPSubtree contain VPSubsubTree then
13:         if VPSubsubTree contain NPSubTree and PPSubTree then
14:             object ← noun and preposition in the both sub trees
15:             result = object
16:         else if VPSubsubTree contain only PPSubTree then
17:             object ← all words of PPSubTree
18:             result = object
19:         else
20:             object ← all words of second child in VPSubsubTree
21:             result = object
        return result
22:
```

Generierung der Anwendungsfälle

Die dritte Funktionalität verwaltet die Generierung der Menge von Anwendungsfällen, in der für jeden Anwendungsfall auch sein Akteur erkannt wird. Die von der vorletzten Methode generierten Liste der Syntaxbäumen von SVO-Tripel wird als die Eingabe in der Methode eingegeben, das bedeutet, dass die Liste der Syntaxbäume nochmal von dem Stanford Parser analysiert werden.
Die POS von der SVO-Tripel werden betrachtet. Besitzt ein Wort in der Liste die POS von der Kategorie „NP“, wird das Wort als ein Akteur gespeichert. Da Objekt ein Anhänger von Verb ist, ist VO-Tripel nur ein einzelner Baum und wird mit POS von der Kategorie „VP“ kennzeichnet. Daher werden alle Wörter, die sich in dem VO-Syntaxbaum befinden, als ein Anwendungsfall gespeichert.

Mit der Generierung der Anwendungsfälle und Akteure wird ein assoziatives Datenfeld entwickelt, das im Java Programm ein ein „Map“ ist. In dem assoziativen Datenfeld wird ein Anwendungsfall auf ein Akteur abgebildet. In dem vergangenen Kapitel 4.2.2 wurde die Map (siehe Abbildung 4.6), die auch ein Zwischenformat des Datenmodells ist, vorgestellt. Dies wird zum Ende der Transformationskomponente als ein modellbasiertes Anforderungsdokument in Format von UMGX gespeichert, das zu der Erzeugung der Modelle für die Anforderungen beitragen kann.

Generierung des UMGX Dokuments

In der letzten Funktionalität wurde ein UMGX Dokument generiert. Ein UMGX Dokument ist ein modellbasiertes Anforderungsdokument, welches das Datenmodell des UMG Systems repräsentiert. In dem Dokument werden die Anwendungsfälle und deren Akteure von dem assoziativen Datenfeld in UMGX-Elemente umgewandelt und gespeichert. Von der Änderung des Inhalt des Dokuments bis hin zu Laden und Speichern des Dokuments kann das erhaltene UMGX Dokument beliebig verarbeitet werden.

5.2.3. Modellerzeugung des Anwendungsfalldiagramms

Die Generierung der Anwendungsfälle und Akteuren ermöglicht die Modellerzeugung aus dem modellbasierten Anforderungsdokument. In dieser Arbeit wird zuerst ein Anwendungsfalldiagramm generiert, das die grundsätzlichen Informationen der funktionalen Anforderungen präsentiert.

Um die Modelle zu interpretieren wurde in der Komponente ein grafischer Editor (siehe Abbildung 5.3) entwickelt. Zur Entwicklung des grafischen Editors wurden im UMG System eine Java Programmschnittstelle und Grafikbibliothek Java Swing sowie eine Standard-API Java AWT zur Erzeugung und Darstellung einer plattformunabhängigen grafischen Benutzerschnittstelle verwendet. Der Editor setzt sich aus drei Bereichen zusammen: ein mittlere Bereich für Modelldarstellung, ein oberer Bereich für Haupttasten und ein rechter Bereich für die Tasten der Erweiterungen jedes Anwendungsfalls.

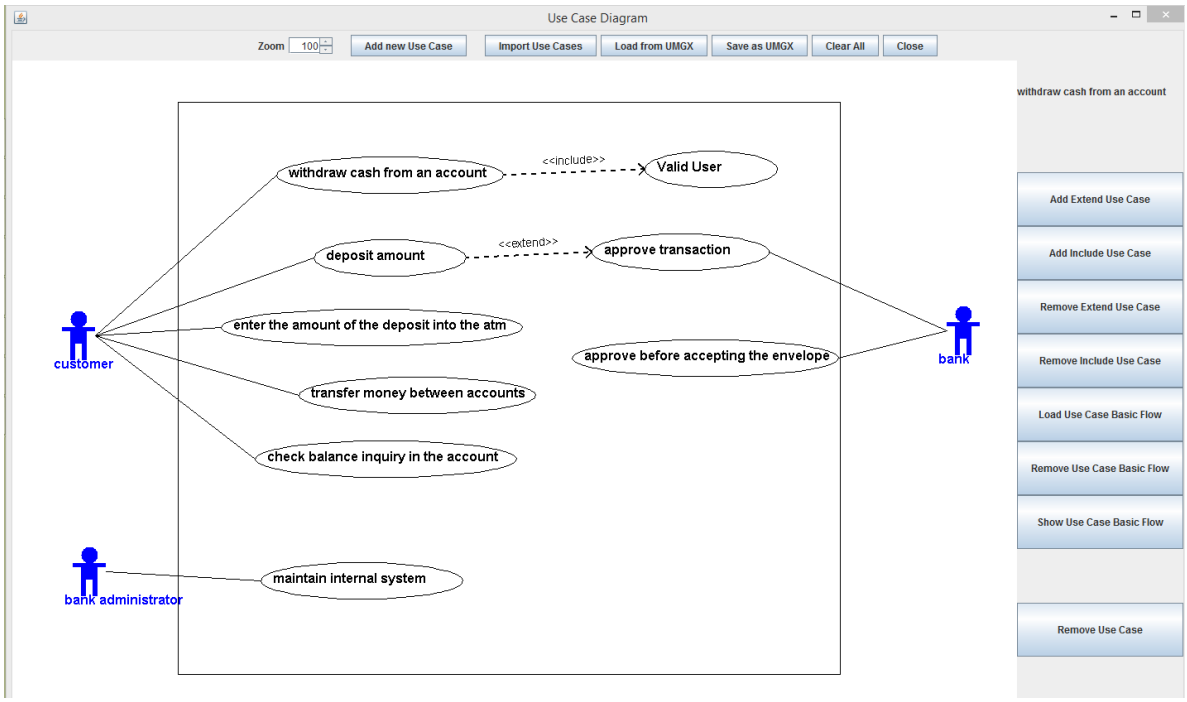

Abbildung 5.3.: Grafischer Editor für Modelldarstellung

Darstellungsbereich

In der obigen Abbildung ist ein mittlerer Bereich zur Darstellung des Anwendungsfalldiagramms zu sehen. Lassen sich Anwendungsfälle von einem Text Dokument mit funktionalen Anforderungen oder von einem UMGX Dokument importieren, wird in dem Darstellungsbereich ein Anwendungsfalldiagramm gezeichnet, indem Akteuren in Form von einem blauen Männchen und Anwendungsfälle durch eine Ellipse mit Text dargestellt werden. Alle Änderungen zu dem grafischen Diagramm werden entsprechend in dem Bereich gezeigt.

Haupttastenbereich

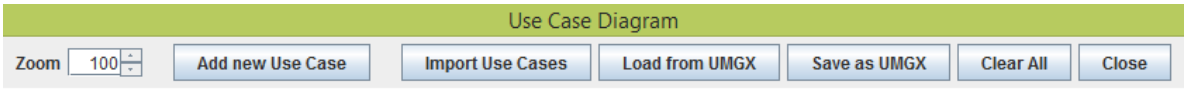

Abbildung 5.4.: Haupttastenbereich mit Funktionstasten.

Der Haupttastenbereich befindet sich oben in dem grafischen Editor und enthält Tasten, die für die Darstellung des Anwendungsfalldiagramms einschließlich dessen Erweiterungen zuständig sind.

Betrachtet von links nach rechts hat die erste Taste eine Zoom-Funktion mit dem Umfangreich von 50 bis 200, durch die die Größe aller Elemente des Anwendungsfalldiagramms gesteuert werden kann. In der Abbildung 5.3 hat das Diagramm die Größe von 100. Die Zoom-Funktion ermöglicht eine geeignete Darstellung des Diagramms in Bildschirmen mit verschiedenen Größen, ohne wichtige Informationen zu verlieren.

Mit der zweiten Taste der linken Seite werden neue Anwendungsfälle sowie deren Akteuren hinzugefügt. Durch die Funktion kann man direkt in dem grafischen Editor eigenes Anwendungsfalldiagramm erstellen, ohne Anwendungsfälle zuerst von einem Dokument zu importieren.
Die hinzugefügten Elemente können ebenfalls als ein modellbasiertes Dokument in Form von UMGX gespeichert werden.
Die Einführung der Anwendungsfälle aus einem Dokument mit funktionalen Anforderungen ermöglicht die Taste *Import Use Cases*. Kickt man auf die Taste, wird ein Fenster für Dateiladen gezeigt. Dabei ist die Dateiformat auf TXT-Datei eingeschränkt. Nach dem erfolgreichen Laden einer gewählten Datei ist ein entsprechendes Anwendungsfalldiagramm in dem grafischen Editor zu generieren.

Ein in dem grafischen Editor korrekt erstelltes Anwendungsfalldiagramm kann man wiederum in eine UMGX-Datei speichern. Die Datei ist als ein modellbasiertes Anforderungsdokument betrachtet, weil sie die Basis weiterer Verarbeitungen anhand der Anforderungen bietet. Die Funktion erfolgt durch die Taste *Save as UMGX*. Beispiele für die UMGX-Dateien sind in Kapitel 6 zu finden.

Gleichermaßen funktioniert die Taste *Load from UMGX* wie die Taste *Import Use Cases*. Unterschied zwischen den beiden Funktionen liegt jedoch daran, dass die durch Taste *Load from UMGX*

geladenen Dateien das Format von UMGX haben, während die Taste *Import Use Cases* nur TXT-Dateien erlaubt. Vorteilhaft kann man, nachdem ein Anwendungsfalldiagramm in Form von UMGX gespeichert wird, nur die UMGX-Datei importieren und sie bearbeiten, ohne die ursprünglichen Anwendungsfälle wiederum zu laden. Dadurch können alle in der UMGX-Datei behaltenen Änderungen zu dem Anwendungsfalldiagramm grafisch dargestellt werden.

Die letzten zwei Tasten auf der rechten Seite des Bereiches dienen jeweils dazu, gesamte Inhalte in dem Darstellungsbereich zu entfernen und den grafischen Editor zu schließen.

Erweiterungsbereich

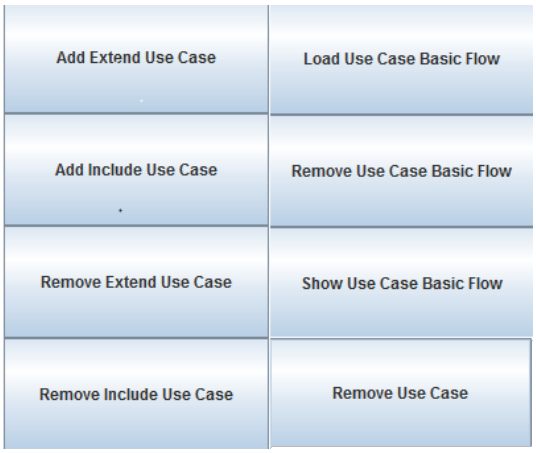

Abbildung 5.5.: Erweiterungsbereich für einzelnen Anwendungsfall eines Anwendungsfalldiagramms

Der Tastenbereich wurde für jeden einzelnen Anwendungsfall eines Anwendungsfalldiagramms entworfen. Klickt man auf die Tasten *Add Extend Use Case* oder *Add Include Use Case*, werden Extend-Beziehung oder Include-Beziehung zu dem Anwendungsfall realisiert. In der Abbildung 5.3 hat der Anwendungsfall „withdraw cash from an account“ beispielsweise eine Include-Beziehung mit dem Anwendungsfall „Valid User“. Zwei *Remove*-Tasten dienen dazu, die erstellten Erweiterungsbeziehungen zu entfernen.

Ein wichtiger Bestandteil des inhaltlichen Aufbaus von einem Anwendungsfall ist ein Standardablauf, der das typische Szenario dargestellt, das leicht zu verstehen oder der am häufigsten vorkommende Fall ist. Ein Standardablauf lässt sich durch eine geladene Datei aus der Datenbank generieren, dabei ermöglicht die Taste *Load Use Case Basic Flow* diese Funktion. Nach einer erfolgreichen Generierung des Standardablaufs erhält man dessen grafische Darstellung in Form von Sequenzdiagramm mit dem Klicken auf die Taste *Show Use Case Basic Flow*.
