Institut für Rechnergestützte Ingenieursysteme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3382

Analyse und Auswertung von gewichteten Anforderungen in technischen Spezifikationen

Felix Zwirn

Studiengang:	Informatik
Prüfer:	Univ-Prof. Hon-Prof. Dr. Dieter Roller
Betreuer:	Dipl.-Inf. Akram Chamakh
begonnen am:	13. September 2012
beendet am:	15. März 2013
CR-Klassifikation:	D.2.1, H.1.2, I.2.7, I.7.m

Abstract

Requirement elicitation from natural language texts is an essential part of every technical project. However the extraction of requirements by hand is time-consuming and prone to error. Therefore, for this study, a system, to analyze and evaluate weighted requirements, was designed. The Requirement Extractor and Classifier (REC) system examines sentences whether they constitute to be requirements. An ontology divides them into classes. The considered classes are requirements, weak requirements, optional requirements, requirement candidates and no requirement. The ontology utilizes (subject, verb, object) triples to find candidates. Based on certain keywords candidates are then further distinguished.

The elicitation ontology has a hit rate of over 80%, at an accuracy of 100% in technical specifications. Thus, no sentences are incorrectly classified as requirements. Inspection of other texts show, that many optional requirements were indeed no requirements at all. This led to a precision of 50% in documents that aren't technical specifications. The classification quality highly depends on the structure of the incoming document.

The REC can also be used to classify documents with other ontologies. The ontologies must extend a base ontology (Figure 5.6). One document can be classified without interaction, by multiple ontologies.

Kurzfassung

Die Erhebung von Anforderungen aus natürlich sprachlichen Texten ist ein wichtiger Aufgabenteil eines jeden technischen Projektes. Die Extraktion von Anforderungen von Hand ist allerdings langwierig und anfällig für Fehler. Für diese Arbeit wurde deswegen ein System entworfen, dass gewichtete Anforderungen analysiert und auswertet. Es werden vom REC System Sätze danach untersucht, ob es sich bei ihnen um Anforderungen handelt. Dafür werden sie mithilfe einer Ontologie in Klassen unterteilt. Die betrachteten Klassen sind Anforderungen, schwache Anforderungen, optionale Anforderungen, Kandidaten für Anforderungen und keine Anforderungen. Die Ontologie nutzt (Subjekt, Verb, Objekt)-Tripel (SVO-Tripel), um Kandidaten zu finden. Anhand bestimmter Schlüsselwörter werden die Kandidaten danach weiter unterschieden.

Die Erkennungsontologie hat bei technischen Spezifikationen eine Trefferquote von über 80% bei einer Präzision von 100%. Es werden also keine Sätze fälschlicherweise als Anforderungen klassifiziert. Betrachtungen von anderen Texten ergaben, dass viele optionale Anforderungen tatsächliche keine Anforderungen waren. Dies führte zu einer Präzision von 50% bei Dokumenten, die keine technische Spezifikation sind. Die Qualität der Klassifizierung hängt somit stark von der Struktur des eingehenden Dokumentes ab.

Der REC kann auch genutzt werden, um Dokumente mit anderen Ontologien zu klassifizieren. Die Ontologien müssen eine Basisontologie (Abbildung 5.6) erweitern. Ein Dokument lässt sich ohne Wechselwirkung von mehreren Ontologien klassifizieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

Die steigende Komplexität moderner Systeme stellt die entwickelden Industriekonzerne vor neue Herausforderungen. Um diese Herausforderungen zu bewältigen, wurden Lösungen wie MDA konzeptionalisiert. Die Komplexität soll dabei durch Einführung weiterer Abstraktionsebenen und eine teilweise Automatisierung, bewältigt werden.

1.1. Ausgangslage

Eine der ersten und wichtigsten Phasen der Software Entwicklung ist die Anforderungserhebung. In einem Kundengespräch hält ein Analyst die Anforderungen zusammen mit dem Kunden in natürlicher Sprache fest. Aus diesen Anforderungen erstellt der Analyst die technische Spezifikation. Der nächste Schritt der Entwicklung besteht nun darin, aus der Spezifikation Modell zu erzeugen. Dabei fallen Unklarheiten (siehe Abbildung 1.1) und unvollständig spezifizierte Anforderungen auf. Der Analyst muss diese nun in einem weiteren Kundengespräch klären. Dieser Ablauf kann sich mehrmals wiederholen, was Zeit und somit Geld kostet.

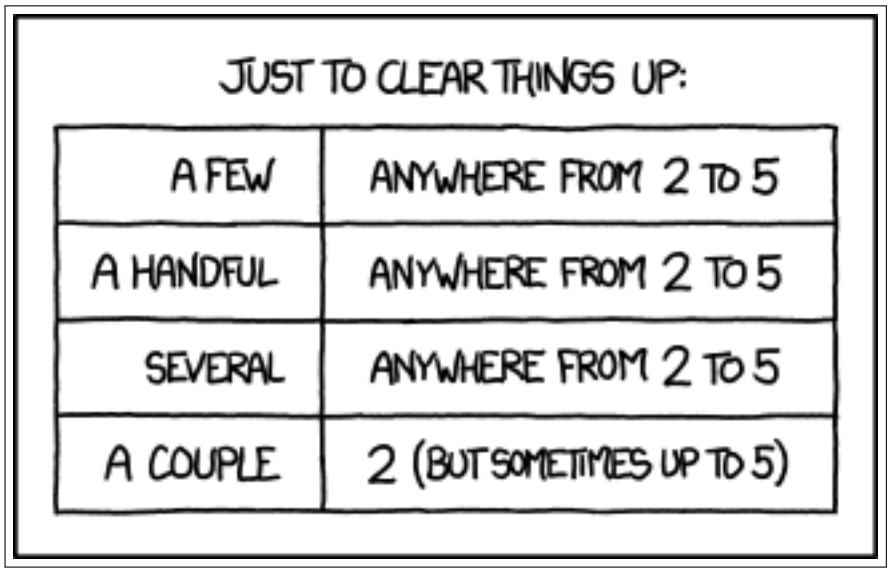


Abbildung 1.1.: Unklare Aussage für kleine Mengen [Mun12] Nr. 1070

Nach [LL07] werden Fehler typisch auf derselben Abstraktionsebene entdeckt, auf der sie begangen wurden. Daraus folgt unmittelbar, dass Fehler in der Analyse üblicherweise erst beim Betrieb des Produktes auffallen.

Die einzige praktikable Lösung für diese Probleme besteht darin, dem Analysten Software zur Verfügung zu stellen, die es ihm ermöglicht bereits beim Erfassen der Anforderungen mögliche Fehlerquellen zu identifizieren. Hierfür muss ein solches System erst einmal Anforderungen identifizieren können. Eine anschließende Unterteilung der Anforderungen in Klassen würde die Arbeit weiter erleichtern.

So wie es Schrauben zuerst nur in der Ausführung Schlitzkopf gab, werden die Anforderungen in derer Arbeit zuerst grob klassifiziert. Anschließend erlauben domänenspezifische Ontologien eine feinere Unterteilung der Anforderungen für unterschiedlichste Anwendugen. So wie es auch für Schrauben dutzende Köpfe für unterschiedlichste Anwendungsgebiete gibt.

1.1.1. Problemstellung

Das Ziel der Arbeit „Analyse und Klassifizierung von Anforderungen in technischen Spezifikationen“, ist es, soweit möglich, Anforderungen in Texten automatisiert zu bestimmen, um diese Anforderungen anschließend in Klassen unterteilen zu können. Teil des Systems wird es sein, die analysierten Anforderungen und die Klassen persistent zu lagern. Die dafür entwickelte Datenstruktur soll auch domänenspezifischen Ontologien verwalten können.

1.1.2. Extraktion von Anforderungen

Die automatisierte Extraktion von Anforderungen ist in den letzten Jahren Gegenstand verschiedener Forschungsprojekte gewesen. Es kann im allgemeinen zwischen zwei verschiedenen Vorgehensweisen unterschieden werden:

Maschinelles Lernen wird genutzt um Indizien für Anforderungen zu gewinnen. Anschließend werden mit den Indizien die Anforderungen extrahiert. Es handelt sich dabei meist um domänenspezifische Lösungen.

Filterung nach statischen Informationen führt dazu, dass Anforderungen domänenunspezifisch gewonnen werden können. Die Erkennungsrate dieses Ansatzes ist in der Regel schlechter als beim domänenspezifischen Ansatz.

1.1.3. Klassifizierung

Allgemein lassen sich Anforderungen nach der Forderung nach einer bestimmten Qualität klassifizieren. Deswegen eignen sich für die Klassifizierung domänenspezifischen Ansätze. Für eine automatisierte Klassifizierung muss zum Einen ein Klassifizierungsprozess, zum Anderen eine Qualtität nach der klassifiziert wird, gewählt werden.

Bei automatischen Prozessen für Klassifizierung werden zwischen statistischen und verteilungsfreien Verfahren unterschieden. Da die statistischen Verfahren auf Dichteberechnungen und Wahrscheinlichkeiten beruhen, lassen sich keine klaren Trennflächen angeben. Die verteilungsfreie Variante ist, aufgrund der klaren Klassentrennung, für diese Diplomarbeit das bevorzugte Vorgehen.

Eine weitere Unterscheidung geschieht zwischen überwachten und nicht überwachten Verfahren. Während bei den überwachten Prozessen des maschinellen Lernens, die Klasseneinteilung vorgegeben ist, muss der Softwareprozess bei den nicht überwachten Lösungen diese selbstständig erlernen. Da der Nutzer, des in dieser Arbeit entworfenen Werkzeugs, die Klassen vorgeben soll, wurde ein überwachtes Verfahren gewählt. Es werden Ontologien genutzt um die Klassen und die Logik zwischen den Klassen zu definieren.

1.1.4. Lösungsansatz

Aus den zuvor genannten Möglichkeiten wurde folgender Lösungsansatz entworfen. Mithilfe von Natural Language Processing (NLP) Analyse Werkzeugen werden die Wortelement der Sätze identifiziert. Abschließend nutzt eine Ontologie diese Informationen um die Anforderungen zu ermitteln. Dabei liefert eine Suche nach Schlüsselbegriffen, die erfoderlichen Indizien, ob es sich bei einem Satz um eine Anforderung handelt.

Eine Anschließende domänenspezifische Klassifizierung wird ebenfalls über Ontologien ermöglicht.

1.2. Vorgehen

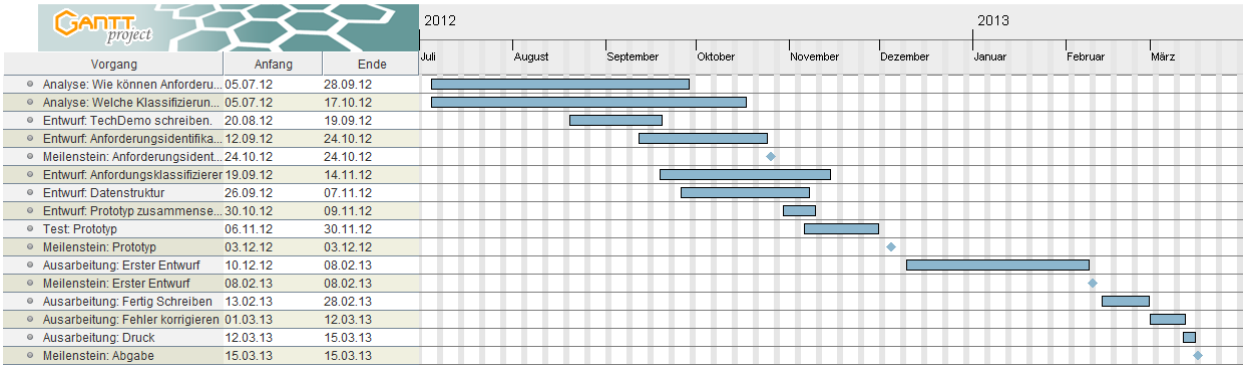
Um eine methodische Durchführung dieser Arbeit zu garantieren wurde ein Projektplan erstellt. Mithilfe von Meilensteinen wurde der zeitliche Fortschritt überwacht.

1.2.1. Projektplanung

Diese Diplomarbeit wurde als Wasserfallmodell mit überlappenden Phasen geplant. Der terminliche Ablauf dieser Arbeit ist im Gantt-Diagramm Abbildung 1.2 festgehalten.

Es wurden die folgenden vier Phasen geplant:

1. Analyse
2. Entwurf
3. Test
4. Dokumentation



Vorgang	Anfang	Ende
Analyse: Wie können Anforderu...	05.07.12	28.09.12
Analyse: Welche Klassifizierun...	05.07.12	17.10.12
Entwurf: TechDemo schreiben.	20.08.12	19.09.12
Entwurf: Anforderungsidentifika...	12.09.12	24.10.12
Meilenstein: Anforderungsident...	24.10.12	24.10.12
Entwurf: Anfordungsklassifizierer	19.09.12	14.11.12
Entwurf: Datenstruktur	26.09.12	07.11.12
Entwurf: Prototyp zusammense...	30.10.12	09.11.12
Test: Prototyp	06.11.12	30.11.12
Meilenstein: Prototyp	03.12.12	03.12.12
Ausarbeitung: Erster Entwurf	10.12.12	08.02.13
Meilenstein: Erster Entwurf	08.02.13	08.02.13
Ausarbeitung: Fertig Schreiben	13.02.13	28.02.13
Ausarbeitung: Fehler korrigieren	01.03.13	12.03.13
Ausarbeitung: Druck	12.03.13	15.03.13
Meilenstein: Abgabe	15.03.13	15.03.13

Abbildung 1.2.: Gantt Diagramm dieser Arbeit

Es ist anzumerken, dass die Analyse und der Entwurf zweiteilig geplant war. Zum Einen sollte die Extraktion von Anforderungen analysiert und implementiert werden. Die Klassifizierung stellte den zweiten Teil dar. Nachdem die Extraktionen, wie später noch aufgeführt, über die Klassifizierung abläuft, fusionierten diese beiden Arbeitspfade.

1.2.2. Analyse

Die Analyse dieser Abeit stützt sich vorallem auf die Artikel von Vlas und Robinson [VR11], Cleland-Huang et. al. [CHSZS06] und die Arbeit von Farfeleder et. al. [FMK+11]. Am Begin der Literaturerecherche standen die Biblothek der Universität Stuttgart, der IEEE Xplore Katalog, die ACM digital library, die E-Books des Springer Verlags, das Internet über allgemeine Suchmaschinen, wie Google Suche[1], sowie spezielle Suchmaschinen, wie Google Scholar [2]. Neben wissenschaftlichen Veröffentlichungen und Büchern wurden auch verschiedene Webpräsenzen für eine umfangreiche Recherche genutzt.

1.3. Gliederung

Kapitel 1 präsentiert eine Einführung in die Themen dieser Arbeit.

Kapitel 2 geht auf die Grundlagen dieser Arbeit ein. Dazu gehören Maschinelle Sprachverarbeitung und Ontologien.

[1] http://www.google.de
[2] http://scholar.google.com

Kapitel 3 betrachtet verwandten Arbeiten und die von dieser Arbeit eingesetzen Technologien.

Kapitel 4 wird die Ausgangslage, den Lösugsansatz, und alternative Lösungen dieser Arbeit präsentieren.

Kapitel 5 stellt die Implementierung des Systems vor. Es beginnt mit einer Erläuterung der Datenstruktur und folgt mit einer Erklärung der Komponenten anhand des Programablaufs.

Kapitel 6 behandelt die Ergebnisse die bei der Klassifizierung technischer Spezifikationen erzielt wurden.

Kapitel 7 schließt die Arbeit mit einer detailierten Zusammenfassung der Hindernisse und des Projektes ab. Zusätzlich liefert es einen Ausblick darauf, wie sich das REC System erweitern lässt.

2. Grundlagen

Zunächst werden in diesem Kapitel die Grundlagen der Arbeit präsentiert. Den Anfang macht eine Erläuterung zur MDA. Danach wird die Anforderungsanalyse erklärt, in der beschrieben wird, was Anforderungen sind und welche Probleme bei der Erhebung und Analyse von Anforderungen auftreten können. Darauf folgt eine allgemeinen Beschreibung der Maschinelle Sprachverarbeitung. Den Abschluss bilden die Ontologien.

2.1. Model Driven Architecture

Unter MDA versteht man einen modellgetriebenen Softwareentwicklungsansatz. Er beruht auf einer klaren Trennung von Geschäfts- und Anwendungslogik von der Plattformtechnologie. Eine Spezifikation wurde von der Object Managment Group (OMG) als wissenschaftliche Arbeit [MM03] herausgegeben.

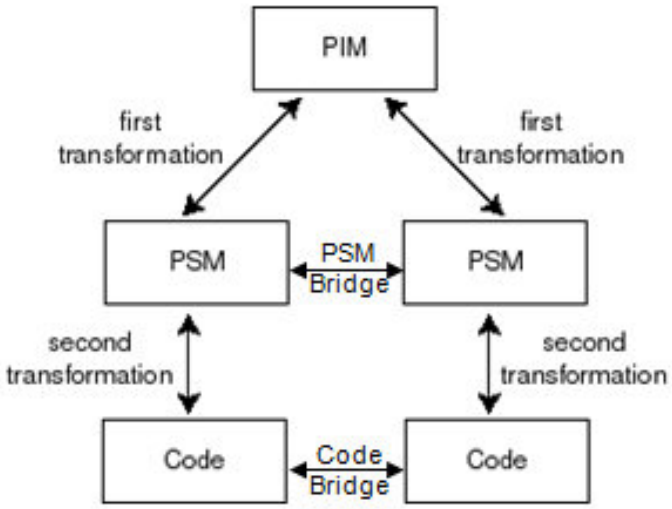


Abbildung 2.1.: Das Vorgehen der MDA [SA 12]

Der grundlegende Ansatz der OMG sieht vor, dass die fachlichen Spezifikationen zuerst in plattformunabhängigen Modellen (Computation Independant Model (CIM)) definiert werden. Die Modelle sollen dabei von der späteren Zielplatform vollständig unabhängig sein. Durch Modelltransformation werden nun die CIM in Domänenmodelle (Platform

Independent Model (PIM)) umgewandelt. Es kommen also zu den plattformunabhängigen fachlichen Spezifikationen nun die spezifischen Konzepte der Zieldomäne hinzu. Die Modelltransformationen werden in der Regel durch automatisierte Werkzeuge bewältigt. Eine weitere Transformationen gewinnt nun aus dem PIM eine Implementierung (Platform Specific Model (PSM)) für die Zielplattform.

Mit der MDA verfolgt die OMG zwei Ziele. Zum Einen soll dur MDA die Entwicklungsgeschwindigkeit gesteigert werden. „Automation durch Formalisierung“ ist das Stichwort zum Erreichen dieses Zieles. Die automatisierte Transformationen der Modelle in Programmcode spart Zeit. Zeit den Code zu implementieren, aber auch Zeit durch weniger Fehler, die einem Menschen beim Programmieren unterlaufen. Natürlich müssen die Modelle einwandfrei formuliert sein, da sich sonst die Fehlerquelle nur verlagert. Die Erzeugung solcher Modelle erfordert somit mehr Zeit. Der Nutzen von MDA ist daher stark von der Größe und Komplexität des zu entwerfenden Systems abhängig. Ein weiteres Ziel der MDA ist es bei hoch komplexen Systemen die Gebrauchstauglichkeit zu verbessern. Dies soll vor allem mithilfe von Abstraktion durch Modellierungssprachen bewerkstelligt werden. Der Gedanke ist der, dass abstrakte, technologieunabhängige Beschreibungen der Schlüsselkonzepte dafür sorgt, dass späterer Technologiewandel einfacher zu handhaben ist.

Trotz des großem Automatisierungsgrades der MDA bleibt die Erzeugung der CIM Aufgabe der Analysten.

2.2. Anforderungsanalyse

Die Anforderungsanalyse ist ein wichtiger Bestandteil bei der Erzeugung von CIMs. Es folgt deswegen eine Erklärung, was Anforderungen sind. Anschließend werden mögliche Schwierigkeiten, die bei der Analyse auftreten können, erläutert.

2.2.1. Anforderungen

Anforderungen treffen Aussagen über die Beschaffenheit oder Fähigkeit, die ein System erfüllen oder besitzen muss um Vorgaben zu entsprechen. Eine formalere Definition, was eine Aussage ist, liefert die Qualitätsmanagementnorm DIN EN ISO 9000:2005. Sie definiert eine Anforderung als

> ein Erfordernis oder eine Erwartung, das oder die festgelegt, üblicherweise vorausgesetzt oder verpflichtend ist.

Es ist ersichtlich, dass eine klare Formulierung der Anforderungen essentiell ist, für eine fehlerfreie Kommunikation zwischen dem Anforderungssteller und dem Auftragnehmer.

Anforderungsklassen

Ludewig und Lichter [LL07] unterscheiden Anforderungen nach verschiedenen Kriterien.

Die offenen gegenüber den latenten Anforderungen. Offene sind Anforderungen, von denen sich der Kunde bewusst ist, dass sie existieren. Die latenten Anforderungen sind dem Kunden nicht bewusst, bis ihre Bedeutung ihm vor Augen geführt wird. Der Analyst muss daher durch gezielte Fragen versuchen, dem Kunden möglichst viele latente Anforderungen bewusst zu machen.

Als zweite Abstufung unterschieden Ludewig und Lichter zwischen harten und weichen Anforderungen. Um eine harte Anforderung handelt es sich, wenn dabei ganze Zahlen berechnet werden, oder „Ja/Nein"-Fragen beantwortet werden müssen. Eine weiche Anforderung ist dem entsprechend Eine, bei der es einen fließenden Übergang zwischen Richtig und Falsch gibt.

Vor allem für Testzwecke werden Anforderungen auch nach ihrer Objektivierbarkeit unterschieden. Objektivierbare Anforderungen sind solche, bei denen die Resultate objektiv überprüft werden können, während vage Anforderungen diejenigen sind, die sich nicht oder nur schlecht quantifizieren lassen.

Als letzte Einteilung ist die zwischen funktionalen und nicht-funktionalen Anforderungen üblich. Funktionale Anforderungen beschreiben alle Aspekte eines Systems, die für die Durchführung der Kernaufgabe erforderlich sind. Nicht-funktionale Anforderungen decken zwei Bereiche ab:

1. Eigenschaften, die das System als Ganzes beeinflussen (wie Benutzbarkeit, Wartbarkeit, Flexibilität, ...)
2. Qualitätsattribute (Genauigkeit, Antwortzeit, Zuverlässigkeit, Sicherheit, ...)

2.2.2. Analyse

Anforderungen werden üblicherweise in Gesprächen und im Schriftverkehr erhoben und werden daher in natürlicher Sprache verfasst. Die natürliche Sprache eignet sich aber nur bedingt zur Darstellung von strukturierten Informationen. Sie enthält Mehrdeutigkeiten und Unklarheiten, die einem ungeübten Ersteller nicht offensichtlich klar sind.

> Die Software soll automatisch Anforderungen extrahieren und klassifizieren.

Diesen Satz kann man auf zwei Weisen interpretieren. Zum Einen kann man den Satz so verstehen, dass Anforderungen automatisch extrahiert und automatisch klassifiziert werden sollen. Alternativ kann man ihn aber auch so auffassen, dass Anforderungen zwar automatisch extrahiert, aber lediglich manuell klassifiziert werden sollen.

Diese Mehrdeutigkeit in der natürlichen Sprache wird bei der Entwicklung von Systemen zu einem beachtenswerten Problem. Selbst wenn sich Kunde und Analyst einig sind, was mit einer Anforderung gemeint ist, so bedeutet das nicht, dass auch der Entwickler weiß, was

verlangt ist. Oftmals hat dann der Entwickler aber keinen Zugriff mehr auf den Kunden und es kommt zu Fehlern im Produkt.

In der Softwareentwicklung bietet es sich deswegen an, Anforderungen mit formalen Methoden zu beschreiben. Mit ihnen kann die Software vollständig validiert und verifiziert werden. Insbesondere im Test erlauben formale Methoden eine objektive Überprüfung auf korrekte Umsetzung der Spezifikation in Software. Es kann jedoch nicht davon ausgegangen werden, dass der Kunde versiert genug ist eine formale Spezifikation zu verstehen [Lan10].

Withall [Wit07] schildert dazu in seinen Beobachtungen zur Anforderungsanalyse:

> Das Einzige was man mit schriftlichen Anforderungen machen kann, ist sie lesen. Es ist unnötig zeitaufwendig und anfällig für menschliche Fehler. Des Weiteren gilt, dass wenn Textstücke eine spezielle Formatierung haben, dies auf eine spezielle Bedeutung hinweist. Wenn es möglich wäre, die Bedeutung direkt hervorzuheben, so würde es uns ein ganzes Stück weiter bringen, es maschinenverständlich zu machen - also die Bedeutung auf andere Weise extrahieren, als dass ein Mensch den Text lesen muss.

Wünschenswert wäre ein System, bei dem der Kunde zusammen mit dem Analysten die Anforderungen erstellt und das System anschließend ad hoc automatisch aus den Anforderungen in natürlicher Sprache formale Modelle erzeugt. Diese Modelle könnten nun auf Widersprüche, fehlende oder unvollständige Anforderungen und auf Mehrdeutigkeiten der Anforderungen geprüft werden. Analyst und Kunde könnten dann sofort diese Fehlerquellen bearbeiten. Eine Marktstudie von Mich et al. aus dem Jahre 1999 [MFI04] zeigt, dass der Großteil des Marktes Interesse an einer automatisierten Lösung hat.

Eine automatisierte Umwandlung von natürlicher Sprache in formale Modelle setzt maschinelle Sprachverarbeitung voraus, die deswegen nun vorgestellt wird.

2.3. Maschinelle Sprachverarbeitung

Die maschinelle Sprachverarbeitung ist ein Teil der Computerlinguistik. Man spricht auch von der natürlichen Sprachverarbeitung (NLP). Sie beschäftigt sich mit zwei Dingen. Zum Einen den Vorgang eines Computers aus einer, in natürlicher Sprache formulierten, Eingabe bedeutungsvolle (sinnvolle) Informationen zu extrahieren. Zum Anderen den Vorgang eines Computers eine Ausgabe in natürlicher Sprache zu erzeugen. Von besonderem Interesse für diese Arbeit ist die Extraktion von Informationen, im Speziellen die Extraktion von Anforderungen aus einem natürlich sprachlichen Text.

Um Informationen aus einem Text zu gewinnen, wird in dieser Arbeit das Saarbrücker Pipelinemodell betrachtet. In diesem Modell der maschinellen Sprachverarbeitung werden folgende Schritte der Reihe nach durchgeführt

- Spracherkennung
- Tokenisierung
- Morphologische Analyse
- Syntaktische Analyse
- Semantische Analyse
- Dialog- und Diskursanalyse

Die Spracherkennung versucht Text, der als Schall vorliegt, in Textform umzuwandeln. Da die hier betrachteten Texte in schriftlicher Form vorliegen, wird sie nicht benötigt.

2.3.1. Tokenisierung

Bei der Tokenisierung wird der Text, den der Computer nur als Kette von Zeichen interpretiert, in Wörter, Sätze und andere Textbestandteile aufgeteilt. Sie ist Voraussetzung für Weiterverarbeitung eines Textes. Die White-Space-Tokenisierung ist die einfachste Form, bei welcher der Text an den Leer- und Interpunktionszeichen aufgetrennt wird. Bei anderen Verfahren werden Token aus Folgen von Buchstaben oder Ziffern gebildet. Alle anderen Zeichen bilden für sich genommen je ein Token. Beide Verfahren haben jedoch Probleme mit Mehrwortlexemen, Eigennamen oder Währungsangaben. Lexeme bilden die Bausteine des Wortschatzes, sie sind lexikalische Einheiten, die in festen strukturellen Beziehungen stehen. Beispielsweise sind singen, singt, singst Instanzen des Lexems „singen“. Sänger und singen jedoch sind zwei Lexeme.