Da der Standardablauf zu den Informationen eines in dem grafischen Editor erstellten Anwendungsfalldiagramms gehört, wird er mit in dem UMGX Dokument gespeichert.

Die zwei restlichen *Remove*-Tasten in dem Bereich dienen auch jeweils dazu, der erstelle Standardablauf zu entfernen oder den gesamten Anwendungsfall zu entfernen. Ein Akteur wird automatisch entfernt, falls alle mit dem Akteur verbundenen Anwendungsfälle entfernt sind.

5.2.4. Modellerzeugung des Sequenzdiagramms

Wie bereits in dem vorherigen Abschnitt erwähnt wurde, dass ein Sequenzdiagramm den Standardablauf eines Anwendungsfalls repräsentieren kann, wird in dem Abschnitt die Erzeugung des Sequenzdiagramms vorgestellt. Bei der Funktionalität wird ein textuelles Dokument mit dem Standardablauf des Anwendungsfalls in UMG System eingegeben, die Erkennung der Objekte und Nachrichten wird automatisch realisiert.

Die Abbildung 4.4 in Kapitel 4.2 hat die Architektur von dem UMG System illustriert. Ausgehend von der Architektur wird ein Sequenzdiagramm durch eine Reihe von Analyse des Standardablaufs generiert. Ein eingehendes Dokument über Standardablauf eines Anwendungsfalls wird durch einen Stanford Parser analysiert, damit Objekte und Nachrichten extrahiert werden können, die wichtige Bestandteile eines Sequenzdiagramms sind.

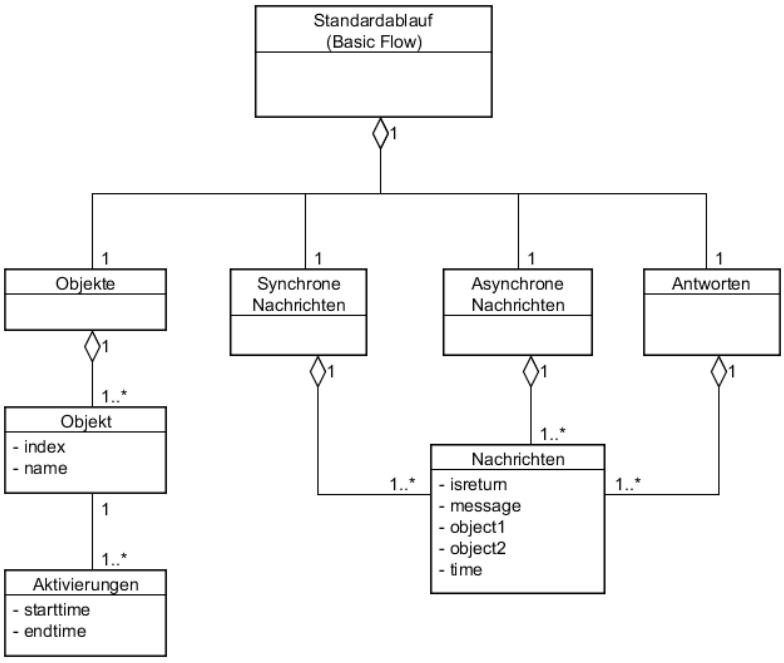

Abbildung 5.6.: Struktur des Standardablaufs

Die Abbildung 5.6 illustriert die Struktur eines Standardablaufs, die von UMG generiert wird. Jeder Standardablauf enthält eine Menge von Objekten und Nachrichten von drei Typen, die synchron, asynchron oder eine Antwort auf andere Nachrichten sind.

In der Menge von Objekten ist ein einzelnes Objekt dargestellt mit mehreren Aktivierungen, indem die Startzeit und die Endzeit der Aktivierungen die Lebenszeit des Objekts präsentieren. Eine Aktivierung ist der Bereich, in dem eine Methode des Objektes ausgeführt wird.

Eine synchrone Nachricht löst immer eine Aktivierung aus, die so lange dauern wird, bis eine Antwort auf die eingehende synchrone Nachricht kommt. Der Aufruf einer synchronen Nachricht erfolgt daher von der Quelle zum Ziel. Das Zielobjekt muss eine entsprechende Antwort schicken, während das Quellobjekt mit der weiteren Verarbeitung wartet, bis das Zielobjekt seine Verarbeitung beendet hat und setzt die Verarbeitung fort[Gui14].
Eine asynchrone Nachricht erfolgt ebenfalls von der Quelle zum Ziel. Das Quellobjekt wartet mit der Verarbeitung nicht auf das Zielobjekt, sondern setzt seine Arbeit nach dem Senden der Nachricht fort. Asynchrone Nachrichten werden verwendet, um parallele Threads zu modellieren[Gui14].

Das UMG System hat die Struktur des Standardablaufs zur Erzeugung des Sequenzdiagramms verwendet. Der grafische Editor bietet eine Funktion, das Sequenzdiagramm (siehe Abbildung 5.7) darzustellen.

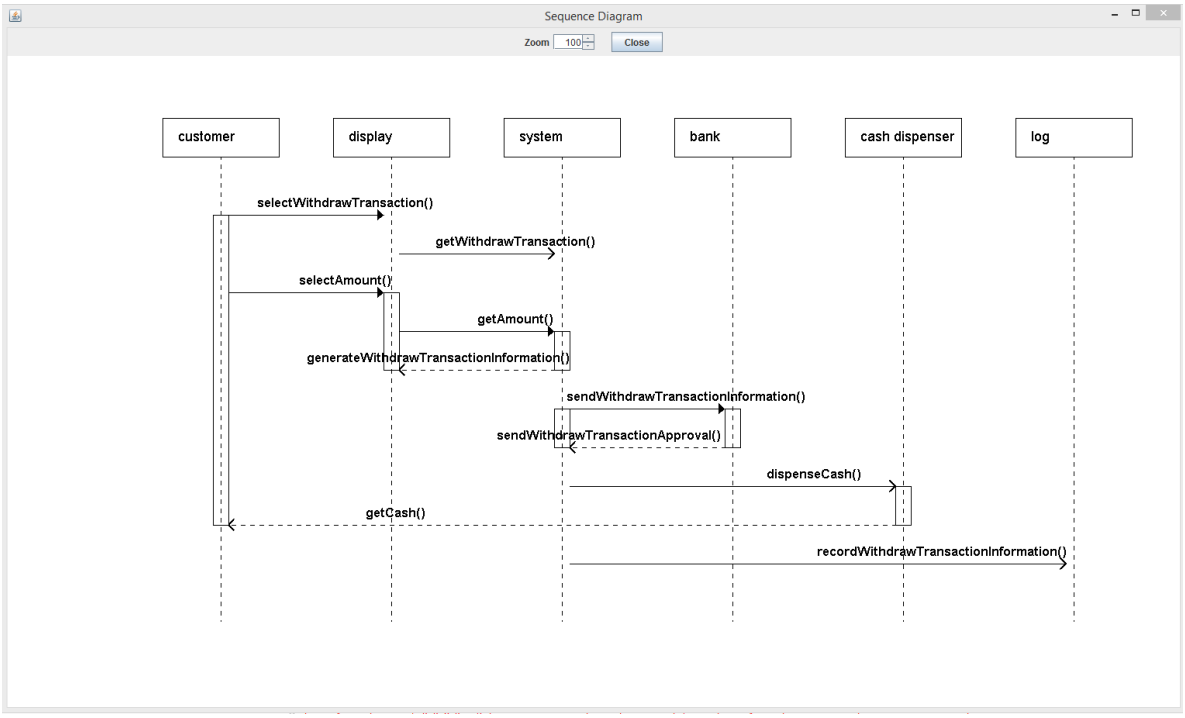

Abbildung 5.7.: Sequenzdiagramm illustriert den Standardablauf eines Anwendungsfalls *withdraw cash from an account.*

Obwohl diese grafische Benutzeroberfläche für Sequenzdiagramm nicht bearbeitbar ist, kann man dabei die Größe des ganzen Diagramms mit der Zoom-Funktion ändern.

In dem von UMG generierten Sequenzdiagramm setzt sich jedes Objekt aus einem Rechteck mit dem Namen des Objekts und einer senkrechten, gestrichelten Linie unterhalb des Rechtecks zusammen. Die Linie stellt die Lebenslinie eines Objekts dar, die den zeitlichen Bereich repräsentiert, in dem das Objekt existiert. Die Aktivierungen eines Objekts werden durch ein schmales Rechteck auf der gestrichelten Linie dargestellt. Der Index eines Objekts wird von UMG anhand der Position des Objekts in dem Standardablaufdokument festgelegt.
Nachrichten werden als Pfeile zwischen den Aktivierungen bezeichnet. Die Pfeile mit einer ausgefüllten Spitze bezeichnet eine synchrone Nachricht, während eine asynchrone Nachricht durch die Pfeile mit einer offenen Pfeilspitze bezeichnet. Eine Antwort, die abhängig von einer synchronen Nachricht ist, wird durch eine gestrichelte Pfeile dargestellt.

6. Evaluierung

In diesem Kapitel wird die Evaluierung des UMG Systems vorgestellt. Der UMG hat die Funktionalitäten, ein textuelles Anforderungsdokument in ein modellbasiertes Anforderungsdokument umzuwandeln und somit Modelle der Anforderungen zu erzeugen. Ziel der Evaluierung bestand darin, Präzision und Vollständigkeit der Funktionen von UMG zu überprüfen. Stimmen die Ergebnisse der Evaluierung mit den Anforderungen des UMG überein, kann man feststellen, dass die Evaluierung erfolgreich ist und das UMG System gute Leistung liefern kann.

Um die Funktionalitäten des UMG Systems zu evaluieren, wurden drei Dokumente als Musterexemplare der Evaluierung verwendet, die Anforderungen an ein zu entwickelndes System beschreiben. Das erste Dokument „Automat Teller Machine (ATM) System“ stammt von [Bjo04], in dem die Anforderungen an das ATM System beschrieben wurden. Das zweite Dokument namens „Address Book System“ stammt von [Bjo05] enthält die Anforderungen an das System für ein Adressbuch. In dem letzten Dokument [VP11] handelt es sich um die Anforderungen an ein „Online Broadcasting Services System“ von einem Unternehmen OTV.

Die drei Dokumente wurden wegen dem Umfang des Inhaltes gewählt. Das erste Dokument hat insgesamt 31 Sätze und den höchsten Umfang. 21 Sätzen hat das zweite Dokument und einen niedrigeren Umfang gegenüber dem ersten Dokument. Der niedrigste Umfang des Inhaltes hat das letzte Dokument mit der gesamten Anzahl der Sätze von 17.
Der Unterschied zwischen den drei Dokumenten spiegelt sich bei der Performanz und Leistung des UMG Systems wider. In den drei Dokumenten sind sowohl funktionale Anforderungen als auch Sätze, die nicht-funktionale Anforderungen sind, beinhaltet.

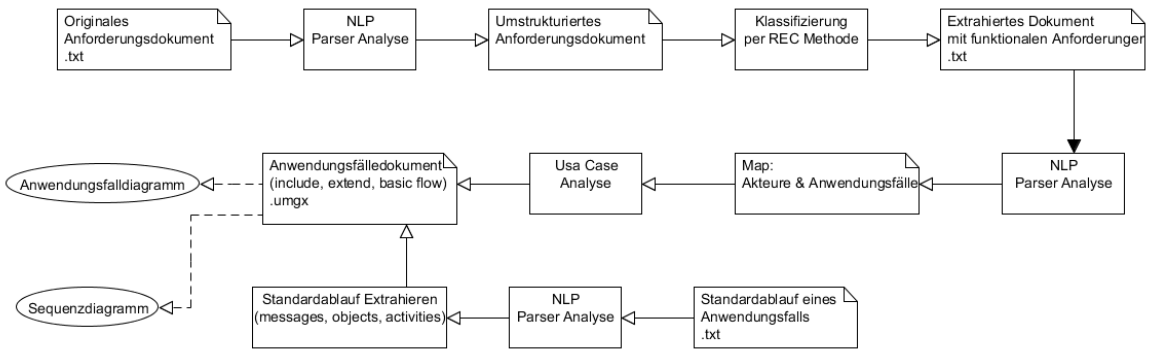

Abbildung 6.1.: Pipeline des Evaluierungsablauf

Der Evaluierungsablauf bestand darin, ein Dokument durch das UMG System anhand der Architekturpipeline des UMG durchzuführen.
Die Evaluierungspipeline (siehe Abbildung 6.1) besteht aus drei Teilpipelines, welche die entsprechenden Teilfunktionalitäten von dem UMG System realisieren. Bei der Evaluierung wurden alle Teilpipelines für jedes Dokument separat evaluiert, indem die jeweilige Eingabe eingegeben wurde und die Ausgabe geprüft wurde.

6.1. Evaluierung der Teilpipeline: Klassifizierung

Die erste Teilpipeline (Abbildung 6.2) führt die Evaluierung des Teilsystems Klassifizierung durch. Die drei Dokumente wurden in der Teilpipeline hintereinander mit dem Teilsystem des UMG Systems bearbeitet, die Ergebnisse der Bearbeitung wurden überprüft und bewertet.

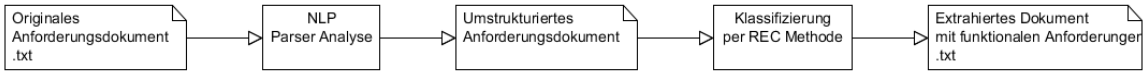

Abbildung 6.2.: Teilpipeline der Klassifizierung

6.1.1. Evaluierung des Dokuments ATM System

Zuerst wurde das originale Dokument „ATM System“ als Eingabe in der ersten Teilpipeline getestet. Die Ausgabe soll ein Dokument mit extrahierten funktionalen Anforderungen sind, die Anwendungsfälle repräsentieren können. Der Inhalt des originalen Dokuments ist in Anhang A.3.1 zu finden.
Die Eingabe für diese Teilpipeline ist der Text in Anhang A.3.1. Die Eingabe wird einmal von dem UMG System evaluiert und einmal von Hand klassifiziert. Danach vergleicht man die Ausgabe des UMG Systems und die von Hand klassifizierte Eingabe, die wegen ihrer Korrektheit als Maßstab dient.

Umstrukturierung

Anhand der Teilpipeline wurde der Text mit Hilfe des Stanford Parsers zuerst umstrukturiert, wobei alle Determinatoren wie z.B. „the“ , „a“ , „any“ usw. erkannt und entfernt wurden. Die Abbildung 6.3 stellt die entfernten Determinatoren in jedem Satz des Textes dar.

```
Loading parser from serialized file edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz ... done [1,3 sec].
Delete determiner: "The"
Delete determiner: "a"
Delete determiner: "a"
Delete determiner: "an"
Delete determiner: "a"
Delete determiner: "the"
Delete determiner: "a"
Delete determiner: "a"
Delete determiner: "a"
Delete determiner: "a"
Delete determiner: "an"
Delete determiner: "the"
Loading parser from serialized file edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz ... done [0,6 sec].
Delete determiner: "The"
Delete determiner: "the"
Delete determiner: "an"
Loading parser from serialized file edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz ... done [0,7 sec].
Delete determiner: "The"
Delete determiner: "the"
Delete determiner: "the"
Delete determiner: "this"
Loading parser from serialized file edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz ... done [0,6 sec].
Delete determiner: "The"
Loading parser from serialized file edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz ...Delete determiner: "a"
 done [0,7 sec].
Delete determiner: "A"
Delete determiner: "an"
Delete determiner: "a"
Delete determiner: "both"
Delete determiner: "the"
Delete determiner: "each"
```

Abbildung 6.3.: Entfernte Determinatoren des Textes

Extraktion der funktionalen Anforderungen

Mit dem Ergebnis wurde der Text basierend auf den Regeln für Extraktion der funktionalen Anforderungen in Kapitel 5.2.1 extrahiert und klassifiziert. Als Resultat des Schrittes ergibt sich die Ausgabe des Teilpipelines. Die in der Abbildung 6.4 illustrierten Sätze sind die von UMG extrahierten funktionalen Anforderungen. Der erst Satz in der Abbildung ist im technischen Sinn keine funktionale