Betrachten wir nun einen Beispielsatz:

> The navigation system must be placed in the center console.

Der Parser des Stanford CoreNLP fügt folgende Token dem Satz hinzu:

> The/DT navigation/NN system/NN must/MD be/VB placed/VBN in/IN the/DT center/NN console/NN ./.

Wie erwartet werden „navigation system“ und „center console“ als Nomen und „be placed“ als Verb erkannt. Dabei wird auch bereits die Part-of-Speech (POS) Analyse vollzogen.

POS Ein POS Tagger ließt einen Text in einer Sprache ein und weist jedem Wort seinen "part of speech"hinzu. Er bestimmt also, ob ein Wort ein Verb, Nomen, Adverb, Adjektiv, usw. ist.

2.3.2. Morphologische Analyse

Mithilfe der morphologischen Analyse werden die Wörter eines Satzes in ihre Grundform gebracht. Es werden also die grammatikalischen Informationen, die sogenannte Flexion oder Beugung, aus dem Text extrahiert. Aber auch Wortbildungsmuster werden untersucht, wie zum Beispiel die Zusammensetzung von Navigationssystem aus Navigation und System.

Um die Lemmatarisierung eine Satzes zu verdeutlichen folgendes Beispiel.

Wort	Extracting	keywords	is	not	a	new	idea	.
Lemma	extract	keyword	be	not	a	new	idea	.

2.3.3. Syntaktische Analyse

Da die morphologische Analyse nichts über die Beziehungen der Wortformen im Satz aussagt, gibt es die syntaktische Analyse. Die grammatikalischen Beziehungen bilden den Gegenstandsbereich der Syntax. Erst durch die genaue Analyse der syntaktischen Struktur ist es möglich, einen Satz zu interpretieren. Dazu gehört zum Beispiel das Ermitteln der strukturellen Funktion der Wörter in einem Satz (Subjekt, Objekt, Modifikator, Artikel, usw.). Die syntaktische Struktur lässt sich als Baum-Diagramm darstellen (siehe Abb. 2.2).

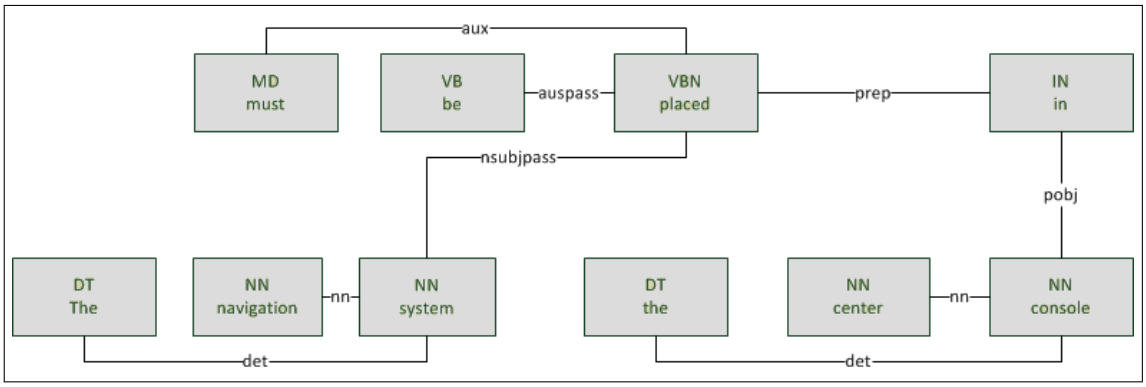


Abbildung 2.2.: Syntakische Analyse (Stanford CoreNLP) des Satzes: The navigation system must be placed in the center console

2.3.4. Semantische Analyse

Die semantische Analyse liefert satzweise die Bedeutung des Textes. Sie übersetzt dabei syntaktischen Bezüge in logische Operationen. In Bezug auf diese Arbeit handelt es sich dabei um die Erkennung der Anforderungen. Die im nächsten Abschnitt erläuterten Ontologien werden diese Aufgabe übernehmen.

Anwendung in dieser Arbeit

Für Tokenisierung, morphologische Analyse und syntaktische Analyse wird auf bestehende Softwarelösungen zurückgegriffen. Die Dialog- und Diskursanalyse versucht Beziehungen zwischen aufeinander folgenden Sätzen zu erkennen. So werden zum Beispiel Kombinationen aus Frage und Antwort, aber auch Aussage und Begründung erkannt. Durch die spezielle Struktur von technischen Spezifikationen wird auf diese beiden Formen der Analyse verzichtet.

2.4. Ontologien

Der Begriff Ontologie stammt aus der Philosophie. Traditionell wurde dieser Bereich „allgemeine Metaphysik“ genannt. Deshalb werden „Metaphysik“ und „Ontologie“ synonym verwendet.

Der zwischen 1693 und 1775 lebende Theologe Johann Georg Walch definiert in seinem „Philosophisches Lexicon“:

> Ontologie bedeutet die Lehre vom Sein und ist eine Benennung, womit einige neuere Philosophen die Wissenschaft, die vom Sein überhaupt und dessen Eigenschaften handelt, verstanden.

Ungefähr hundert Jahre zuvor gibt es erste Belege für den Gebrauch des Wortes Ontologie in der deutschen Sprache.

Ontologie steht für die Lehre vom Sein. Genauer gesagt, von den Möglichkeiten und Bedingungen des Seienden. Es geht also um die Grundstrukturen der Wirklichkeit. Diskutiert wird dabei über eine Systematik grundlegender Typen von Entitäten und ihrer strukturellen Beziehungen zueinander.

2.4.1. Ontologien in der Informatik

In der Informatik dienen Ontologien als Mittel der Strukturierung und zum Datenaustausch. Im Gegensatz zu Datenbanken werden in Ontologien nicht nur Mengen von Begrifflichkeiten in einer formalen Darstellung geordnet, sondern auch die Beziehungen, in einem bestimmten Umfeld zwischen ihnen.

> „Ontologie ist eine explizite formale Spezifikation einer gemeinsamen Konzeptualisierung“ - T. Gruber

Durch Integritätsregeln wird es möglich, die Gültigkeit einer Ontologie zu gewährleisten. Zusätzlich ermöglichen es Inferenzregel Schlussfolgerungen aus den meist sprachlich gefassten Begrifflichkeiten zu schließen. Wie auch bei Datenbanken ist es somit möglich mit Ontologien Wissensbestände zusammen zu führen, in ihnen zu suchen und sie zu editieren.

Da Ontologien im Gegensatz zu Datenbanken auch Regeln über den Zusammenhang der Daten besitzen, ist es möglich, Rückschlüsse aus den Daten zu ziehen oder auch Widersprüche innerhalb der Datenbestände zu erkennen.

Dank des „semantischen Webs“ haben Ontologien in den letzten Jahren an Beliebtheit gewonnen. Sie werden gerne als Wissensrepräsentation im Gebiet der künstlichen Intelligenz genutzt.

Die Inferenzberechnungen in Ontologien werden von sogenannten Inferenzmaschinen erledigt. Der, in der englischen Sprache gängige Begriff des „Reasoner“ hat sich auch im deutschsprachigen Raum eingebürgert. Die beiden Begriffe „Inferenzmaschine“ und „Reasoner“ werden deswegen in dieser Arbeit als äquivalent betrachtet.

Bestandteile von Ontologien

Begriffe sind die Beschreibung gemeinsamer Eigenschaften (z.B. der Begriff „Wort“, der alle Wörter enthält). Sie werden auch als Klassen oder Konzepte bezeichnet.

Typen repräsentieren Objekttypen in der Ontologie.

Instanzen sind die Objekte in der Ontologie. Sie repräsentieren damit das Wissen der Ontologie. Ein anderer Begriff für Instanzen ist der der Individuen.

Relationen beschreiben die Beziehungen zwischen Instanzen (z.B. das Wort „Auto“ *ist Teil* vom Satz „Das Auto ist rot.“).

Vererbung von Relationen und Eigenschaften ist in Ontologien möglich. Dabei werden alle Eigenschaften an das Kind Element weitergegeben.

Axiome sind Aussagen innerhalb der Ontologie die immer wahr sind. Verwendet werden sie normalerweise, um Wissen darzustellen, dass nicht abgeleitet werden kann (z.B. „Auto ist ein Nomen.“).

3. Verwandte Arbeiten und genutzte Technologien

Liefern Grundlagen die Bausteine, so findet man in den verwandten Arbeiten Hinweise darauf, wie man die Bausteine nutzen kann. Deshalb werden jetzt zunächst die wichtigsten themenverwandten Arbeiten und anschließend die genutzten Technologien vorgestellt.

3.1. Zielverwandte Arbeiten

Im Bereich der automatisierten Anforderungsextraktion gibt es diverse Herangehensweisen. In [VR11] kommen Ontologien zu Einsatz um Anforderungen in Open Source Software Entwicklung (Open Source Software Development) (OSSD) Projekten aufzufinden. Auch in [CHSZS06] werden Ontologien genutzt, mit dem Unterschied, dass durch statistische Methoden hier Ontologien trainiert werden. Dies ermöglicht eine Suche nach speziellen Termen mit großer Genauigkeit. Andere ([AHH06]) nutzen NLP Technologien um Anforderungen in XML und dann in Unified Modeling Language (UML) Modelle umzuwandeln.

3.1.1. A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects

Auch in Open Source Projekten existieren Anforderungen. In Foren, Funktionswünschen und anderer Korrespondenz werden die Anforderungen meist nur informell in Textform verfasst. In ihrer Arbeit beschreiben Vlas und Robinson [VR11] eine Methode, um natürlich sprachliche Anforderungen automatisiert analysieren und klassifizieren zu können. Sie stützen sich auf Arbeiten von Scacchi, der in der OSSD zwei Dutzend Artefakte (Chat, E-Mail, Foren, ...) identifiziert, die unstrukturiert festhalten, was die Software erledigen soll. Der Anforderungsklassifizierer von Vlas und Robinson soll Forschern dabei helfen, Muster und Entwicklungen in OSSD zu erkennen.

Eins der Hauptprobleme beim Entdecken von Anforderungen in natürlich sprachlichen Texten besteht darin, dass Texte wie sie in OSSD Projekten vorkommen, nicht nur Anforderungen enthalten. Sie enthalten oft auch soziale Kommunikation oder Code Fragmente. Zusätzlich führen Umgangssprache, Schreibfehler, Abkürzungen und Emoticons zu weiteren Problemen beim Identifizieren von Anforderungen. Der erste Schritt besteht darin, die Anforderungen innerhalb einer Quelle abzugrenzen. Dann erst kann damit begonnen werden die Anforderungen zu klassifizieren. Bei der Klassifizierung von Anforderungen wird, wie

in Kapitel 2.2 beschrieben, zwischen funktionalen Anforderungen und nicht-funktionalen Anforderungen unterschieden. Weitere Klassifizierungen der Anforderungen werden über eine Agenten-basierende Perspektive geregelt.

Vlas und Robinson betrachten musterbasierte Erkennungssysteme für Anforderungen. Dabei wird Wissen über bisherige Anforderungen genutzt, um fehlende Informationen finden zu können und diese Inkonsistenzen zu "beheben".

In ihrem System wird eine mehrstufige Ontologie genutzt. Die unteren Ebenen sind Grammatik basierend während die höheren Ebenen Anforderungs basierend sind. Da die Textquellen in OSSD nicht gezwungenermaßen mit der Englischen Grammatik übereinstimmen, ist ihr Klassifizierer eine Art schwaches Ontologie basierendes Informationsextraktionssystem. Die Ontologie des „Requirement Classifier for Natural Language“ beinhaltet sechs Stufen. Die ersten beiden Stufen behandeln die üblichen grammatikalischen Konzepte der natürlichen Sprache. Dazu gehören Tokenisierung, und die „Parts-of-Speech“ Analyse. Die nächsten drei Ebenen beinhalten Konzepte für logische Aussagen: Es beginnt mit einer Qualifizierung, dabei werden Ausdrücke identifiziert, die auf das Vorkommen einer Anforderung hinweisen. Die nächste Stufe erkennt die drei fundamentalen Elemente einer Anforderung, Subjekt, Verb und Objekt. In der letzten Stufe werden die Klassifizierungskonzepte angewendet, um die Anforderungen zu erkennen.

3.1.2. The Detection and Classification of Non-Functional Requirements with Application to Early Aspects

In ihrer Arbeit stellen Cleland-Huang et. al. [CHSZS06] einen auf Informationsrückgewinnung basierenden Ansatz vor, mit dem sich das Entdecken und die Klassifikation von Nicht-funktionalen Anforderungen (Non-functional Requirements) (NFR) automatisieren lässt. Die Früherkennung von NFRs ist sinnvoll, um frühzeitig Systemeinschränkungen beim Planen der Architektur zu berücksichtigen. NFRs können sowohl in strukturierten, als auch unstrukturierten Dokumenten erkannt werden.

Klassischerweise handelt es sich bei NFRs um Anforderungen an Sicherheit, Performanz, Verfügbarkeit, Erweiterbarkeit und Portabilität. Es ist sinnvoll diese nicht-funktionalen Anforderungen bereits bei der Architektur einen Systems zu berücksichtigen. Das Hauptproblem besteht darin, dass NFRs meist erst relativ spät im Entwicklungsprozess entdeckt werden. Dieses Projekt soll dem Abhilfe verschaffen, indem NFRs automatisch aus Dokumenten extrahiert werden.