```
1 ATM must be able to provide following services to customer:
2 customer can withdraw cash from account.
3 Bank must approve transaction before cash is dispensed.
4 customer can deposit amount.
5 customer must enter amount of deposit into ATM.
6 Bank must approve before accepting envelope.
7 customer can transfer money between accounts.
8 customer can check balance inquiry in account.
```

Abbildung 6.4.: Extrahierte funktionale Anforderungen aus dem originalen Dokument

Anforderung. Der Satz wurde dabei mit extrahiert wegen dem Schlüsselwort „must“ . Abgesehen davon wurden alle funktionale Anforderungen korrekt extrahiert und klassifiziert. Die Ausgabe der Teilpipeline ist gelungen.

6.1.2. Evaluierung des Dokuments Address Book System

Für die Evaluierung des Dokuments wurde der originale Text über „Address Book System“ eingegeben, der Text ist in Anhang A.3.2 zu sehen.

Durch die Klassifikationskomponente in der ersten Teilpipeline (Abbildung 6.2) wurde die folgenden Sätze (Abbildung 6.5) als funktionale Anforderungen extrahiert.

```
1  Users can add new person to address book.
2  Users can edit existing information about person.
3  Users can delete person.
4  Users can sort entries in address book.
5  software must sort entries alphabetically by last name or by ZIP code.
6  software must print out all entries in address book.
7  Users can create new address book.
8  Users can open disk file to close address book.
9  Users can save address book to disk file.
10 Users can use standard File menu options.
```

Abbildung 6.5.: Extrahierte funktionale Anforderungen aus dem Dokument Adress Book System

6.1.3. Evaluierung des Dokuments Online Broadcasting Services System

Für die Evaluierung des Dokuments wurde der originale Text über „Online Broadcasting Services System“ verwendet, der Text ist in Anhang A.3.3 zu sehen.
Der Text wurde erfolgreich umstrukturiert und klassifiziert. Alle funktionalen Anforderungen wurden korrekt extrahiert.

6.2. Evaluierung der Teilpipeline: Generierung des Anwendungsfalldiagramms und der Datenstruktur

In dieser Teilpipeline (siehe Abbildung 6.6) wurde die Generierung einer Datenstruktur und eines Anwendungsfalldiagramms auf dem grafischen Editor von UMG evaluiert. Die Eingabe für diese Pipeline ist das extrahiertes Dokument mit funktionalen Anforderungen, aus dem schließlich eine Datenstruktur als modellbasiertes Anforderungsdokument in UMGX Datei gespeichert wurde und ein Anwendungsdiagramm daraus generiert wurde.

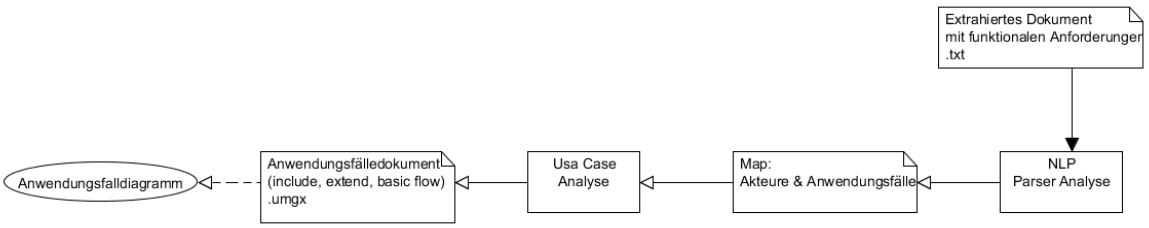

Abbildung 6.6.: Teilpipeline der Generierung der Datenstruktur und des Anwendungsfalldiagramms

Für die Evaluierung der Teilpipeline wurden die drei Dokumente ebenfalls hintereinander getestet.

6.2.1. Evaluierung des Dokuments ATM System

Extrahierung der Akteure und Anwendungsfälle

Unter Verwendung des Stanford Parsers wurden Akteure und Anwendungsfälle aus den funktionalen Anforderungen des extrahierten Dokuments analysiert. In folgender Tabelle 6.1 sind Akteure und entsprechende Anwendungsfälle zu sehen, die jeweils das Subjekt und das Prädikat jedes Satzes des extrahierten Dokuments sind.

Actor	Use Case
customer	withdraw cash from an account
bank	approve transaction
customer	deposit amount
customer	enter the amount of the deposit into the atm
bank	approve before accepting the envelope
customer	transfer money between accounts
customer	check balance inquiry in the account

Tabelle 6.1.: Extrahierte Anwendungsfälle

Von allen Sätzen wurden Akteure und Anwendungsfälle korrekt extrahiert.

Erweiterungen der Anwendungsfälle

UMG bietet Funktionalitäten an, die direkt auf dem grafischen Editor neue Anwendungsfälle und deren Akteuren hinzufügen und die Erweiterungen jedes Anwendungsfalls generieren. In der folgenden Abbildung 6.7 sind neue Anwendungsfälle zu sehen.

Einer der neuen Anwendungsfälle hat den Namen „maintain internal system“ und seinen Akteur „bank administrator“ . Der andere neue Anwendungsfall „Valid User“ besitzt keinen Akteur und wurde importiert von dem Anwendungsfall „withdraw cash from an account„ als Import-beziehung. Neben den beiden Fällen wurde ein bereits vorhandener Anwendungsfall „approve transaction“ mit dem Akteur „bank“ von dem Anwendungsfall „diposit amount“ erweitert.

Die Änderungen zu dem Diagramm auf dem grafischen Editor wurden ebenfalls in der Datenstruktur gespeichert. In Anhang A.4.1 ist die Ausgabe der Datenstruktur in UMGX Format zu sehen, die alle Elemente und Informationen in dem Anwendungsdiagramm enthält.

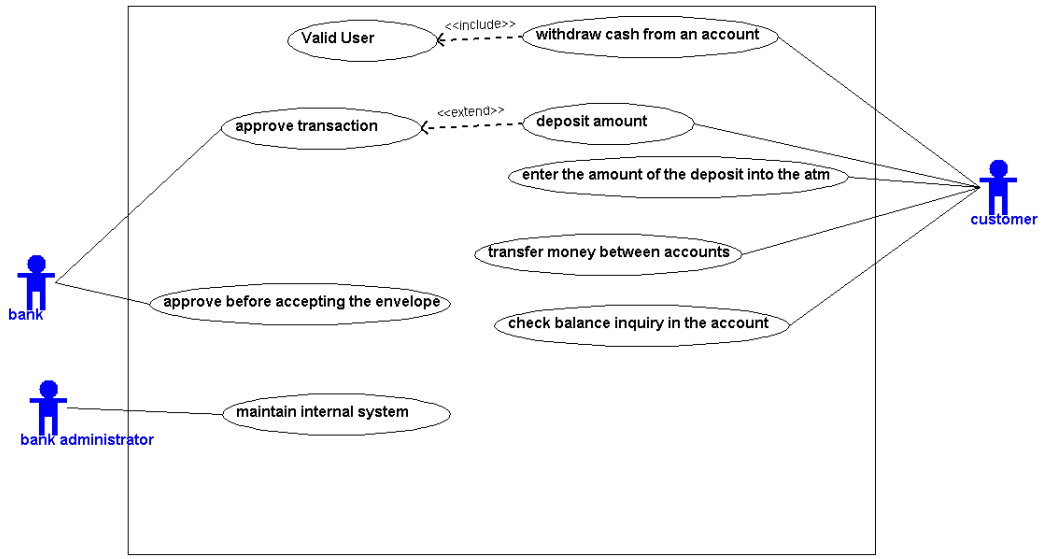

Abbildung 6.7.: Neue Anwendungsfälle und Erweiterungen

6.2.2. Evaluierung des Dokuments Address Book System

Aus den funktionalen Anforderungen wurden entsprechende Akteuren und einige gültige Anwendungsfälle generiert. Zwei Probleme tauchten jedoch bei der Extraktion der Anwendungsfälle auf. Satz 1 und Satz 8 in der Abbildung 6.5 enthalten beide ein Prädikat mit einer Konjunktion „to“ , die falsche Analyse des Stanford Parser verursacht.

Obwohl Stanford Parser ein intelligentes NLP Software ist, erkennt er Wörter nicht, die sowohl Normen als auch Verb sein können. In dem Satz 1 wurde das Wort „address“ fälschlicherweise vom Stanford Parser als ein Verb erkannt. In der Tat ist das Wort in dem Kontext ein Normen, das als Attribut zum Wort „book“ wirkt. Gewissermaßen wurde der Satz 8 wegen der Kombination „to + Verb (close)“ auch falsch analysiert. Daher wurde für Anwendungsfälle in beiden Sätzen nur das Teil an der linken Seite zu dem Wort „to“ extrahiert, die rechte Seite waren verlorengegangen.

Das zweite Problem liegt daran, dass Stanford Parser keine Verbalphrase erkennt. In dem Satz 6 wurde deswegen die Verbalphrase „print out“nicht korrekt analysiert, welches ebenfalls zu einer falschen Extraktion des Anwendungsfalls führt.

Nach Bearbeitung der drei ungültigen Sätze wurden die Anwendungsfälle korrekt extrahiert. Die folgende Abbildung 6.8 illustriert das Anwendungsfalldiagramm.
Das Anwendungsfalldiagramm wurde erfolgreich in UMGX Dokument gespeichert.

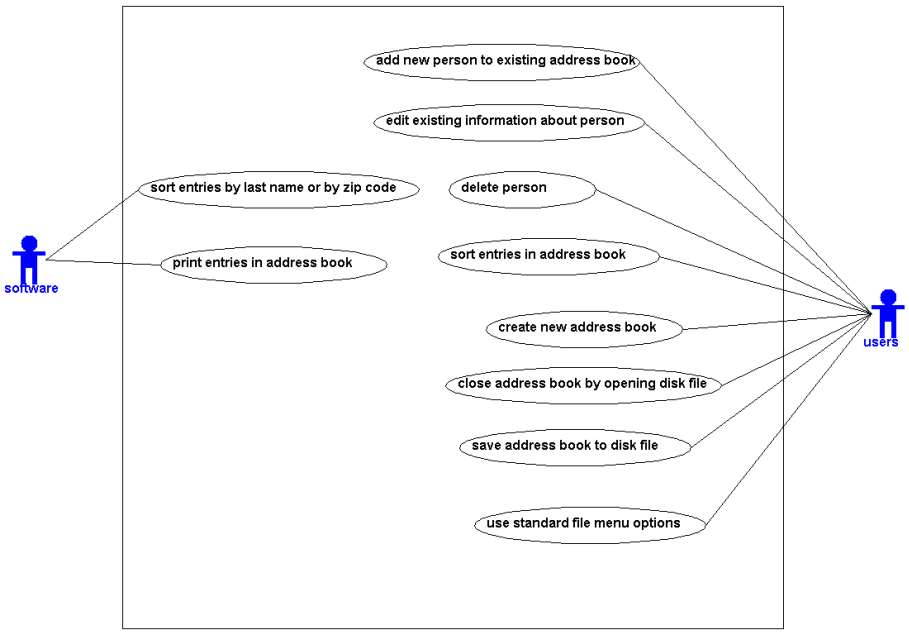

Abbildung 6.8.: Anwendungsfalldiagramm für das Dokument Adress Book System mit richtigen Anwendungsfällen

6.2.3. Evaluierung des Dokuments Online Broadcasting Services System

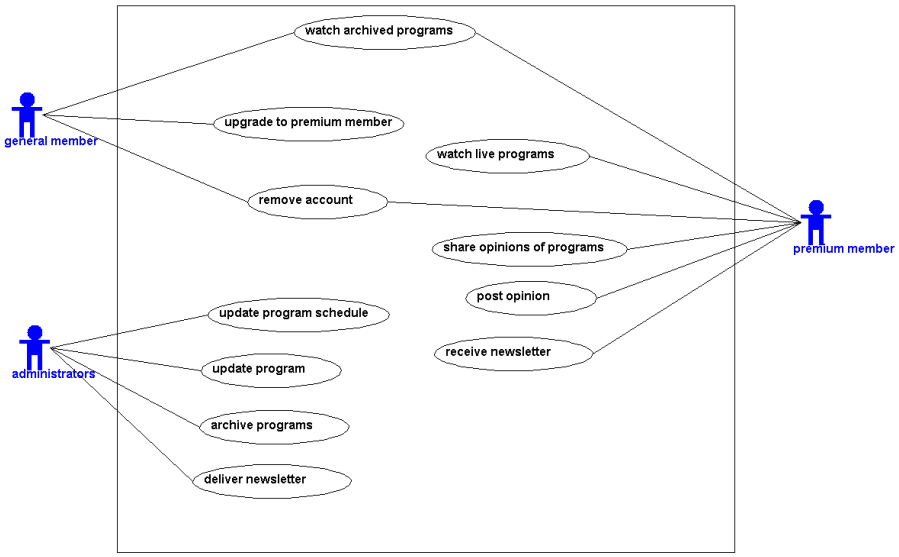

Abbildung 6.9.: Anwendungsfalldiagramm für das Dokument Online Broadcasting Services System mit richtigen Anwendungsfällen und Akteuren.

Für die extrahierten funktionalen Anforderungen wurden Anwendungsfälle und deren Akteure bei dieser Evaluierung der Teilpipeline ebenfalls erfolgreich generiert.
Keine syntaktischen Probleme tauchten bei der Generierung der Anwendungsfälle auf. Die Abbildung 6.9 illustriert das Anwendungsfalldiagramm, in dem alle Anwendungsfälle und Akteure korrekt gezeichnet wurden.
Die Speicherung des Anwendungsfalldiagramms in UMGX Dokument war erfolgreich. Dies findet man in Anhang A.4.5 als eine Ausgabe des UMGX Dokuments mit richtig extrahierten Anwendungsfällen mit Akteuren.

6.3. Evaluierung der Teilpipeline: Generierung des Sequenzdiagramm eines Anwendungsfalls

Die letzte Teilpipeline (siehe Abbildung 6.10) der Evaluierung strebt an, ein Sequenzdiagramm aus dem Standardablauf eines Anwendungsfalls zu generieren.

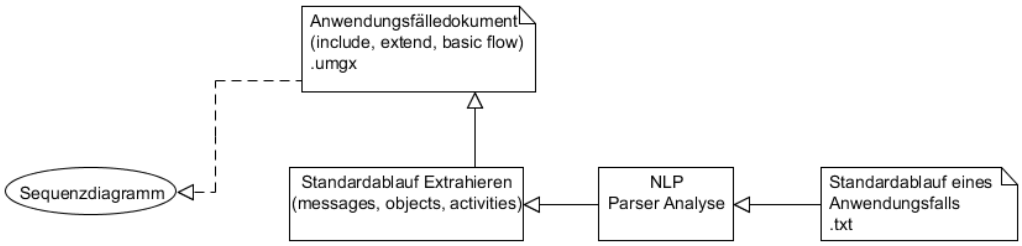

Abbildung 6.10.: Teilpipeline der Generierung des Sequenzdiagramm

6.3.1. Evaluierung des Dokuments ATM System

Die Eingabe der Pipeline ist ein Textdokument über den Standardablauf eines Anwendungsfalls. Für die Evaluierung des Dokuments Automat Teller Machine System waren die Standardabläufe von dem Anwendungsfall „withdraw cash from an account“ und von dem Anwendungsfall „deposit amount“ zum Test gekommen.

Standardablauf von Anwendungsfall Withdraw

Mit dem Text in der folgender Abbildung 6.11 über den Standardablauf wurden zuerst die Objekten, die in der Interaktion beteiligten, und die Nachrichten, die zwischen den Objekten getauscht wurden, extrahiert.

```
the customer can select the withdraw transaction on the display.
the system must get the withdraw transaction from the display.
the customer can select the amount on the display.
the system must get the amount from the display.
the system must generate the withdraw transaction information on the display.
the system must send the withdraw transaction information to the bank.
the bank must send the withdraw transaction approval to the system.
the system must dispense the cash to the cash dispenser.
the customer can get the cash from the cash dispenser.
the system must record the withdraw transaction information into log.
```

Abbildung 6.11.: Originaler Text über den Standardablauf des Anwendungsfalls „withdraw cash from an account“

Bei der Evaluierung des Sequenzdiagramms wurden die synchronen und asynchronen Nachrichten und Antworten betrachtet. Dabei wurden die Aktivierungsbalken, welche die aktive Zeit eines Objekts repräsentieren, unter Berücksichtigung genommen. Folgende Abbildung 6.12 illustriert das Sequenzdiagramm basierend auf dem Standardablauf des Anwendungsfalls.

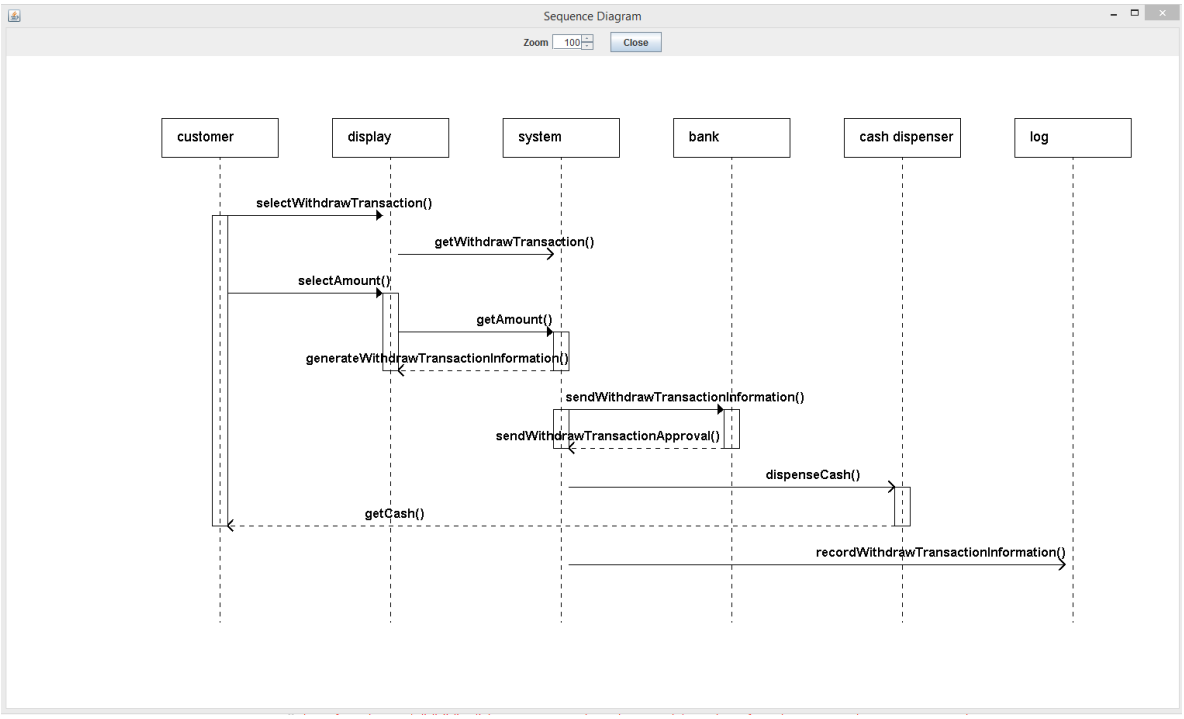

Abbildung 6.12.: Sequenzdiagramm illustriert den Standardablauf des Anwendungsfalls *withdraw cash from an account*

Anhand der Abbildung wurde das Sequenzdiagramm mit einer hohen Erfolgsrate generiert. Dabei wurden alle Objekte und Nachrichten inhaltlich korrekt generiert, außerdem wurden die drei Nachrichtentypen ebenfalls problemlos gezeichnet.
Fälschlicherweise trat ein Aktivierungsbalken für das Objekt *display* nicht in dem Diagramm auf, der sich auf der rechten Seite der ersten Nachricht befinden sollte.

Neben der Generierung des Sequenzdiagramms wurde der Standardablauf auch als ein Element des Anwendungsfalls in der Datenstruktur gespeichert. Somit ist die Vollständigkeit der Struktur eines Anwendungsfalls gewährleistet und in Anhang A.4.2 ist die Datenstruktur mit hinzugefügtem Standardablauf zu sehen.

Standardablauf von Anwendungsfall deposit amount

Mit dem Text in der folgender Abbildung 6.13 über den Standardablauf wurden wie in der obigen Evaluierung zuerst die Objekten, die in der Interaktion beteiligten, und die Nachrichten, die zwischen den Objekten getauscht wurden, extrahiert.

```
The customer can insert the card into ATM.
The ATM must valid the card on system.
The customer can get approval of the card from ATM.
The customer can enter the PIN on the ATM.
The ATM must send the PIN information to system.
The ATM must get validation of PIN from system.
the customer can get the approval of PIN from ATM.
The customer can select deposit on ATM.
The ATM must send deposit information to system.
The customer can select amount on ATM.
the ATM must send amount information to system.
the system must send deposit amount to bank.
the bank must send deposit approval to system.
the customer can put money into ATM.
the ATM must send amount of money to system.
the system must send amount of money to the bank.
the bank must get amount information from the system.
the bank must send approval information to the system.
the ATM must get approval information from the system.
the customer can get approval information of deposit from ATM.
```

Abbildung 6.13.: Originaler Text über den Standardablauf des Anwendungsfalls „deposit amount“

Das Sequenzdiagramm für den Standardablauf wurde in der folgender Abbildung 6.14 dargestellt, das sowohl Objekte als auch Nachrichten korrekt interpretiert. In dem Diagramm entstand das gleiche Problem wie bei dem Sequenzdiagramm in der Abbildung 6.12, dass die Aktivierungsbalken bei einer ausgehenden asynchronen Nachricht eines Objekts nicht optimal dargestellt wurden. Das liegt an der Verwaltung der Nachrichten von dem UMG System.

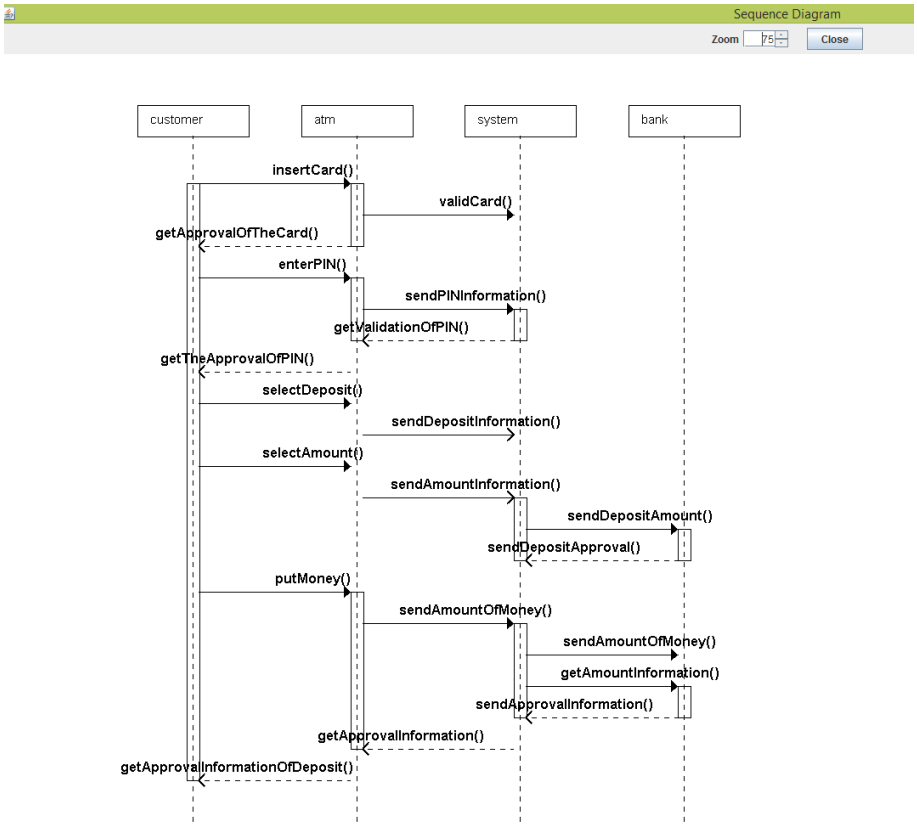

Abbildung 6.14.: Sequenzdiagramm illustriert den Standardablauf des Anwendungsfalls *deposit amount*

Der Standardablauf wurde ebenfalls erfolgreich in dem Datenmodell gespeichert. Dadurch hat das ursprüngliche Dokument *Automat Teller Machine System* ein modellbasiertes Anforderungsdokument, in dem zwei Anwendungsfälle ihre eigenen Standardablauf enthalten.

Die gesamte Ausgabe der Evaluierung der drei Teilpipelines für das Dokument ATM System erhält man in Anhang A.4.3 als eine UMGX Datei.
Dieses Dokument wurde erfolgreich evaluiert, die Ergebnisse jeder Teilpipeline erfüllen die Anforderungen der Funktionalitäten vom UMG System.

6.3.2. Evaluierung eines Anwendungsfalls aus dem Dokument Address Book System

Zur Evaluierung der Erzeugung eines Sequenzdiagramms wurde der Standardablauf des Anwendungsfalls „add new person to existing address book“ von dem Anwendungsfalldiagramm (Abbildung 6.8) mit dem Text in der Abbildung 6.15 verwendet.
Wegen der Schwäche des Stanford Parsers wurde der Text nach dem Fehlschlag bei dem ersten

Testversuch überarbeitet. Beispielsweise wurden die ungültigen Verbalphrasen durch gültige Wörter mit der gleichen Bedeutung ersetzt.