Der NFR-Klassifizierer nutzt dabei „Indikator Ausdrücke“ (indicator terms), Begriffe die für bestimmte NFR Typen erlernt werden. Zuerst muss also ein Analyst Anforderungen spezifizieren, die der „Indicator miner“ analysieren kann, um daraus die Indikator Ausdrücke zu erlernen. Woraufhin der NFR-Klassifizierer Texte analysieren kann, um daraus NFR Kandidaten zu extrahieren. Diese Kandidaten müssen von einem Analysten bestätigt werden.

Recall	Anzahl an NFRs die korrekt entdeckt und kategorisiert wurden.	$\frac{TruePositives}{(TruePositives+FalseNegatives)}$
Precision	Anzahl der korrekt erkannten NFRs in Bezug auf Anzahl der entdeckten NFRs.	$\frac{TruePositives}{(TruePositives+FalsePositives)}$
Specificity	Fähigkeit des Klassifizierers NFRs zu verwerfen, die vom falschen Typ sind.	$\frac{TrueNegatives}{(TrueNegatives+FalsePositives)}$

Tabelle 3.1.: Als Standard Metriken zu Evaluation verwenden [CHSZS06] „recall“, „precision“ und „specificity“.

Ein erster Ansatz bestand darin, die Indikatorausdrücke im Vorfeld festzulegen. Hierbei wurde jedoch festgestellt, dass die Präzision bei 40 - 60% lag. Sie entdeckten, dass viele der Ausgewählten Schlüsselbegriffe von mehreren NFR Typen geteilt wurden.

Der aktuelle NFR-Klassifizierer geht dieses Problem an, indem er auf einer Trainingsmenge gewichtete Indikatorausdrücke für jeden NFR-Typen findet. Der NFR-Klassifizierer kann also nur NFRs erkennen und zu Tage fördern, für die er trainiert wurde. Sie versuchen diesen Nachteil dadurch auszugleichen, dass er sich mit bereits existierenden prekategorisierten Anforderungsspezifikationen trainieren lässt. Er ist also anpassbar für die spezifischen Terme einer Organisation.

3.1.3. Ontology-Driven Guidance for Requirements Elicitation

Farfeleder et.al. [FMK+11] schlagen ein System vor, dass Domänenspezifische Ontologie verwendet, um bei der Anforderungserhebung bereits Analyse durchzuführen und Vorschläge zu machen. Das System soll automatisch Teile der Anforderungen vorschlagen, indem es Informationen aus der Domänen Ontologie verwendet. Außerdem soll es Relationen und Axiome der Ontologie nutzen, um mehr als nur ein Wörterbuch zu sein. Dafür haben sie ihr Vorlagen Erhebungswerkzeug (DODT) erweitert.

Vorlagen haben zum Beispiel die Form „<System> soll <Aktion>“, wobei <System> und <Aktion> Attribute sind und „soll“ ein festes Syntaxelement ist.

In ihrem Werkzeug wählt der Anforderungsingenieur eine Vorlage aus. Die Liste der möglichen Attribute wird auf die Entitäten der Domänen Ontologie beschränkt. Ein semantisches Leitsystem beeinflusst die Ontologie, die Attributwerte und die Instanziierung.

Die Ontologie verwaltet die Fakten der Domäne, die für das Anforderungsengineering relevant sind. Wissen, das nur für ein Projekt nützlich ist soll dabei nicht hinzugefügt werden. Über eine Domäne werden drei Arten von Fakten gelagert.

Konzept Ein Konzept repräsentiert eine Entität in der Domäne. OWL Klassen werden verwendet, um Konzepte darzustellen. Der Grund dafür ist, dass sich Klassen, im Gegensatz zu Individuen, erweitern lassen. Ein Konzept hat zwei Attribute, seinen

Attribut	Beschreibung	Beispiel
<Aktion>	Ein Verhalten von dem erwartet wird, dass das System es erfüllt, oder eine Fähigkeit.	"öffne die Tür"
<Entität>	Eine getrennte Entität der Domäne, die nicht in <Nutzer> oder <System> passt.	"Türstatus"
<Nummer>	Ein numerischer Wert, er eine Anzahl beschreibt.	"100"
<Operationale Bedingung>	Eine Bedingung, oder ein Event, das während des Betriebs auftritt.	"Der Nutzer versucht die Tür zu öffnen"
<System>	Ein Teil des Systems, Subklasse von Entität.	"Tür"
<Einheit>	Eine Maßeinheit	"Millisekunden"
<Nutzer>	Eine Person die mit dem System interagiert, Subklasse von Entität.	"Pilot"

Tabelle 3.2.: Attribute der Vorlagen

Namen und eine Definition. Die Definition soll die Möglichkeit bieten, zu überprüfen ob die richtige Terminologie genutzt wurde.

Beziehung Eine Beziehung ist eine beschriftete, gerichtete Verbindung zwischen zwei Konzepten. Als Beschriftung wird ein Verb erwartet. Damit bilden Beziehungen SVO-Triplets.

Axiom Es gibt zwei relevante Axiome, Subklasse und Äquivalenz. Die Erste klassifiziert, dass ein Konzept eine Subklasse eines Anderen ist. Das Äquivalenzaxiom wird verwendet, um auszudrücken, dass zwei Konzepte mit verschiedenen Namen sich auf dieselbe Entität beziehen.

3.2. Genutze Technologien

Die, von dieser Arbeit eingesetzen, Technologien werden im Folgenden vorgestellt. Den Anfang macht das verwendete NLP Werkzeug. Daraufhin werden die für Ontologien relevanten Systeme präsentiert. Abschließend wird das Framework, auf das diese Arbeit aufbaut, vorgestellt.

3.2.1. Stanford CoreNLP

Die Stanford Natural Language Processing Group [Sta] ist eine der führenden Gruppen auf dem Gebiet des NLP. Ihre Arbeiten reichen von Grundlagenforschung auf dem Gebiet der Sprachverarbeitung bis hin zu Anwendungen in der menschlichen Sprachtechnologie. Ein vielfältiges Angebot von probailistischen Parsern, aufbauend auf kontextfreien Grammatiken, ist eines der Resultate ihrer Forschung.

Eigenschaftsname	Annotator Klassenname	Beschreibung
Tokenize	PTBTokenizerAnnotator	Fügt Token dem Text hinzu.
Cleanxml	CleanXML Annotator	Entfernt XML Token aus dem Dokument.
Ssplit	WordToSentenceAnnotator	Teilt eine Sequenz von Token in Sätze.
Pos	POSTaggerAnnotator	Beschriftet Token mit ihren POS Tag.
Lemma	MorphaAnnotator	Erzeugt Wort Lemmas für alle Token im Corpus.
Ner	NETClassifierCombiner	Erkennt benannte (PERSON, LOCATION, ORGANISATION, MISC) und numerische (DATE, TIME, MONEY, NUMBER) Entitäten.
Regexner	RegexNERAnnotator	Implementiert einen Regelbasierenden NER mit Hilfe von Java regulären Ausdrücken.
Truecase	TrueCaseAnnotator	Erkennt den wahren Fall von Token im Text, bei dem diese Information verloren gegangen ist (z.B nur Großschreibung).
Parse	ParserAnnotator	Liefert eine vollständige syntaktische Analyse, verwendet dabei Bestandteil und Abhängigkeit Representationen.
Dcoref	DeterministicCorefAnnotator	Implementiert pronominale und nominale Coreferenz Auflösung.

Tabelle 3.3.: Annotatoren des Stanford CoreNLP [Sta12b]

Der REC nutzt den Stanford CoreNLP für sämtliche NLP Analysen. Der Stanford CoreNLP deckt eine Folge von NLP Programmen für die Englische Sprache ab. Er ist in Java geschrieben und zeichnet sich dadurch aus, dass er nicht trainiert werden muss. Er nimmt Texteingaben in Englischer Sprache an und gibt einen annotierten Satz zurück. Dabei werden Wörter in ihre Grundform gebracht, ihre POS bestimmt, Namen aufgelöst, Daten, Zeiten und numerische Mengen normalisiert und die Struktur des Satzes aufgelöst. Einzelne Bestandteile des Stanford CoreNLP existieren auch für andere Sprachen, unter anderem Deutsch.

Die Grundstruktur des CoreNLP Packetes besteht aus zwei Klassen: Annotation und Annotator. Annotations sind die Datenstrukturen, in denen die Annotationen gespeichert sind.

Annotationen sind im Grunde genommen Maps, die Schlüssel auf verschiedene Annotationen abbilden. Annotators (siehe Abb. 3.3) sind Funktionen, die auf Annotationen arbeiten, anstatt auf Objekten. Sie erledigen Aufgaben wie Tokenisierung, Parsing, oder NER Tag Sätze (erkennt benannte und numerische Entitäten). Annotators und Annotations werden in AnnotationPiplines integriert, die Sequenzen von generischen Annotators erzeugen.

3.2.2. Web Ontology Language

Diese Arbeit nutzt die Web Ontology Language (OWL). OWL ist eine Spezifikation des World Wide Web Consortium (W3C), um Ontologien anhand einer formalen Beschreibungssprache erstellen, publizieren und verteilen zu können. OWL existiert in mehreren Sprachebenen, wobei die höchste Ebene die Prädikatenlogik höheren Grades beherrscht. Die OWL Spezifikation erweitert das Ressource Description Framework (RDF) und RDF-Schema. OWL nutzt Internationalized Ressource Identifier (IRI), um alle Ressourcen eindeutig zu identifizieren. Die allgemeine Struktur von OWL 2 wird in Abbildung 3.1 dargestellt.

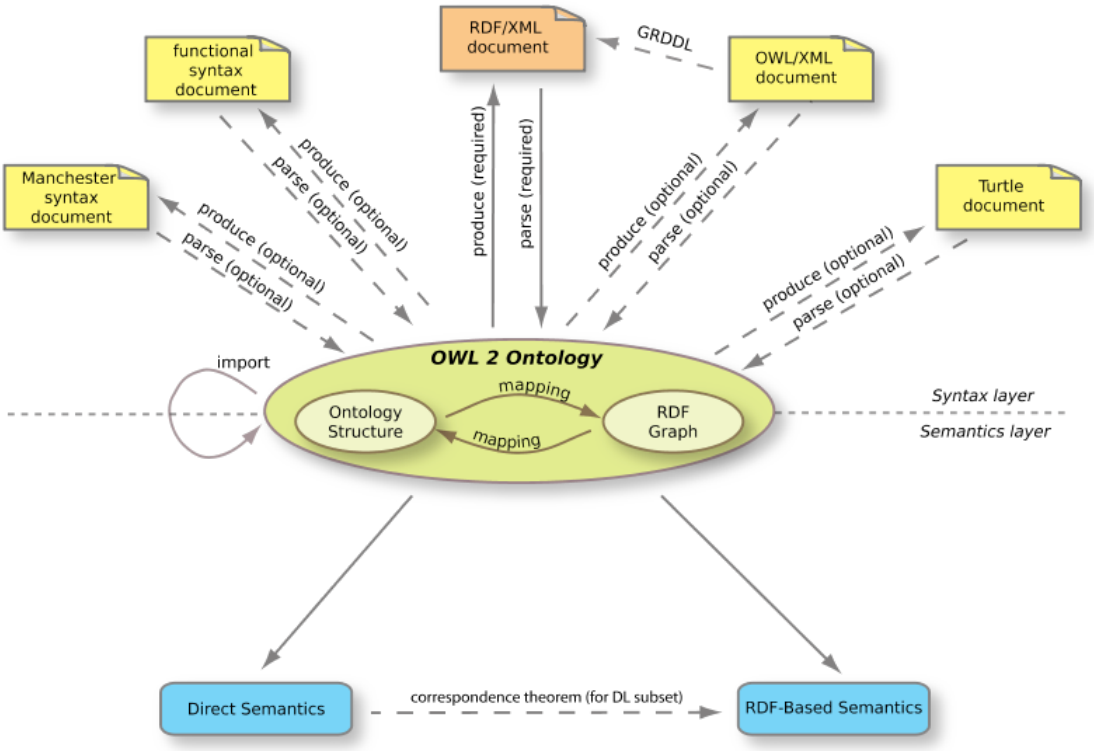


Abbildung 3.1.: Struktur von OWL 2 nach [W3C12]

Sprachebenen von OWL

OWL existiert in den Versionen Lite, DL und Full. Lite dient vor allem zum Erschaffen einfacher Taxonomien und leicht axiomatisierter Ontologien. In DL (description logic) wird die Beschreibungslogik implementiert. Die Beschreibungslogik ist äquivalent zu einer entscheidbaren Untermenge der Prädikatenlogik erster Stufe. Um diese Logik zu gewährleisten, wurden diverse Einschränkungen implementiert: so darf zum Beispiel eine Klasse keine Instanz einer anderen Klasse sein. OWL Full verzichtet auf diese Einschränkungen. Die daraus resultierenden Ontologien sind somit unentscheidbar, aber ermöglichen prädikatenlogische Ausdrücke höheren Grades.

OWL API

Um Ontologien zu verwalten, wird die OWL API [Uni12b] verwendet. Es handelt sich dabei um eine Java API und Referenzimplementierung für die Erzeugung, Manipulierung und Serialisierung von OWL Ontologien. Sie wurde gewählt, da sie unter Anderem durch die Möglichkeit unterschiedliche Inferenzmaschinen zu nutzen, eine hohe Flexibilität liefert. Des Weiteren bietet sie Möglichkeiten, um Ontlogien in verschiedenen Dateiformaten zu speichern und zu laden. Dies ermöglicht es theoretisch, Ontologien, die für den REC erzeugt wurden auch in anderen mit OWL kompatiblen Programmen zu nutzen.

Die Inferenzmaschinen werden über Interfaces eingebunden. Dies bietet die Möglichkeit, den Reasoner zu wechseln.

3.2.3. HermiT

Die Kriterien zur Auswahl einer Inferenzmaschine sahen vor, dass der Reasoner von der OWL API unterstützt werden musste. Der Reasoner musste OWL Version 2 unterstützen und sollte nach Möglichkeit Java 1.6 oder neuer unterstützen. Von den verschiedenen Inferenzmaschinen, die zur Auswahl standen, wurde HermiT [Dep12] gewählt.

Er wird von der Information Systems Group der Universität Oxford entwickelt. Er ist der erste öffentlich verfügbare OWL Reasoner mit einer „hypertableau“ Rechenmethode. Dies führt angeblich zu einer erhöhten Effizienz bei der Inferenzberechnung. Er erfüllt sämtliche Kompabilitätstests für die OWL 2 Referenz und implementiert das Interface der OWL API.

Als direkte Alternative ist der Fact++ [Uni12a] der Uni Manchester zu nennen. Es konnten im Vergleich zum HermiT keine Unterschiede in der Berechnungseffizienz bemerkt werden. Fact++ ist in C++ implementiert und kommuniziert über einen Wrapper mit der OWL API. Um mögliche Fehlerquellen zu minimieren, wurde deswegen der in Java implementierte HermiT ausgewählt.

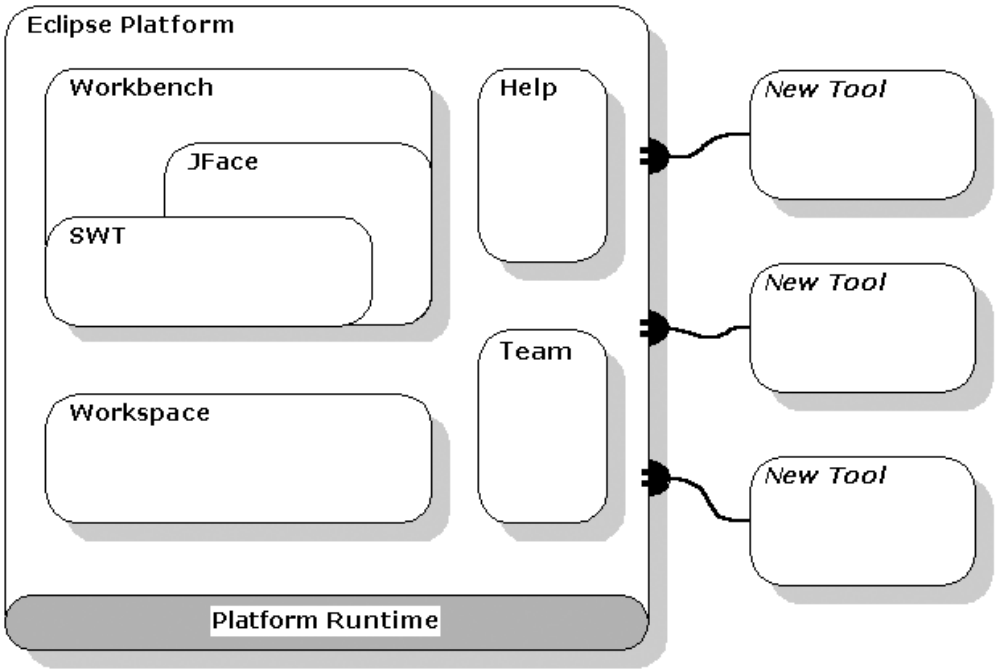


Abbildung 3.2.: Architektur der Eclipse RCP, [Ecl12]

3.2.4. Eclipse RCP

Bei Eclipse [Ecl12] handelt es sich um ein quelloffenes Programmierwerkzeug. Eclipse bietet eine Rich Client Platform (RCP), die es Entwicklern ermöglicht auf dem Eclipse Framework basierend von der IDE unabhängige Anwendungen zu entwickeln. Die Architektur der Eclipse RCP wird in Abbildung 3.2 präsentiert. Sie werden durch Kombinieren mehrerer Komponenten, die als Plugins bezeichnet werden, aufgebaut. Es können verschiedene Versionen eines Plugins nebeneinander von einer Eclipse RCP genutzt werden. Außerdem können Plugins auch von weiteren Eclipse RCPs genutzt werden. Die Eclipse RCP bietet dem Entwickler standardisierte Schnittstellen für Hilfe und Teamarbeit. Die grafischen Komponenten einer Eclipse RCP werden über Standard Widget Toolkit (SWT) und JFace angesprochen.

SWT ist ein quelloffener Werkzeugsatz für Java. Er wurde entworfen um effizienten, übertragbaren Zugriff, auf die Benutzerobeflächenausstattung des Betriebsystem, bereit stellen zu können.

Diese Funktionalitäten der Eclipse RCP ermöglichen einfache Weiterentwicklung und Anpassung eines Projektes. Da Eclipse in Java entwickelt wurde läuft es Platformunabhängig auf allen Systemen die Java unterstützen. Diese Vorteile waren ausschlaggebend dafür die Komponenten des REC als Plugins einer Eclipse RCP zu entwickeln.

Damit ein Plugin von der Eclipse RCP verwendet werden kann muss es einen zuvor definierten „Extension Point“ implementieren. Bei den Extension Points handelt es sich um Schnittstellen, die Plugins die Funktionalität mitteilen. Der REC hat zwei Extension Points definiert, die in Kapitel 5 erklärt werden.

4. Lösungsansatz

In Kapitel 2 wurde bereits erwähnt, dass die Gewinnung von Anforderungen mit einigen Problemen behaftet ist. Angefangen damit, dass Anforderungen von einem unstrukturiertem Format, wie Text, in eine strukturierte Form (Programme, Modelle, usw.) gebracht werden müssen, bis hin zu Unklarheiten (siehe Abb. 4.1) und Widersprüchen in den Anforderungen selbst. Um den menschlichen Aufwand zu reduzieren, sollten automatisiert Lösungen für diese Probleme gefunden werden.

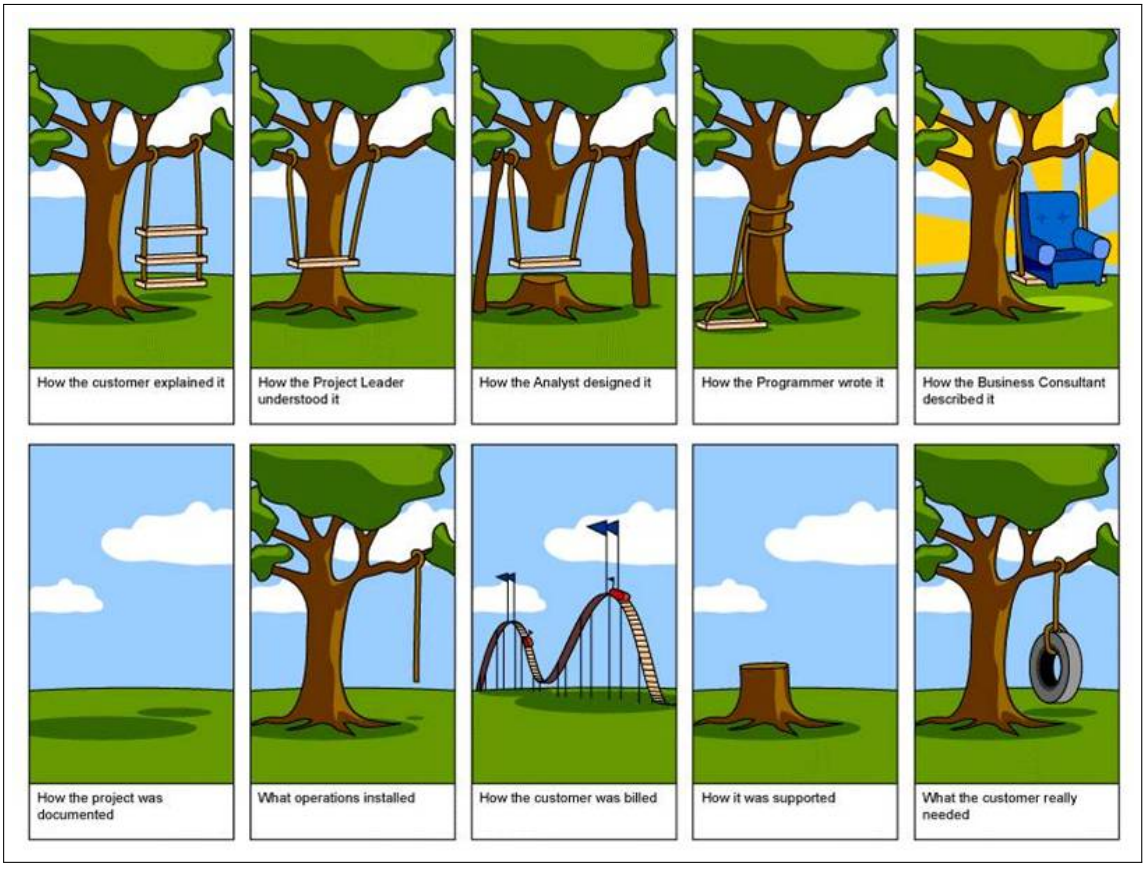


Abbildung 4.1.: Missverständnisse bei der Anforderungserhebung [Spe12]

4.1. Ausgangslage

Die Anforderungsanalyse benötigt Tage, Wochen oder sogar Monate bis sie von Analysten abgeschlossen ist. Heutzutage ist jedoch Zeit eine der wichtigsten Ressourcen. Diesen Prozess zu beschleunigen bedeutet, dass mehr Zeit für andere Aufgaben in einem Projekt bleibt, oder dass das Projekt schneller abgeschlossen werden kann. Da ein Mensch Anforderungen zuerst lesen muss, besteht kaum Spielraum die Analyse, durch eine Prozessoptimierung der menschlichen Komponente, zu beschleunigen. Ein Computerprogramm ist jedoch in der Lage Texte schneller zu verarbeiten als ein Mensch. Das große Problem, dass ein Computer mit Texten hat, ist das Verständnis des Inhaltes. Gesucht wird also eine automatisierte Lösung, die zuverlässig und schnell die Semantik eines Textes extrahiert.

Selbst wenn man davon ausgeht, dass eine automatisierte Lösung nicht ausreicht, um einen qualifizierten Analysten zu ersetzen, bietet sie Vorteile. Die Fähigkeit, Texte schneller zu verarbeiten als der menschliche Analyst, kann genutzt werden um Texte für den Menschen vorzubereiten. Ein System, dass bei 30% der Sätze zuverlässig ausschließen kann, dass es sich um Anforderungen handelt, sorgt dafür, dass ein Analyst noch maximal 70% des Textes lesen muss. Geht man weiter davon aus, dass ein solches System 20% der Sätze korrekt als Anforderungen identifiziert, so bedeutet das, dass die Teile des Textes, die der Analyst bearbeiten muss, nur noch 50% des ursprünglichen Dokumentes ausmachen. Und dadurch die Zeit, die der Analyst mit dem Text beschäftigt ist, reduziert wird.

Es lässt sich aber nicht nur die Nutzung der Zeit optimieren. Ein solches System kann genutzt werden, um bereits beim Erheben der Anforderungen die Qualität an mehreren Stellen zu verbessern. Fehler durch Mehrdeutigkeiten oder Widersprüche in den Anforderungen kann ein Computerprogramm objektiv betrachten. Ein automatisiertes Vorgehen vermeidet, beziehungsweise reduziert solche Fehler bereits frühzeitig. Die verarbeiteten Sätze würden bereits eine gewisse Struktur bekommen. Je höher der Automatisierungsgrad, desto mehr können weitere Systeme eingreifen und Anforderungen auf Konsistenz oder Widersprüche überprüfen.

All dies trägt dazu bei, dass eine automatisierte Lösung für die Erhebung von Anforderungen wünschenswert ist.

4.1.1. Anforderungsanalyse

In dieser Arbeit wird deswegen versucht ein Konzept zu entwickeln, welches Anforderungen automatisiert extrahieren kann. Es ist nun erforderlich zunächst zu identifizieren wie Anforderungen vorliegen. Betrachtet werden dabei zwei Punkte:

1. Wie sieht eine Anforderung aus?
2. Welche Bedeutung hat sie?

Anforderungen liegen implizit oder explizit in gewichteter Form vor. Die Unterscheidung kann offensichtlich sein, zum Beispiel eine Anforderung, die explizit optional ist.

> The windscreen wiper may have a rain sensor.

Diese Anforderung besagt, dass sie eintreten kann, aber nicht muss.

Es ist aber auch denkbar, dass eine Anforderung zwingend erfüllt werden muss, sie aber Teil eines optionalen Subsystems ist. Sie kann ignoriert werden, wenn das Subsystem weggelassen wird.

> If a rain sensor is installed, the windscreen wiper must be activated once the sensor registers rain.

Es ist klar, dass sie zwingend erfüllt werden muss, sofern ein Regensensor vorliegt. Ansonsten ist sie unerfüllbar.

Eine dritte Möglichkeit für Gewichtung verdeutlicht dieses Beispiel.

> The windscreen wiper should have a rain sensor.

Das Verb „should“, sollte, deutet darauf hin, dass es eine Anforderung ist, die man zur Not weglassen kann. Es ist nicht eindeutig, ob es sich um eine optionale Anforderung handelt oder nicht.

All dies sind Beispiele dafür, warum es möglich sein sollte, Anforderungen nicht nur zu identifizieren, sondern sie auch zu klassifizieren. Ein Auswertung durch Klassifizierung ermöglicht eine gezielte Weiterverarbeitung der Daten.

4.1.2. Auswertung von Anforderungen

Die Auswertung mit Hilfe von Klassifizierungen erlaubt es weitere Informationen in die Daten aufzunehmen. In einer technischen Spezifikation eines großen Systems werden Anforderungen für viele Untersysteme beschrieben. Bei der Spezifikation eines Autos gibt es beispielsweise Anforderungen an die Lenkung, an den Antrieb, an Scheibenwischer und Fensterheber und viele weitere Untersysteme. Indem wir Anforderungen klassifizieren gruppieren wir die Anforderungen der einzelnen Untersysteme in Klassen unterteilen. Es wird also möglich alle Anforderungen zu einem bestimmten Bereich gezielt zu betrachten.