```
1  the User can select the add new person on the software.
2  the Software must send validation request for register to the user.
3  the User can enter account information into the software.
4  the Software must send the account information to the system.
5  the Software must get validation of account information from the system.
6  the User can enter the name of a new person into the software.
7  the Software must send request for name validation to the system.
8  the Software must get name validation from system.
9  the Software must send request for edition to the user.
10 the User can choose edition button on the software.
11 the Software must send edition information to the system.
```

Abbildung 6.15.: Standardablauf des Anwendungsfalls „add new person to existing adress book“

Nach der Überarbeitung des Texts wurden alle beteiligten Objekte und getauschten Nachrichten korrekt extrahiert. Somit wurde der Standardablauf in Sequenzdiagramm (Abbildung 6.16) erfolgreich transformiert.

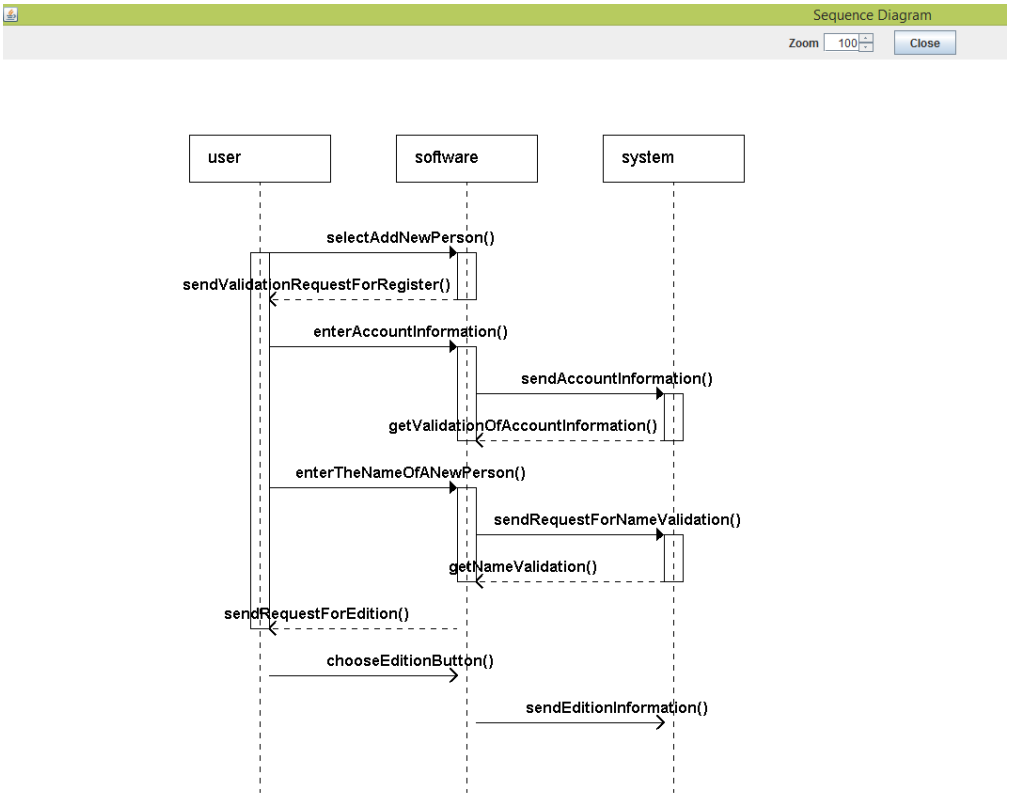

Abbildung 6.16.: Sequenzdiagramm für den Standardablauf

Wie üblich besteht das Datenmodell nach einer vollständigen Evaluierung mit drei Teilpipelines aus Anwendungsfällen und dem Standardablauf des Anwendungsfalls „add new person to existing adress book“ und es wurde erfolgreich gespeichert als ein UMGX Dokument (siehe Anhang A.4.4).

6.4. Zusammenfassung

Mit dem UML Model Generator (UMG) System können Anforderungen in textuellen Dokumenten ohne bemerkte programmatische Fehler erfolgreich in modellbasiertes Dokument umgewandelt werden und Modelle in Anwendungsdiagramm und Sequenzdiagramm aus denen generiert werden. Basierend auf dem System REC kann UMG die funktionale Anforderungen problemlos aus dem originalen Dokument extrahieren. Anhand der funktionalen Anforderungen werden Anwendungsfälle und Akteuren in meisten Fällen korrekt generiert, aus denen eine hochwertige Datenstruktur als modellbasiertes Anforderungsdokument erzeugt wird, die beliebig bearbeitet werden kann und richtige Anwendungsfalldiagramme sowie deren Sequenzdiagramme generieren kann.

Die Qualität des Dokuments kann jedoch die Präzision der Ergebnisse und die Leistung von UMG beeinflussen. Da Stanford Parser, der zur Textanalyse des Anwendungsfalls und Standardablaufs dient, Wörter bezüglich der Verbalphrase und Verben fälschlicherweise analysieren kann, müssen die ungültigen Sätze in den Dokumenten überarbeitet werden, bevor sie sich in UMG klassifizieren und extrahieren lassen.

Die Aktivitätsbalken in Sequenzdiagrammen wurden nicht immer richtig dargestellt. Es liegt an der unvollständigen Analyse der Beziehungen zwischen Objekten und deren Nachrichten, die nicht alle Fälle über synchrone und asynchrone Nachrichten abdecken. Um das Problem zu beheben muss die Textanalyse des Standardablaufs noch tiefer ins Detail gehen. Dies bietet eine Möglichkeit für die zukünftige Arbeit.

7. Zusammenfassung und Ausblick

Die Erhebung und Analyse der Anforderungen ist eine Schlüsselphase eines technischen Projekts. Schlechte oder fehlerhafte Anforderungen an komplizierte Softwareprodukte können zu einer schwerwiegenden Konsequenz führen.
Ein verbreitetes Verfahren ist die Erhebung der Anforderungen aus natürlich sprachlichen Texten. Einerseits kann das Verfahren die gute Verständlichkeit natürlicher Sprache bieten, andererseits lassen sich dabei die Mehrdeutigkeiten natürlicher Sprache nicht vermeiden. Außerdem wird in manchen Bereichen, bei denen modellbasierte Ansätze zur Anwendung kommen, die Anforderungsanalyse unter Verwendung von natürlich sprachlichen Texten durchgeführt. Dadurch ist die Verbindung zwischen Anforderungsanalyse und Entwurf nicht so gut möglich.

In der vorliegenden Arbeit wurde das System UML Model Generator (UMG) entworfen, das Modelle aus klassifizierten und modellbasierten Anforderungen generiert. Mit den Modellen können die Anforderungen nicht nur textuell sondern auch grafisch dargestellt werden. Die Qualität der Anforderungsanalyse wird dadurch verbessert und ein Übergang von der Anforderungsanalyse hin zum Entwurf wird ebenfalls ermöglicht und Mehrdeutigkeiten werden vermieden.

Das UMG System bietet die Funktionalität der Klassifizierung, die auf einem vorhandenen REC System aus der Diplomarbeit [Zwi13] basiert, welches Anforderungen aus technischen Spezifikationen extrahiert und klassifiziert. Zur Erzeugung der Modelle extrahiert UMG die funktionalen Anforderungen aus einem texutellen Dokument, welche Funktionen eines zu entwickelnden Systems beschreiben können.
Für UMG wurde ein Datenmodell entworfen, das die extrahierten Anforderungen in Format von UML Model Generator XML, abgekürzt als UMGX, darstellt. Darauf aufbauend kann ein modellbasiertes Anforderungsdokument generiert, in dem alle essentiellen Informationen zur Erzeugung der Modelle von Anforderungen beinhaltet sind. UMGX enthält Anwendungsfälle und gegebenenfalls zu diesen die Standardabläufe. Als ein grundsätzliches Konzept von UMG ermöglicht das Datenmodell beliebige Änderungen und Erweiterungen zu den Anforderungen und deren Modelle.

Neben dem Datenmodell enthält UMG einen grafischen Editor. UML Modelle von Anwendungsfalldiagramme für Anwendungsfälle aus funktionalen Anforderungen und Sequenzdiagramme für Standardabläufe werden mit Hilfe des grafischen Editors von UMG erzeugt.
Dabei bietet der grafische Editor auch diverse Funktionen. Hinzufügen neuer Anwendungsfälle ermöglicht nicht nur die Erweiterung eines vorhandenen Anwendungsfalldiagramms, sondern auch Erstellung eines neuen Anwendungsfalldiagramms. In dem grafischen Editor ist erstelltes Anwendungsfalldiagramm in Datei von UMGX Format zu speichern. Generierung eines Anwendungsfalldiagramms wird durch Import eines Anforderungsdokument oder Laden einer UMGX Datei

realisiert.
Außer der allgemeinen Verwaltung eines Anwendungsfalldiagramms ist auch einzelner Anwendungsfall eines Diagramms mithilfe von dem grafischen Editor erweiterbar. Dabei ist zu jedem Anwendungsfall die Generierung der Include-Beziehungen und Extend-Beziehungen möglich.
Ein Sequenzdiagramm wird von UMG zur Visualisierung von Standardablauf eines Anwendungsfalls verwendet, wobei der Standardablauf als ein wichtiger Bestandteil eines Anwendungsfalls in der UMGX Datei gespeichert wird und das Sequenzdiagramm nicht grafisch bearbeitbar ist.

Obwohl die Evaluierungsergebnisse für UMG positiv sind, gibt es jedoch zwei Schwachstellen. Eine Schwachstelle taucht bei der Extraktion der funktionalen Anforderungen aus einem eingehenden Anforderungsdokument auf.
Unbearbeitete Dokumente als Eingabe ist nicht optimal für die Klassifizierung von UMG. Wegen der Einschränkung von Stanford Parser müssen die Sätze in einem Dokument anhand der Regeln für eine funktionalen Anforderungen überarbeitet werden. Sätze mit einer Verbalphrase als Prädikat müssen ebenfalls überarbeitet werden, indem die Verbalphrase durch ein gleichbedeutender Verb ersetzt wird.

Die andere Schwachstelle des UMG Systems befindet sich bei der nicht optimal bezeichneten Aktivitierungsbalken eines Sequenzdiagramms.
Ein Sequenzdiagramm für den Standardablauf eines Anwendungsfalls kann sehr gut von UMG generiert werden. Jedoch werden für manche Objekte, die asynchrone Nachrichten senden oder empfangen, ihre Aktivierungsbalken nicht vollständig dargestellt. Dies liegt daran, dass UMG zwar die Beziehungen zwischen Objekten und Nachrichten analysieren kann, werden werden asynchrone Nachrichten nicht immer automatisch erkannt. Um das Problem zu beheben muss die Beziehungen zwischen Objekten und Nachrichten noch genauer analysiert werden, ob ein Objekt durch eine Nachricht aktiviert ist.

Ausblick

Da das UMG System nur prototypisch implementiert wurde, sind aus diesem Grund viele weitere Verbesserungen der Erweiterung möglich.

Die Klassifizierung und Extraktion der funktionalen Anforderungen sind bisher nur semi-automatisch realisiert. Die Einschränkungen des Stanford NLP Parsers für eine automatischen Klassifizierung können durch Verwendung des Maschinellen Lernens beseitigt werden.
[BC14] schlägt vor, dass der automatischen Klassifizierung eine Lernphase voraus geht, in der dem Klassifikationssystem umfangreiche Trainingsdaten zur Verfügung gestellt werden. Oft wird Clusteranalyse, welches Verfahren zur Entdeckung von Ähnlichkeitsstrukturen in Datenbeständen ist, als gleichwertig zur automatischen Klassifikation angesehen.
[PLW90] hat die die Klassifikation in Unterphasen geteilt mit Clusteranalyse zur Generierung einer Kategorienstruktur, Erkennung der zu einer Kategorie gehörten Objekte. Funktionale und nichtfunktionale Anforderungen können jeweils in Unterklassen unterteilt werden, dies ermöglicht das Anwenden der Clusteranalyse. Wird die Kategorienstruktur festgelegt, können die Konflikte und

Wechselbeziehungen zwischen den beiden Anforderungsklassen ebenfalls erkannt werden.

Neben der Erweiterung der Klassifikation können die Modelldarstellungen von Anwendungsfalldiagramm und Sequenzdiagramm vervollständigt werden.
Ein Anwendungsfalldiagramm enthält außer den üblichen Elementen, die von UMG generiert werden, noch Erweiterungspunkt (Extension Point), der bei einer Extend-Beziehung den Punkt angibt, an dem der erweiternde Anwendungsfall im erweiterten Anwendungsfall eingehängt wird. Obwohl die Bezeichnung des Erweiterungspunkts in der Darstellung eines Anwendungsfalldiagramms nicht zwingend ist, kann dadurch die vollständige Darstellung eines Anwendungsfalls ermöglicht werden.

Die Spezifikation eines Anwendungsfalls ist eine weitere Verbesserungsmöglichkeit für künftige Arbeit, die in Form von anderen UML-Verhaltensdiagrammen erfolgen oder als strukturierter Text hinterlegt werden kann. Der von UMG generierte Standardablauf ist ein Bestandteil von einer Anwendungsfallspezifikation. In der Spezifikation werden neben dem Standardablauf noch Namen, auslösendes Ereignis, Vorbedingungen, Verhalten im Fehlerfall, Nachbedingungen sowie Ergebnis beschrieben.
Die Darstellung des Sequenzdiagramm von UMG System enthält die grundsätzlichen Informationen. Viele weitere Bestandteile eines Sequenzdiagramms sind zu realisieren. Bei einer Rekursion für ein Objekt sendet das Objekt Nachrichten an sich selbst, dabei wird seine eigene Methode gerufen. Für die Schritte in einem Standardablauf, die alternatives Verhalten haben, spielt ein kombinierte Fragmente in dem Sequenzdiagramm eine wichtige Rolle. Dadurch kann die Spezifikation eines Anwendungsfalldiagramms korrekt in Form vom Sequenzdiagramm visualisiert werden.
Weitere Modellierungssprachen wie Aktivitätsdiagramm, die alle Abläufe, nicht nur der Standardablauf darstellen können, Klassendiagramm können ebenfalls den Zweck der modellbasierten Anforderungen erfüllen.

Anforderungsanalyse in Form von strukturierten Texten oder von UML-Modellen kann jedoch nur die CIM bei der MDA repräsentieren. Durch die Erzeugung der Modelle aus modellbasierten Anforderungen ist eine Modelltransformation auch möglich, die ein wichtiges Mittel bei der modellgetriebenen Softwareentwicklung ist. Transformationsregeln werden auf der Ebene der Metamodelle definiert. Durch die Modelltransformation kann man die generierten Modelle insgesamt in einen modellbasierten Entwicklungsansatz integrierten und sogar Code generieren.

A. Anhang

A.1. Abkürzungsverzeichnis

CIM Computation Independent Model

LIDA Linguistic assistant for Domain Analysis

MDA Model Driven Architecture

MDE Model Driven Engineering

MDRE Model Driven Requirements Engineering

MOF Meta Object Facility

NLP Natural Language Processing

OOAD Object Oriented Analysis and Design

PCFG Probabilistic Context Free Grammars

PIM Platform Independent Model

POS Part-of-Speech

PSM Platform Specific Model

REC Requirement Extractor and Classifier

SVO Subjekt-Objekt-Verb

RUP Rational Unified Process

SUGAR Static UML Model Generator from Analysis of Requirements

SVG Scalable Vector Graphics

UCDA Use Case Driven Development Assistant Tool

UMG UML Model Generator

UMGX UML Model Generator XML

UML Unified Modeling Language

XSD XML Schema Definition

XSLT Extensible Stylesheet Language Transformation

A.2. Tabellen

A.2.1. Tabelle für Stanford Typed Dependencies

Stanford typed dependencies	Definitionen	Beispiele
acomp: adjectival complement	An adjectival complement of a verb is an adjectival phrase which functions as the complement.	“She looks very beautiful” acomp(looks, beautiful)
amod: adjectival modifier	An adjectival modifier of an NP is any adjectival phrase that serves to modify the meaning of the NP.	“Sam eats red meat” amod(meat, red)
aux: auxiliary	An auxiliary of a clause is a non-main verb of the clause	a modal auxiliary, or a form of “be”, “do” or “have” in a periphrastic tense.
conj: conjunct	A conjunct is the relation between two elements connected by a coordinating conjunction, such as “and”, “or”, etc.	“Bill is big and honest” conj(big, honest)
det: determiner	A determiner is the relation between the head of an NP and its determiner.	“The man is here” det(man, the)
dobj: direct object	The direct object of a VP is the noun phrase which is the (accusative) object of the verb.	They win the lottery” dobj(win, lottery)
iobj: indirect object	The indirect object of a VP is the noun phrase which is the (dative) object of the verb.	“She gave me a raise” iobj(gave, me)
nn: noun compound modifier	A noun comp ound mo difier of an NP is any noun that serves to mo dify the head noun.	“cash card” nn(card, cash)

nsubj: nominal subject	A nominal subject is a noun phrase which is the syntactic subject of a clause.	“Clinton defeated Dole” nsubj(defeated, Clinton)
ref: referent	A referent of the head of an NP is the relative word introducing the relative clause modifying the NP.	“I saw the book which you bought” ref(book, which)
xsubj: controlling subject	A controlling sub ject is the relation between the head of a open clausal complement and the external subject of that clause.	“Tom likes to eat fish” xsubj(eat, Tom)

Tabelle A.1.: Die am häufigsten vorgekommenen Abhängigkeiten von Stanford Typed Dependencies [MM08]

A.3. Originale Texte

A.3.1. Dokument Automat Teller Machine System

The software to be designed will control a simulated automated teller machine (ATM) having a magnetic stripe reader for reading an ATM card, a keyboard and display for interaction with the customer, a slot for depositing envelopes, a dispenser for cash , a printer for printing customer receipts, and a key-operated switch to allow an operator to start or stop the machine. The ATM will communicate with the bank’s computer over an appropriate communication link.

The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a personal identification number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions. The card will be retained in the machine until the customer indicates that he/she desires no further transactions, at which point it will be returned - except as noted below.

The ATM provides the following services to the customer. A customer can withdraw cash from an account. Bank must approve transaction before cash is dispensed. A customer can deposit amount. The customer must enter the amount of the deposit into the ATM. Bank must approve before accepting the envelope. A customer can transfer money between accounts. A customer can check balance inquiry in the account.

The ATM will communicate each transaction to the bank and obtain verification that it was allowed by the bank. In the case of a cash withdraw or deposit, a second message will be sent after the transaction has been physically completed (cash dispensed or envelope accepted).

If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the PIN before a transaction can proceed. If the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to contact the bank to get it back.

If a transaction fails for any reason other than an invalid PIN, the ATM will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction.

The ATM will provide the customer with a printed receipt for each successful transaction, showing the date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account („to" account for transfers).

The ATM will have a an operator panel with a key-operated switch (located on the „inside the bank" side) that will allow an operator to start and stop the servicing of customers. The machine will shut down, when the switch is moved to the „off" position, so that the operator may remove deposit envelopes and reload the machine with cash, blank receipts, etc. The operator will be required to verify and enter the total cash on hand before starting the system from this panel.

A.3.2. Dokument Address Book System

The software to be designed is a program that can be used to maintain an address book. An address book holds a collection of entries, each recording a person's first and last names, address, city, state, zip, and phone number.

Users can add a new person to a existing address book. Users can edit existing information about a person. Users can delete a person. Users can sort the entries in the address book. The software must sort the entries alphabetically by last name or by ZIP code. The software must print all the entries in the address book. The entries are in „mailing label" format.

Users can create a new address book. Users can close an address book by opening a disk file. Users can save an address book to a disk file. Users can use standard File menu options. The program's File menu will also have a Quit option to allow closing all open address books and terminating the program.

The initial requirements call for the program to only be able to work with a single address book at a time; therefore, if the user chooses the New or Open menu option, any current address book will be closed before creating/opening a new one. A later extension might allow for multiple address books to be open, each with its own window which can be closed separately, with closing the last open window resulting in terminating the program. In this case, New and Open will result in creating a new window, without affecting the current window.

The program will keep track of whether any changes have been made to an address book since it was last saved, and will offer the user the opportunity to save changes when an address book is closed either explicitly or as a result of choosing to create/open another or to quit the program.

The program will keep track of the file that the current address book was read from or most recently saved to, will display the file's name as the title of the main window, and will use that file when executing the Save option. When a New address book is initially created, its window will be titled

„Untitled“, and a Save operation will be converted to Save As - the user will be required to specify a file.

A.3.3. Dokument Online Broadcasting Services System

OTV (Online Television) is a company which delivers both paid and free online television broadcasting services to all TV fans. Members are allowed to watch both live and archived TV programs on OTV's website, anytime and anywhere. There are two kinds of memberships - general and premium. It is free of charge for visitors to register as a general member, On the other hand, they can register as a premium member for US 30 per month. general member can watch archived programs. premium member can watch archived programs. premium member can watch live programs. A general member can upgrade to a premium member. general member can remove account. premium member can remove account. premium member can share opinions of the programs. premium member can post opinion. Premium member can receive newsletter. administrators must update the program schedule. administrators must update the program. administrators must archive programs. Administrators must deliver newsletter.

A.4. Beispiele der UMGX Format

A.4.1. ATM System UMGX Ausgabe mit Anwendungsfällen