Für die Extraktion von Anforderungen werden drei Klassen unterschieden:

1. Die Klasse der 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.

Eine weitere denkbare Unterteilung ist die Klassifizierung nach Priorität. Wichtige Anforderungen müssen vor Optionalen implementiert werden. Je früher also klar ist, welche Anforderungen wie wichtig sind, desto besser kann die Entwicklung geplant werden.

Da die Gruppierung aber vom Umfeld der Spezifikation abhängig ist, handelt es sich um domänenspezifische Klassifizierungen. Sie erziehlen also in bestimmten Domänen sehr gute Ergebnisse, bieten aber keine Garantien in anderen Domänen. So wird bei einem Auto von „Lenkrad“, bei einem Flugzeug jedoch von „Ruder“ gesprochen. Ein Schema für Flugzeugen jedoch könnte das „Seitenruder“ und das „Höhenruder“ als Spezialfälle von „Ruder“ identifizieren.

4.2. Gewählte Lösung

Um das Bisherige in einem Satz zusammenzufassen:

> Ziel ist es unstrukturierte textuelle Anforderungen in eine strukturierte Form umzuwandeln, um eine computergestützte Weiterverarbeitung zu ermöglichen.

Bei der Präsentation des Lösungsansatzes für diese Aufgabe wird zuerst die Klassifizierung vorgestellt.

Bei der Klassifizierung sollen Anforderungen in Klassen unterteilt werden. Es werden Sätze einer besonderen Form auf Bestandteile analysiert und dann gruppiert. Die Extraktion der Anforderungen kann nun als Spezialfall der Klassifizierung betrachtet werden. Untersucht man, welche Besonderheiten einen Satz zu einer Anforderung werden lassen, so kann man Sätze nach diesen Besonderheiten klassifizieren. Die Klasse der Sätze, die alle Merkmale einer Anforderung besitzen, ist die hier gesuchte Klasse. Die Klassifizierung wird in dieser Arbeit von Ontologien (W3C OWL - Kapitel 3.2.2) übernommen. Sie wurden gewählt, da Ontologien eine einfache Form der Strukturierung bieten und weil sie erlauben, dass Schlussfolgerungen geschlossen werden.

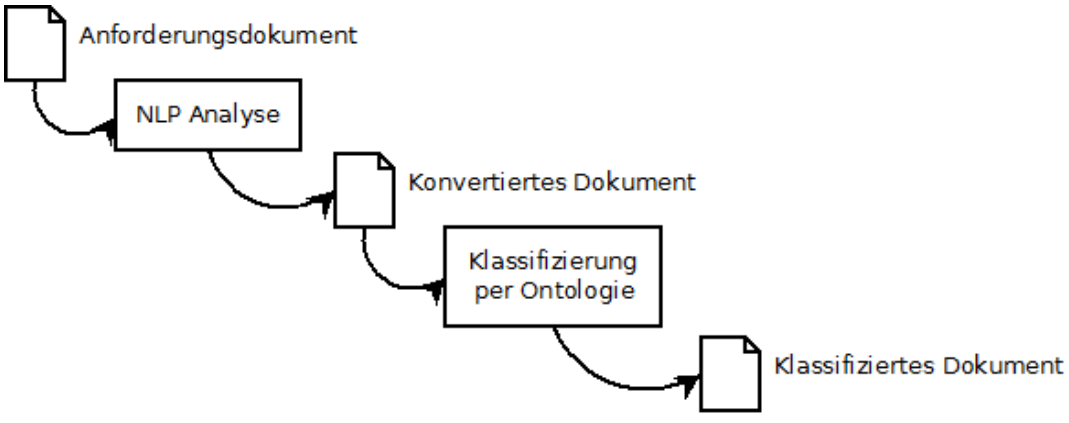


Abbildung 4.2.: Vorgehen dieser Arbeit bei der Klassifizierung einer technischen Spezifikation.

Daraus hat sich folgender Ablauf für die Extraktion von Anforderungen ergeben (siehe auch Abbildung 4.2): Zuerst liefert ein NLP Programm (Kapitel 2.3) die Struktur der Sätze und Wörter. Insbesondere die POS Analyse und die Lemmatisierung der Wörter werden von der Extraktionsontologie verwendet. Anschließen können die Anforderungen identifiziert werden. Um dies zu bewerkstelligen, muss identifiziert werden, was eine Anforderung für Eigenschaften hat. Anforderungen haben ein Subjekt, das Subjekt beeinflusst ein Objekt. Sätze, die ein (Subjekt, Verb, Objekt)-Tripel (SVO-Tripel) enthalten, sind also Kandidaten für Anforderungen. Sie werden deswegen in dieser Arbeit als „Anforderungskandidaten“ bezeichnet. Eine Suche nach bestimmten Schlüsselwörtern liefert nun aus den Anforderungskandidaten solche, bei denen es sich um Anforderungen handelt. Schlüsselwörter wurden anfangs möglichst restriktiv gewählt, um sie anschließend zu erweitern. Dabei wurde versucht eine möglichst hohe Erkennungsrate zu erzielen, ohne jedoch Sätze fälschlicher Weise als Anforderungen zu klassifizieren. Da die Schlüsselwörter in einer bestimmten Sprache sind, muss hier erwähnt werden, dass nur Dokumente in Englisch klassifiziert werden können.

Da es sich bei der Extraktion von Anforderungen um einen Spezialfall der Klassifizierung handelt, basiert sie auf demselben Vorgehen. Die mit NLP analysierten Sätze werden in eine Ontologie eingefügt, damit diese per Inferenz die Anforderungen klassifiziert.

4.3. Alternative Lösungsansätze

Ein erster Ansatz sah vor, dass die per NLP Analyse strukturierten Sätze programmatisch in Anforderungen und Nicht-Anfoderungen unterteilt werden. Dabei hätte ein Modul im Programm die Sätze nach SVO-Tripel und Schlüsselwörtern untersucht. Unter dem Aspekt der Flexibilität wurde dieser Ansatz jedoch verworfen. Um die Anforderungsextraktionsontologie anzupassen reicht es eine XML Datei zu verändern. Vergleicht man das mit dem Aufwand, ein Programm umzuschreiben, so ist die größere Flexibilität offensichtlich.

Der von Cleland-Huang et. al. [CHSZS06] entworfene Klassifizierer nutzt Trainingsmengen um Indikatorausdrücke zu erlernen. Diese Ausdrücke lassen sich mit den SVO-Tripel und den Schlüsselwörtern vergleichen. Sie nutzen im Grunde genommen also Lernalgorithmen, um bessere Schlüsselwörter zu finden. Daraus entstand der Gedanke verstärkt mit Hilfe von „Künstlicher Intelligenz“ die Extraktion zu bewerkstelligen. Nach [CHSZS06] und [GK11] erzielt Text Mining bessere Ergebnisse als NLP Analyse. Als großen Nachteil muss allerdings aufgeführt werden, dass für alle Ansätze die Lernalgorithmen nutzen Trainingsmenge benötigt werden. Die resultierenden Lösungen sind in der Regel sehr spezifisch für ein bestimmtes Problem.

5. Implementierung

Nachdem in Kapitel 4 der allgemeine Lösungsansatz dargelegt wurde, wird die Implementierung des Programmes vorgestellt. Das REC Werkzeug wurde als Java Programm entworfen. Den Kern bildet eine Eclipse RCP, in den die einzelnen Komponenten als Plugins eingebunden werden. Eclipse und alle verwendeten Programme sollten neben Windows auch unter Linux und Mac ausgeführt werden können, so dass auch der REC auf diesen Plattformen verwendet werden kann.

Zunächst wird die entworfene Datenstruktur und der Aufbau der gespeicherten Dokumente erklärt. Anschließend werden, dem Programmablauf folgend, die einzelnen Komponenten erläutert.

5.1. Datenstruktur

Eine Datenstruktur für Anforderungsdokumente, die persistente Lagerung unterstützt, war die erste Komponente des REC. Die Findung dieser Datenstruktur präsentierte sich mit einigen Problemen. So werden Dokumente als Textdateien eingelesen und als String vom Stanford CoreNLP verarbeitet. Als Ausgabe liefert der Stanford CoreNLP XML Dokumente. Die Ausgabe des Satzes „Stanford University is located in California. It is a great university.“ kann man im Anhang A.2.1 betrachten. Stanford CoreNLP speichert die Baumstruktur des Dokumentes, in diesem Fall mit zwei Sätzen, in der Datei ab. Jedes Wort beinhaltet als Blatt des Baumes die extrahierten Informationen der NLP Analyse.

Ontologien arbeiten auch mit XML Dokumenten, allerdings mit einer komplett unterschiedlichen Struktur. Die XML Repräsentation der in Abbildung 5.1 findet sich unter A.2.2. Es

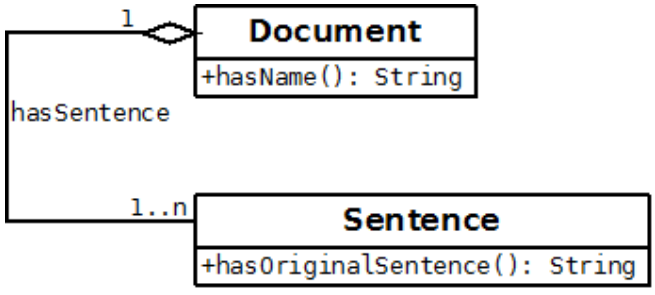


Abbildung 5.1.: Beispiel einer OWL Ontologie für Textdokumente

wird jede Klasse, jede Eigenschaft, jedes Individuum und jede Beziehung zwischen den Individuen einzeln gespeichert.

Es musste nun also eine interne Repräsentation gefunden werden, die beide Modelle in sich vereint. Das resultierende Datenmodell orientiert sich stark an der XML des Stanford CoreNLP. Für die Inferenzberechnung wird dann das Modell zur Laufzeit, in eine Ontologie übertragen und abschließend wieder in das interne Format umgewandelt. Daraus resultierte eine Basisontologie, auf der jede Klassifizierungsontologie aufbauen muss. Andernfalls funktioniert die Umwandlung nicht. Die Ontologie wird in 5.2.2 erklärt. Zunächst wird das interne Dokumentenmodell im Detail präsentiert.

5.1.1. Internes Dokumentenmodell

Der REC verwaltet jedes eingelesene Dokument auch intern als Dokument. Abbildung 5.2 verdeutlicht in einer reduzierten UML Darstellung die Klassenstruktur des Modells. Die Methoden wurden in der Darstellung gespart. Ein Dokument beinhaltet neben dem

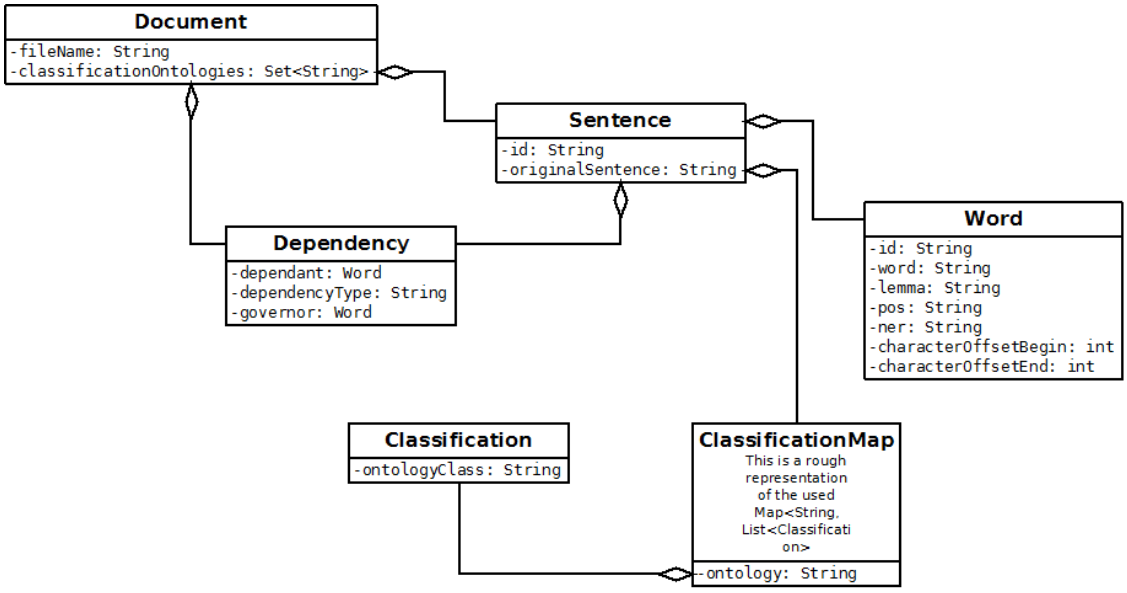


Abbildung 5.2.: Das Klassendiagramm des REC Dokumentenmodells

Dateinamen, eine Liste der Sätze eine Liste der Koreferenzen und die Ontologien mit der das Dokument bereits klassifiziert wurde. Bei den Koreferenzen handelt es sich um Abhängigkeiten zwischen verschiedenen Wörtern der Sätze.

> The plane must have 120 passenger seats. It muss have additional seating space for 12 crew members.

Die, zu diesen Sätzen gehörende, Koreferenz wäre „(plane, it)“. Damit wird verdeutlicht, dass es sich bei „it“ um eine weitere Instanz von „plane“ handelt.

Jedes Dokument speichert die IRI aller Ontologien mit denen es bereits klassifiziert wurde. Damit wird die Zuordnung der Sätze zu bereits angewendeten Klassifizierungen garantiert. Es kommt insbesondere bei der Umwandlung eines Dokumentes in eine Ontologie und die Rückumwandlung zu tragen.

Jeder Satz verwaltet eine Kopie des originalen Textes, eine eindeutige ID, eine Liste der enthaltenen Wörter, die Abhängigkeiten zwischen den Wörtern und die Klassifizierungen. Die ID der Sätze wird gebildet aus Dokumentenname und Position des Satzes im Text. Die Abhängigkeiten zwischen den Wörtern werden vom selben Konzept verwaltet wie auch die Koreferenz. Es liegt immer eine Beziehung „governor“, „dependant“ und die Art der Abhängigkeit vor. Die Klassifizierung liegt als Zuordnung von Klassifizierungsname, die IRI der angewendeten Ontologie und eine Liste mit den zugeordneten Klassifizierungen.

Bei den Klassifizierungen handelt es sich um die abgekürzten IRIs der Klassen, denen ein Satz zugeordnet wurde. Der Satz „The car must have four wheels.“ hätte die in Abbildung 5.3 dargestellte Klassifizierung vorliegen, nachdem die Anforderungsextraktion den Satz analysiert hat.

Key	http://www.rec.iris.uni-stuttgart.de/RequirementExtractionOntology.owl	
	ontologyClass	Candidate
	ontologyClass	Requirement

Abbildung 5.3.: Beispiel für Klassifizierung eines Satzes im REC.

Betrachtet man die Wörter eines Satzes, so enthält jedes Wort eine eindeutige ID, das ursprüngliche Wort und einige vom Stanford CoreNLP ermittelte Informationen. Die vom Stanford CoreNLP übernommenen Attribute sind das Lemma, POS, Erkenner von benannten Entitäten (Named-entity recognition) (NER) und Position im Satz, festgehalten durch die Positionen des ersten und letzten Buchstabens.

5.1.2. Der Aufbau eines REC-Dokumentes

Um Dokumente später sinnvoll weiter verarbeiten zu können, ist eine persistente Datenhaltung nötig. Der REC sollte also in der Lage sein, Dokumente die von ihm verarbeitet wurden, auch zu speichern. Als Dateiformat wurde aufgrund der Vielseitigkeit und der allgemeinen Akzeptanz XML gewählt. Java bietet mit Java Architecture for XML Binding (JAXB) eine Möglichkeit Klassen zu annotieren, um sie dann als XML Dokumente abzuspeichern. Dieser Vorgang wird als „marshall“ bezeichnet. Es geschieht eine 1:1 Zuordnung zwischen dem internen Datenmodell und der XML Datei. Eine Klassifizierung des Satzes „The airplane must have two wings.“ kann sich unter A.2.3 angeschaut werden.

JAXB übernimmt auch das Laden von bereits gespeicherten Dokumenten was als „unmarshall“ bezeichnet wird.

5.2. Programablauf

Klassifizierung mit dem REC läuft in zwei Schritten: Zuerst muss mit dem Stanford CoreNLP ein Dokument in das „.rec“ Fomat überführt werden. Im zweiten Schritt wird mit einer Ontologie das REC Dokument klassifiziert.

5.2.1. Dokumentenimport

Der erste Schritt stellt den Import von Dokumenten dar. Um die Vorteile der Eclipse RCP Architektur auszunutzen, wurde ein Import Modul erstellt. Der Extension Point dieses Moduls verlangt ein Interface mit zwei Methoden. Die „import“ Methode hat den Pfad des zu importierenden Dokumentes als Eingabe. Sie liefert als Ausgabe ein „Document“, das interne Modell für ein Dokument, zurück. Die zweite Methode gibt die erlaubten Dateiendungen als Array zurück. Ein Plugin, dass einen alternativen Importer bereitstellt, muss dieses Interface implementieren.

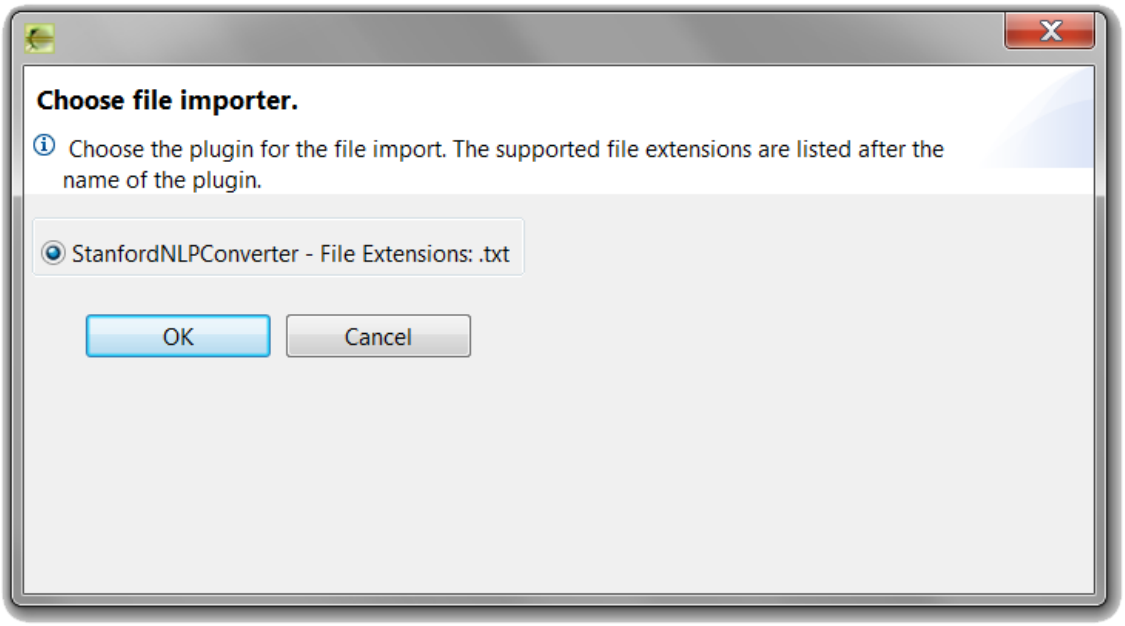


Abbildung 5.4.: Der Auswahldialog für Import Plugins

Für die Überführung durch den Stanford CoreNLP wird das Dokument, dass im „.txt“ Format vorliegen muss, als String eingelesen. Dieser String wird dem Stanford CoreNLP übergeben, welcher im Anschluss die Sprachverarbeitung übernimmt. Anschließend wird der Dokumentenbaum des Stanford CoreNLP in das REC Modell überführt.

Der Stanford CoreNLP hat beim Einlesen von Dokumenten einige Eigenschaften, die bedacht werden müssen, um optimale Resultate zu erzielen. Überschriften werden von ihm nicht als solche erkannt. Eine Überschrift ohne Satz beendende Satzzeichen werden zum nächsten

Satz hinzugefügt. Dies lässt sich beheben, in dem Überschriften entweder entfernt werden oder in dem Satz abschließende Zeichen hinzugefügt werden.

Eine weitere Eigenschaft des Stanford CoreNLP ist es, dass er Zeilenumbrüche und extra Leerzeichen ignoriert. Wenn bei einem Zeilenumbruch kein Leerzeichen nach dem letzten Wort steht, so wird er deswegen das Wort mit dem nächsten zu einem Wort zusammenfügen, was folgendes Beispiel verdeutlichen soll.

```
Leerzeichen am Ende
einer Zeile vergessen
führt zu Problemen
```

Dieser Satz würde vom Stanford CoreNLP zu folgendem Satz gemacht werden.

```
Leerzeichen am Endeeiner Zeile vergessenführt zu Problemen.
```

5.2.2. Klassifizierung

Die Klassifizierung erfolgt ebenfalls mittels eines Plugins. Das zu implementierende Interface hat nur eine Methode, die ein „Document“ sowohl als Eingabe, als auch als Ausgabe verwendet.

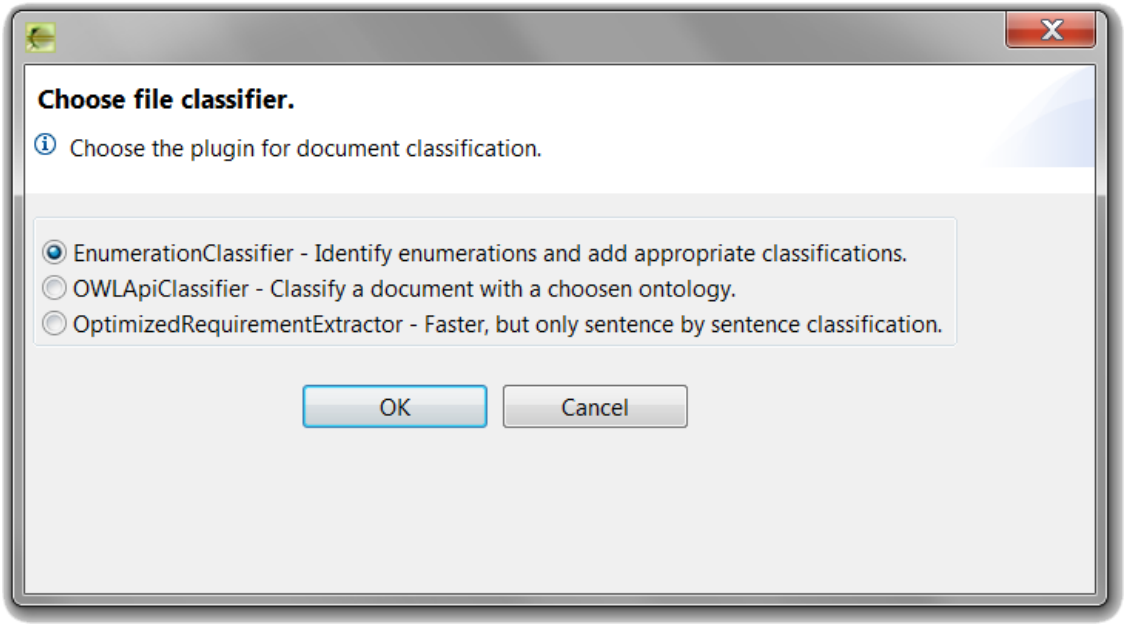


Abbildung 5.5.: Der Auswahldialog für Klassifizierungs Plugins

Wie bereits in Kapitel 5.1 angesprochen wird das interne Modell für die Berechnung der Inferenz in eine Ontologie umgewandelt. Dies führt leider zu einer Einschränkung der

unterstützten Ontologien. Jede Ontologie die für die Klassifizierung erstellt wird, muss auf der Basisontologie aufbauen. Es wird empfohlen, die bereitgestellten Hilfsmethoden für die Umwandlungen zwischen Dokumentenmodell und Ontologie zu verwenden.

Da der Fokus des REC auf der Klassifizierung von Sätzen liegt, werden auch nur Sätze ausgewertet. Nach der Inferenzberechnung werden alle Individuen, die Sätze sind, betrachtet. Alle weiteren Klassen, denen ein solches Individuum angehört, werden als Klassen, denen der Satz zu geordnet wurde, angesehen. Im Dokumentenmodell werden die Klassen eines Satzes unter der IRI der Ontologie abgespeichert. Dabei wird die abgekürzte IRI der Klasse verwendet.

Wenn ein Dokument mit einer Ontologie klassifiziert wird, mit der es bereits bearbeitet wurde, so werden alle Sätze den Klassen zugeordnet, mit denen sie von dieser Ontologie bereits klassifiziert wurden. Damit diese Informationen bei der Klassifizierung mit einer anderen Ontologie nicht verloren gehen, werden sie von der Basisontologie mit verwaltet.

Basisontologie

Die Basisontologie (Abbildung 5.6) repräsentiert das interne Dokumentenmodell. Die Bestandteile des Dokumentes werden in die Ontologie als Individuen eingefügt. Jede Klassifi-

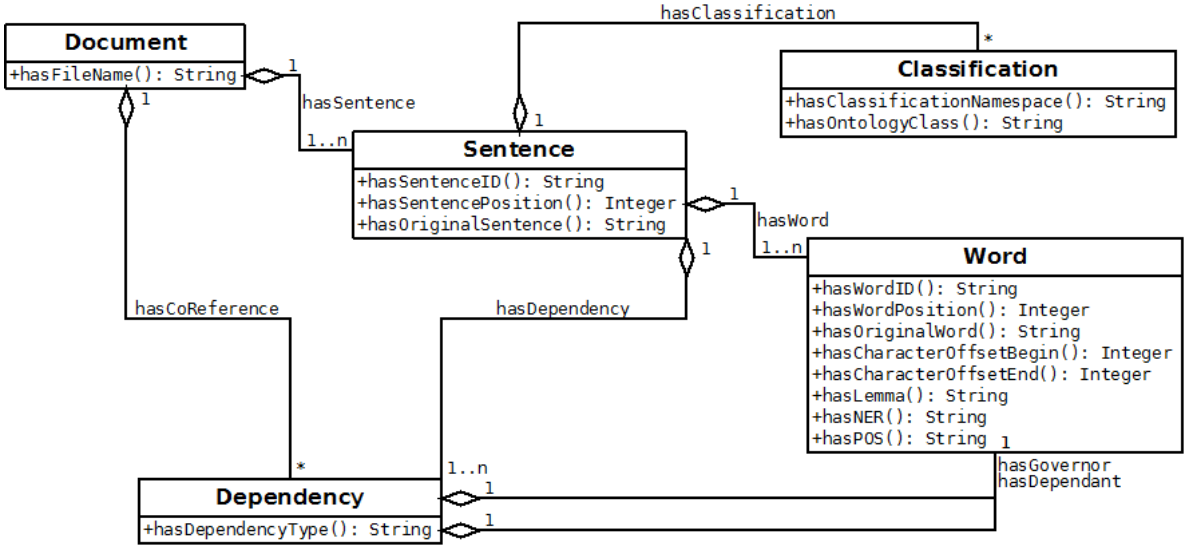


Abbildung 5.6.: Basisontologie des REC

zierungsontologie muss diese Basisontologie erweitern, um Kompatibilität zu gewährleisten. Die Basisontologie speichert unter anderem sämtliche Ontologien mit denen Sätze bereits klassifiziert wurden. Diese Metainformationen werden für die Klassifizierung nicht benötigt, ermöglichen es aber, dass ein klassifiziertes Dokument ohne weitere Informationen dargestellt werden kann. Ansonsten müsste jedem Dokument mitgegeben werden, wie die Ontologien, mit denen es klassifiziert wurde, angezeigt werden sollen.