```
1  <?xml version="1.0" encoding="UTF−8" standalone="no"?>
2  <usecases>
3      <usecase name="withdraw cash from an account">
4          <actor name="customer"/>
5          <include name="Valid User"/>
6      </usecase>
7      <usecase name="approve transaction">
8          <actor name="bank"/>
9      </usecase>
10     <usecase name="deposit amount">
11         <actor name="customer"/>
12         <extend name="approve transaction"/>
13     </usecase>
14     <usecase name="enter the amount of the deposit into the atm">
15         <actor name="customer"/>
16     </usecase>
17     <usecase name="approve before accepting the envelope">
18         <actor name="bank"/>
19     </usecase>
20     <usecase name="transfer money between accounts">
21         <actor name="customer"/>
22     </usecase>
23     <usecase name="check balance inquiry in the account">
24         <actor name="customer"/>
25     </usecase>
26     <usecase name="Valid User"/>
27     <usecase name="maintain internal system">
```

```
28          <actor name="bank administrator"/>
29      </usecase>
30  </usecases>
```

Listing A.1: UMGX Ausgabe mit Anwendungsfällen für Dokument ATM System

A.4.2. ATM System UMGX Ausgabe mit Standardablauf des Anwendungsfall Withdraw

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <usecases>
3      <usecase name="withdraw cash from an account">
4          <actor name="customer"/>
5          <include name="Valid User"/>
6          <basicflow>
7              <objects>
8                  <object index="0" name="customer">
9                      <activity endtime="8" starttime="0"/>
10                 </object>
11                 <object index="1" name="display">
12                     <activity endtime="4" starttime="2"/>
13                 </object>
14                 <object index="2" name="system">
15                     <activity endtime="4" starttime="3"/>
16                     <activity endtime="6" starttime="5"/>
17                 </object>
18                 <object index="3" name="bank">
19                     <activity endtime="6" starttime="5"/>
20                 </object>
21                 <object index="4" name="cash dispenser">
22                     <activity endtime="8" starttime="7"/>
23                 </object>
24                 <object index="5" name="log"/>
25             </objects>
26             <synccallmessages>
27                 <message isreturn="false" message="select withdraw transaction" object1="customer"
                       object2="display" time="0"/>
28                 <message isreturn="false" message="select amount" object1="customer" object2="display"
                       time="2"/>
29                 <message isreturn="false" message="get amount" object1="display" object2="system"
                       time="3"/>
30                 <message isreturn="false" message="send withdraw transaction information"
                       object1="system" object2="bank" time="5"/>
31             </synccallmessages>
32             <asynccallmessages>
33                 <message isreturn="false" message="get withdraw transaction" object1="display"
                       object2="system" time="1"/>
34                 <message isreturn="false" message="dispense cash" object1="system" object2="cash
                       dispenser" time="7"/>
35                 <message isreturn="false" message="record withdraw transaction information"
                       object1="system" object2="log" time="9"/>
36             </asynccallmessages>
37             <returnmessages>
```

```
38                  <message isreturn="true" message="generate withdraw transaction information"
                        object1="display" object2="system" time="4"/>
39                  <message isreturn="true" message="send withdraw transaction approval" object1="system"
                        object2="bank" time="6"/>
40                  <message isreturn="true" message="get cash" object1="customer" object2="cash dispenser"
                        time="8"/>
41              </returnmessages>
42          </basicflow>
43      </usecase>
44      <usecase name="approve transaction">
45          <actor name="bank"/>
46      </usecase>
47      <usecase name="deposit amount">
48          <actor name="customer"/>
49          <extend name="approve transaction"/>
50      </usecase>
51      <usecase name="enter the amount of the deposit into the atm">
52          <actor name="customer"/>
53      </usecase>
54      <usecase name="approve before accepting the envelope">
55          <actor name="bank"/>
56      </usecase>
57      <usecase name="transfer money between accounts">
58          <actor name="customer"/>
59      </usecase>
60      <usecase name="check balance inquiry in the account">
61          <actor name="customer"/>
62      </usecase>
63      <usecase name="Valid User"/>
64      <usecase name="maintain internal system">
65          <actor name="bank administrator"/>
66      </usecase>
67  </usecases>
```

Listing A.2: UMGX Ausgabe mit Standardablauf des Anwendungsfall Withdraw für Dokument ATM System

A.4.3. Gesamte UMGX Ausgabe für Dokument ATM System

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <usecases>
3      <usecase name="withdraw cash from an account">
4          <actor name="customer"/>
5          <include name="Valid User"/>
6          <basicflow>
7              <objects>
8                  <object index="0" name="customer">
9                      <activity endtime="8" starttime="0"/>
10                 </object>
11                 <object index="1" name="display">
12                     <activity endtime="4" starttime="2"/>
13                 </object>
14                 <object index="2" name="system">
```

```
15                      < activity  endtime="4"  starttime ="3"/>
16                      < activity  endtime="6"  starttime ="5"/>
17                  </ object >
18                  <object  index="3"  name="bank">
19                      < activity  endtime="6"  starttime ="5"/>
20                  </ object >
21                  <object  index="4"  name="cash dispenser">
22                      < activity  endtime="8"  starttime ="7"/>
23                  </ object >
24                  <object  index="5"  name="log"/>
25              </ objects >
26              <synccallmessages>
27                  <message isreturn=" false "  message="select  withdraw transaction "  object1 ="customer"
                        object2 =" display "  time="0"/>
28                  <message isreturn=" false "  message="select  amount" object1="customer"  object2 =" display "
                        time="2"/>
29                  <message isreturn=" false "  message="get amount" object1=" display "  object2 ="system"
                        time="3"/>
30                  <message isreturn=" false "  message="send withdraw transaction  information"
                        object1 ="system"  object2 ="bank"  time="5"/>
31              </ synccallmessages >
32              <asynccallmessages>
33                  <message isreturn=" false "  message="get withdraw transaction "  object1 =" display "
                        object2 ="system"  time="1"/>
34                  <message isreturn=" false "  message="dispense cash"  object1 ="system"  object2 ="cash
                        dispenser "  time="7"/>
35                  <message isreturn=" false "  message="record withdraw transaction  information"
                        object1 ="system"  object2 ="log "  time="9"/>
36              </ asynccallmessages >
37              <returnmessages>
38                  <message isreturn="true "  message="generate withdraw transaction  information"
                        object1 =" display "  object2 ="system"  time="4"/>
39                  <message isreturn="true "  message="send withdraw transaction  approval"  object1 ="system"
                        object2 ="bank"  time="6"/>
40                  <message isreturn="true "  message="get cash"  object1 ="customer"  object2 ="cash  dispenser "
                        time="8"/>
41              </returnmessages>
42          </ basicflow >
43      </usecase>
44      <usecase  name="approve transaction">
45          <actor  name="bank"/>
46      </usecase>
47      <usecase  name="deposit amount">
48          <actor  name="customer"/>
49          <extend name="approve transaction"/>
50          <basicflow >
51              < objects >
52                  <object  index="0"  name="customer">
53                      < activity  endtime="19"  starttime ="0"/>
54                  </ object >
55                  <object  index="1"  name="atm">
56                      < activity  endtime="2"  starttime ="0"/>
57                      < activity  endtime="5"  starttime ="3"/>
58                      < activity  endtime="18"  starttime ="13"/>
59                  </ object >
```

```
60              <object index="2" name="system">
61                  <activity endtime="5" starttime="4"/>
62                  <activity endtime="12" starttime="10"/>
63                  <activity endtime="17" starttime="14"/>
64              </object>
65              <object index="3" name="bank">
66                  <activity endtime="12" starttime="11"/>
67                  <activity endtime="17" starttime="16"/>
68              </object>
69          </objects>
70          <synccallmessages>
71              <message isreturn="false" message="insert card" object1="customer" object2="atm"
                    time="0"/>
72              <message isreturn="false" message="valid card" object1="atm" object2="system" time="1"/>
73              <message isreturn="false" message="enter PIN" object1="customer" object2="atm"
                    time="3"/>
74              <message isreturn="false" message="send PIN information" object1="atm" object2="system"
                    time="4"/>
75              <message isreturn="false" message="select deposit" object1="customer" object2="atm"
                    time="7"/>
76              <message isreturn="false" message="select amount" object1="customer" object2="atm"
                    time="9"/>
77              <message isreturn="false" message="send deposit amount" object1="system" object2="bank"
                    time="11"/>
78              <message isreturn="false" message="put money" object1="customer" object2="atm"
                    time="13"/>
79              <message isreturn="false" message="send amount of money" object1="atm" object2="system"
                    time="14"/>
80              <message isreturn="false" message="send amount of money" object1="system"
                    object2="bank" time="15"/>
81              <message isreturn="false" message="get amount information" object1="system"
                    object2="bank" time="16"/>
82          </synccallmessages>
83          <asynccallmessages>
84              <message isreturn="false" message="send deposit information" object1="atm"
                    object2="system" time="8"/>
85              <message isreturn="false" message="send amount information" object1="atm"
                    object2="system" time="10"/>
86          </asynccallmessages>
87          <returnmessages>
88              <message isreturn="true" message="get approval of the card" object1="customer"
                    object2="atm" time="2"/>
89              <message isreturn="true" message="get validation of PIN" object1="atm" object2="system"
                    time="5"/>
90              <message isreturn="true" message="get the approval of PIN" object1="customer"
                    object2="atm" time="6"/>
91              <message isreturn="true" message="send deposit approval" object1="system"
                    object2="bank" time="12"/>
92              <message isreturn="true" message="send approval information" object1="system"
                    object2="bank" time="17"/>
93              <message isreturn="true" message="get approval information" object1="atm"
                    object2="system" time="18"/>
94              <message isreturn="true" message="get approval information of deposit"
                    object1="customer" object2="atm" time="19"/>
95          </returnmessages>
```