Die Extraktionsontologie

Bei der Extraktionsontologie handelt es sich um eine Erweiterung der Basisontologie mit dem Fokus, Anforderungen zu erkennen. Wie in Kapitel 4.2 erklärt, besteht der erste Schritt darin, Anforderungskandidaten zu finden. Das sind Sätze, die SVO-Tripel enthalten.

SVO-Tripel Erkennung Damit die Ontologie diese Tripel erkennen kann, muss sie zuerst Verben, Subjekte und Objekte erkennen können (Abbildung 5.7).

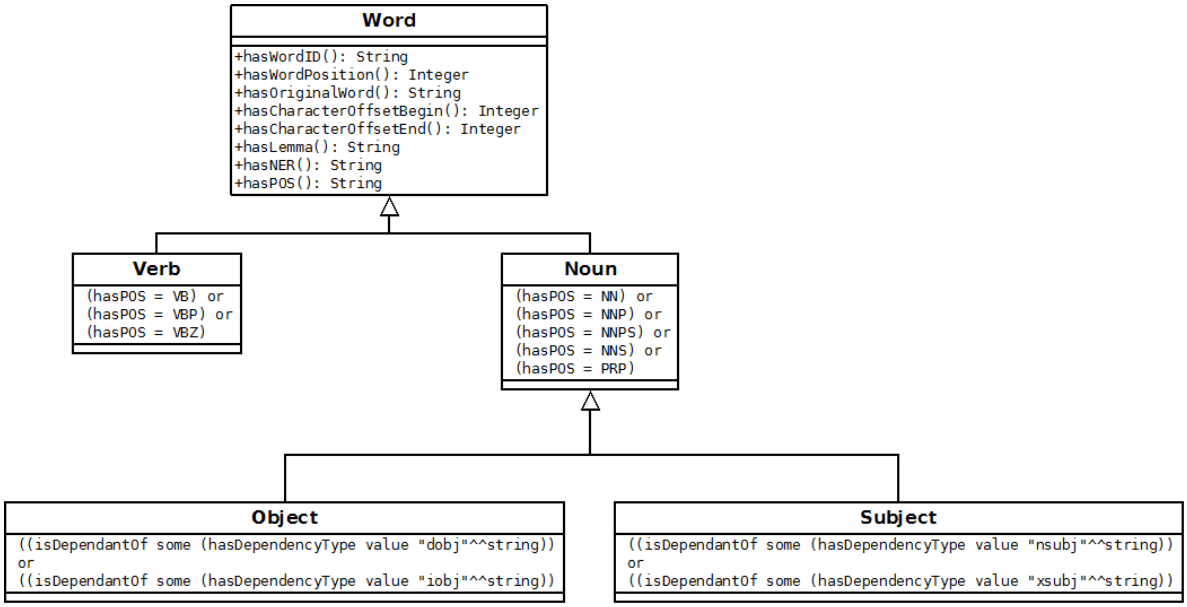


Abbildung 5.7.: Ontologieausschnitt zu SVO-Tripel

Die Erkennung von Verben liefert der Stanford CoreNLP über die POS Analyse. Verben sind Wörter die als POS „VB“, „VBP“ oder „VBZ“ haben. Damit werden Anforderungen auf Verben in der Gegenwart oder ihrer Grundform eingeschränkt.

Um Objekt und Subjekt zu erkennen, werden zunächst die Nomen und Personalpronomen erkannt. Die entsprechenden POS Tags können in der (Penn) Treebank [Uni90] nachgelesen werden. Aus den Nomen (und Personalpronomen) werden nun die Subjekte und Objekte mit Hilfe der Abhängigkeiten erkannt. Ein Objekt liegt vor, wenn ein Nomen als „Dependant“ in einer „dobj“ oder „iobj“ Beziehung vorliegt. Es muss das Nomen also ein indirektes, oder ein direktes Objekt sein. Ein Subjekt liegt vor, wenn ein Nomen also Dependant in einer „nsubj“ oder „xsubj“ Beziehung vorliegt. Wenn das Nomen also ein nominales oder ein kontrollierendes Subjekt ist.

Anforderungserkennung Damit aus den Kandidaten die Anforderungen extrahiert werden können, werden die Verben betrachtet (Abbildung 5.8). In der aktuellen Fassung werden Anforderungen in drei Klassen unterteilt, die sie durch die verwendeten Schlüsselwörter unterschieden. Als Schlüsselwörter werden die Lemma, also die Grundformen, der Verben betrachtet.

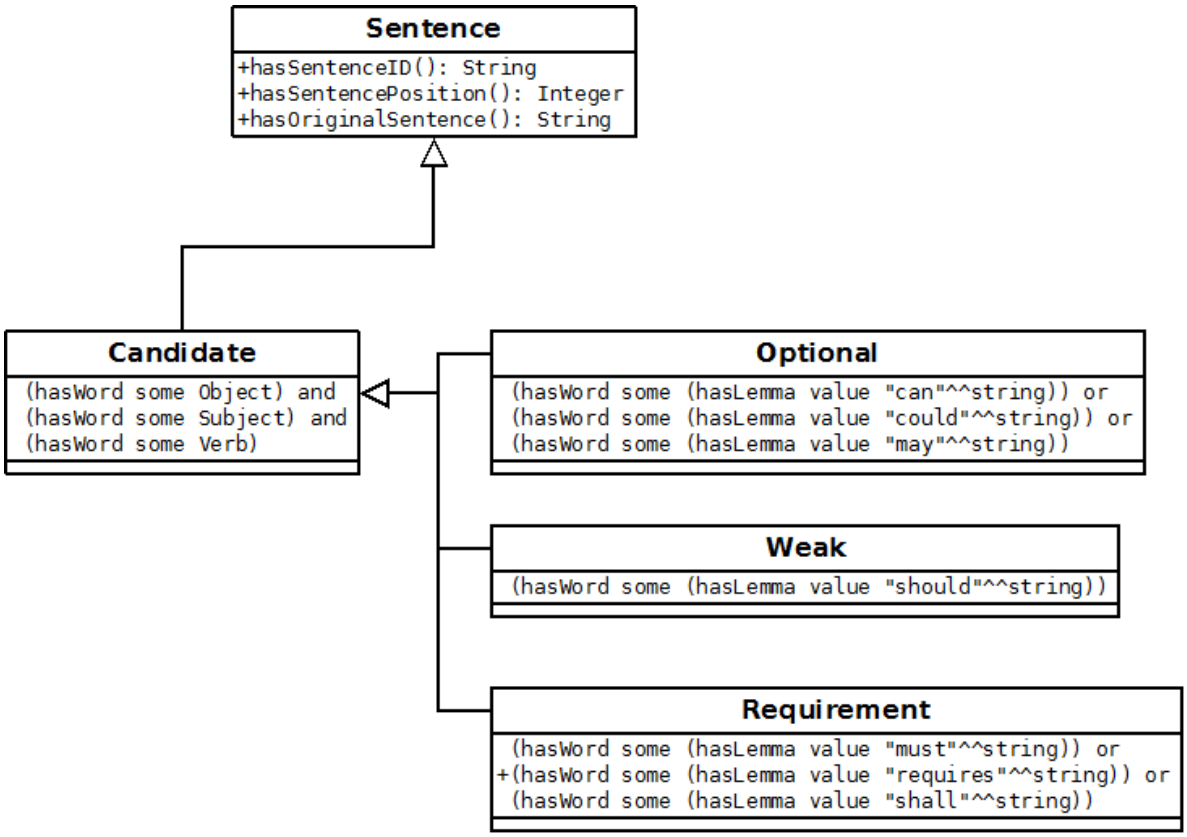


Abbildung 5.8.: Ontologieausschnitt zu Anforderungsunterscheidung

Die Klasse der optionalen Anforderungen liegt vor, wenn bei einem Kandidaten eines der Verben, „can“, „could“ oder „may“ vorliegt.

Um eine schwache Anforderung handelt es sich, wenn es sich um das Verb „should“ handelt.

Als echte Anforderungen werden solche betrachtet, bei denen die Verben, „must“, „requires“ oder „shall“ enthalten sind.

Um die Klassen zu erweitern müssen zusätzliche Schlüsselwörter der Ontologie hinzugefügt werden.

Inferenz

Danach werden mit dem Reasoner die Inferenzen berechnet. Betrachtet werden dabei Klassenaussagen (Class Assertions) und die Dateneigenschaftsaussagen (Data Property Assertions). Das bedeutet, dass der Reasoner versucht aus den gegebenen Klassen und Dateneigenschaften neue Informationen zu beziehen. So zum Beispiel die Information, dass ein Wort ein Verb ist, wenn es als POS den Wert „VB“ hat.

Als Reasoner wird der HermiT Reasoner genutzt. Die Inferenzeberechnung ist der Teil der Klassifizierung, der die meiste Zeit benötigt, da der Reasoner für die Inferenzberechnung nicht nur jedes Wort für sich betrachtet, sonderen es mit jedem Anderem vergleicht. Es liegt ein Wachstum des Aufwands von $(Anzahl\ der\ Ontologielklassen)^n$, mit $n = Anzahl\ Wörter + Anzahl\ Sätze + 1$, vor.

5.2.3. Plugins für die Klassifizierung

Im Laufe der Entwicklung entstanden drei Plugins für die Klassifzierung. Sie werden für ein besseres Verständnis vorgestellt.

OWLApiClassifier

Der „OWLApiClassifier“ wandelt ganze Dokumente in eine einzige Ontologie um. Diese Ontologie wird anschließend dem Reasoner als Ganzes übergeben. Nachdem die Inferenzberechnungen abgeschlossen sind wird die Ontologie wieder in ein Dokument umgewandelt. Dies ist das Kernplugin für die Inferenz. Es klassifiziert zuverlässig Dokumente mit allen verfügbaren Informationen.

OptimizedRequirementExtractor

Der „OptimizedRequirementExtractor“ entstand, da der „OWLApiClassifier“ nicht die erhoffte Performanz erbrachte. Diese Variante des „OWLApiClassifier“ wandelt das eingehende Dokument Satzweise in einzelne Dokumente um. Da dem Reasoner somit nur Dokumente mit nur je einem Satz übergeben werden, gelingt die Klassifizierung schneller.

So wurde ein Dokument, das mit dem „OWLApiClassifier“ in 42 Minuten klassifiziert wurde, mit dem „OptimizedRequirementExtractor“ in nur 50 Sekunden ausgewertet. Das Wachstum dieses Plugins liegt „nur“ in:

$$n' * (Anzahl\ der\ Ontologielklassen)^k, \text{ mit } n' = Anzahl\ Sätze \\ \text{und } k = Anzahl\ der\ Wörter\ im\ längsten\ Satz$$

Der Performanzgewinn dieses Moduls führt zu einem Nachteil. Es können nur Ontologien verwendet werden, die keine dokumentenweite Informationen benötigen. Damit werden die

Koreferenzen von diesem Plugin nicht beachtet. Sobald alle für die Klassifzierung nötigen Daten in einem Satz liegen, bietet dieses Klassifzierungsplugin deutliche Vorteile in Sachen Geschwindigkeit.

EnumerationClassifier

Das letzte Klassifizierungsplugin löst ein strukturelles Problem mit technischen Spezifikationen. Anforderungen werden von Nutzern gerne als Aufzählungen (wie das nachfolgende Beispiel zeigt) aufgeschrieben.

> Ein Auto muss folgende Eigenschaften haben:
>
> - Vier Räder
> - Mindestens zwei Türen
> - Einen Motor

Diese Form der Aufzählung hat zwei Probleme. Erstens erkennt der Stanford CoreNLP die Aufzählungpunkte nicht als Sätze. Daraus folgt, dass sie an den Anfang des darauffolgenden Satzes geschrieben werden. Dies lässt sich umgehen, indem jeder Aufzählungspunkt mit einem Punkt beendet wird.

Das zweite Problem bezieht sich auf die Klassifizierung. Die in dieser Arbeit verwendete Ontologie benutzt aussagenlogische Ausdrücke für die Klassifizierung. Mit solchen Ausdrücken ist es nur möglich eine Zuordnung zwischen der Einleitung einer Aufzählung und den Aufzählungspunkten zu schaffen, wenn die Texte bereits semantische Informationen vorliegen haben.

Um Aufzählungen klassifizieren zu können und trotzdem nur ein Mindestmaß an Vorstrukturierung der Texte zu verlangen, wurde ein Plugin für die Klassifizierung von Aufzählungen entworfen. Um dessen Funktionsweise zu gewährleisten, muss eine Aufzählung folgende Form haben.

```
Ein Auto muss folgende Eigenschaften haben $§.
§$ Vier Räder.
§$ Mindestens zwei Türen.
§$ Einen Motor.
```

Die Regeln für eine Aufzählung lauten also:

- Die Aufzählung muss mit einem Satz eingeleitet werden, der mit „$§.“ aufhört.
- Jeder Aufzählungspunkt muss ein Satz sein, also mit einem Punkt, Ausrufezeichen oder Fragezeichen enden.
- Außerdem muss jeder Aufzählungspunkt mit einem „§$“ beginnen.

Wenn diese Regeln eingehalten werden, wendet das Plugin die Klassifizierungen des Einleitungssatzes auf jeden Aufzählungspunkt an. Handelt es sich beim einleitenden Satz um eine optionale Anforderung, so werden auch alle Aufzählungspunkte zu optionalen Anforderungen.

Starten lässt sich das Plugin über den „EnumerationClassifier“ (Abbildung 5.5).

5.2.4. Dokumentenansicht

Sobald ein Dokument in „.rec“-Format vorliegt, kann es vom REC Editor geöffnet werden. Abbildung 5.9 zeigt den REC mit drei geöffneten Dokumenten.