```
96          </basicflow>
97      </usecase>
98      <usecase name="enter the amount of the deposit into the atm">
99          <actor name="customer"/>
100     </usecase>
101     <usecase name="approve before accepting the envelope">
102         <actor name="bank"/>
103     </usecase>
104     <usecase name="transfer money between accounts">
105         <actor name="customer"/>
106     </usecase>
107     <usecase name="check balance inquiry in the account">
108         <actor name="customer"/>
109     </usecase>
110     <usecase name="Valid User"/>
111     <usecase name="maintain internal system">
112         <actor name="bank administrator"/>
113     </usecase>
114 </usecases>
```

Listing A.3: Gesamte UMGX Ausgabe für Dokument ATM System

A.4.4. Gesamte UMGX Ausgabe für Dokument Address Book System

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <usecases>
3      <usecase name="add new person to existing address book">
4          <actor name="users"/>
5          <basicflow>
6              <objects>
7                  <object index="0" name="user">
8                      <activity endtime="8" starttime="0"/>
9                  </object>
10                 <object index="1" name="software">
11                     <activity endtime="1" starttime="0"/>
12                     <activity endtime="4" starttime="2"/>
13                     <activity endtime="7" starttime="5"/>
14                 </object>
15                 <object index="2" name="system">
16                     <activity endtime="4" starttime="3"/>
17                     <activity endtime="7" starttime="6"/>
18                 </object>
19             </objects>
20             <synccallmessages>
21                 <message isreturn="false" message="select add new person" object1="user"
                       object2="software" time="0"/>
22                 <message isreturn="false" message="enter account information" object1="user"
                       object2="software" time="2"/>
23                 <message isreturn="false" message="send account information" object1="software"
                       object2="system" time="3"/>
24                 <message isreturn="false" message="enter the name of a new person" object1="user"
                       object2="software" time="5"/>
25                 <message isreturn="false" message="send request for name validation" object1="software"
                       object2="system" time="6"/>
```

```
26              </synccallmessages>
27              <asynccallmessages>
28                  <message isreturn=" false " message="choose edition button" object1="user"
                        object2="software" time="9"/>
29                  <message isreturn=" false " message="send edition information" object1="software"
                        object2="system" time="10"/>
30              </asynccallmessages>
31              <returnmessages>
32                  <message isreturn="true" message="send validation request for register " object1="user"
                        object2="software" time="1"/>
33                  <message isreturn="true" message="get validation of account information"
                        object1="software" object2="system" time="4"/>
34                  <message isreturn="true" message="get name validation" object1="software"
                        object2="system" time="7"/>
35                  <message isreturn="true" message="send request for edition " object1="user"
                        object2="software" time="8"/>
36              </returnmessages>
37          </basicflow>
38      </usecase>
39      <usecase name="edit existing information about person">
40          <actor name="users"/>
41      </usecase>
42      <usecase name="delete person">
43          <actor name="users"/>
44      </usecase>
45      <usecase name="sort entries in address book">
46          <actor name="users"/>
47      </usecase>
48      <usecase name="sort entries by last name or by zip code">
49          <actor name="software"/>
50      </usecase>
51      <usecase name="print entries in address book">
52          <actor name="software"/>
53      </usecase>
54      <usecase name="create new address book">
55          <actor name="users"/>
56      </usecase>
57      <usecase name="close address book by opening disk file ">
58          <actor name="users"/>
59      </usecase>
60      <usecase name="save address book to disk file ">
61          <actor name="users"/>
62      </usecase>
63      <usecase name="use standard file menu options">
64          <actor name="users"/>
65      </usecase>
66  </usecases>
```

Listing A.4: Gesamte UMGX Ausgabe für Dokument Address Book System

A.4.5. UMGX Ausgabe für Dokument Online Broadcasting Services System

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <usecases>
```

```
3       <usecase name="watch archived programs">
4           <actor name="general member"/>
5           <actor name="premium member"/>
6       </usecase>
7       <usecase name="watch live programs">
8           <actor name="premium member"/>
9       </usecase>
10      <usecase name="upgrade to premium member">
11          <actor name="general member"/>
12      </usecase>
13      <usecase name="remove account">
14          <actor name="general member"/>
15          <actor name="premium member"/>
16      </usecase>
17      <usecase name="share opinions of programs">
18          <actor name="premium member"/>
19      </usecase>
20      <usecase name="post opinion">
21          <actor name="premium member"/>
22      </usecase>
23      <usecase name="receive newsletter ">
24          <actor name="premium member"/>
25      </usecase>
26      <usecase name="update program schedule">
27          <actor name="administrators"/>
28      </usecase>
29      <usecase name="update program">
30          <actor name="administrators"/>
31      </usecase>
32      <usecase name="archive programs">
33          <actor name="administrators"/>
34      </usecase>
35      <usecase name="deliver newsletter ">
36          <actor name="administrators"/>
37      </usecase>
38  </usecases>
```

Listing A.5: UMGX Ausgabe für Dokument Online Broadcasting Services System

Literaturverzeichnis

[MSB14] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, D. McClosky. *The Stanford CoreNLP Natural Language Processing Toolkit.* Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, S. 55–60. 2014. URL http://nlp.stanford.edu/software/corenlp.shtml. (Zitiert auf den Seiten 8 und 31)

[KS08] D. D. Kumar, R. Sanyal. *Static UML Model Generator from Analysis of Requirements (SUGAR).* 2008 Advanced Software Engineering Its Applications, 2008. (Zitiert auf den Seiten 27, 29 und 30)

[Ebe12] C. Ebert. *Systematisches Requirements Engineering.* dpunkt.verlag, 2012. (Zitiert auf den Seiten 9, 17 und 18)

[Sta09] Standish Group. *Chaos Reports und diverse Pressmeldungen.* 2009. URL http://www.standishgroup.com/newsroom/chaos_2009.php. (Zitiert auf den Seiten 8 und 12)

[SLF04] K. Subramaniam, D. Liu, B. H. Far, A. Eberlein. *UCDA: Use Case Driven Development Assistant Tool for Class Model Generation.* Proceedings of the Sixteenth International Conference on Software Engineering Knowledge Engineering (SEKE'2004), 2004. (Zitiert auf den Seiten 8, 27, 28 und 29)

[MM08] M. d. Marneffe, C. D. Manning. *Stanford typed dependencies manual.* 2008. URL http://nlp.stanford.edu/software/dependencies_manual.pdf. (Zitiert auf den Seiten 9, 32 und 79)

[Pat10] H. Patsch. *Requirements-Engineering systematisch – Modellbildung für software-gestützte Systeme.* 2. Auflage, Springer, 2010. (Zitiert auf den Seiten 8, 20, 21, 22, 35, 36 und 37)

[Poh08] K. Pohl. *Requirements Engineering-Grundlagen, Prinzipien, Techniken.* dpunkt.verlag, 2008. (Zitiert auf den Seiten 8, 11, 15, 16, 17, 19, 20 und 21)

[San91] B. Santorini. *Part-of-Speech Tagging Guidelines for the Penn Treebank Project.* 1991. (Zitiert auf Seite 32)

[OLR01] S. Overmyer, B. Lavoie, O. Rambow. *Conceptual Modeling through Linguistic Analysis Using LIDA.* In Proceedings of 23rd International Conference on Software Engineering (ICSE 2001), 2001. (Zitiert auf Seite 29)

[Zwi13] F. Zwirn. *Analyse und Auswertung von gewichteten Anforderungen in technischen Spezifikationen.* 2013. (Zitiert auf den Seiten 3, 5, 8, 13, 18, 24, 27, 28, 30, 33, 34, 35, 41 und 73)

[DT03] D. Dranidis, K. Tigka. *Writing Use Cases in XML*. 9th Panhellenic Conference in Informatics, 2003. URL http://delab.csd.auth.gr/bci1/Panhellenic/600dranidis.pdf.

[Atw90] E. Atwell. *The University of Pennsylvania (Penn) Treebank Tag-set*. University of Pennsylvania, 1990. URL http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html. (Zitiert auf Seite 48)

[RDF07] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, D. Mladenić. *Triplet Extraction from Sentences*. Conference on Data Mining and Data Warehouses (SiKDD 2007), 2007. URL http://ailab.ijs.si/dunja/SiKDD2007/Papers/Rusu_Trippels.pdf. (Zitiert auf Seite 49)

[Som11] I. Sommerville. *Software Engineering*. 9th edition, Harlow et al.: Ad-dison-Wesley, 2011. (Zitiert auf Seite 16)

[LL13] J. Ludwig, H. Lichter. *Software Engineering*. 3. Auflage, dpunkt Verlag, 2013.

[Sch14] L. Schwärzlier. *Modellbasierte Anforderungsanalyse*. MID-The Modeling Company, 2014. URL http://www.mid.de/fileadmin/pdf/Solutions/Requirements_Engineering_and_Management/Modellbasierte_Anforderungsanalyse.pdf. (Zitiert auf Seite 19)

[OMG] Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure. Versi-on 2.4.1*. Retrieved 9 April 2014. (Zitiert auf den Seiten 8 und 23)

[RR06] S. Robertson, J. Robertson. *Mastering the Requirements Process*. 2. Auflage. Addison-Wesley, Harlow 2006. (Zitiert auf Seite 17)

[RG01] C. Rupp, Sophist Group. *Requirements Engineering und -Management*. Hanser, München 2001. (Zitiert auf Seite 17)

[Ang14] G. Angermeier. *Randbedingungen*. Projektmanagement-Glossar, Projekt Magazin 2014. URL https://www.projektmagazin.de/glossarterm/randbedingungen. (Zitiert auf den Seiten 9 und 18)

[IT14] IT Wissen. *MOF (meta object facility)*. Das große Online-Lexikon für Informationstechnologie 2014. URL http://www.itwissen.info/definition/lexikon/MOF-meta-object-facility.html. (Zitiert auf den Seiten 8 und 21)

[Bjo05] R. C. Bjork. *Requirements and User Interface for a Simple Address Book*. 2005. URL http://www.math-cs.gordon.edu/courses/cs211/AddressBookExample/index.html. (Zitiert auf Seite 59)

[Bjo04] R. C. Bjork. *Requirements Statement for Example ATM System*. 2004. URL http://www.math-cs.gordon.edu/courses/cps122/ATMExample/index.html. (Zitiert auf Seite 59)

[VP11] Visual-Paradigm. *Online Broadcasting Services System*. 2011. URL http://d1dlalugb0z2hd.cloudfront.net/vpuml/tutorials/textualanalysis_screenshots/resources/OTV.txt. (Zitiert auf Seite 59)

[Gui14] M. Guist. *Sequenzdiagramm*. 2014. URL https://www.fbi.h-da.de/labore/case/uml/sequenzdiagramm.html. (Zitiert auf Seite 56)

[BC14] Bayard Consulting. *Die Herausforderung - Vollständige und Systemunabhängig Klassifizierte Stammdaten*. 2014. URL http://www.bayard-consulting.com/index.php/de/dienstleistungen/automatische-klassifikation. (Zitiert auf Seite 74)

[PLW90] J. D. Palmer, Y. Liang, L. Wang. *Classification as an Approach To Requirements Analysis*. Proceedings of the 1st ASIS SIG/CR Classification Research Workshop 1990. URL http://journals.lib.washington.edu/index.php/acro/article/viewFile/12472/11011. (Zitiert auf Seite 74)

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift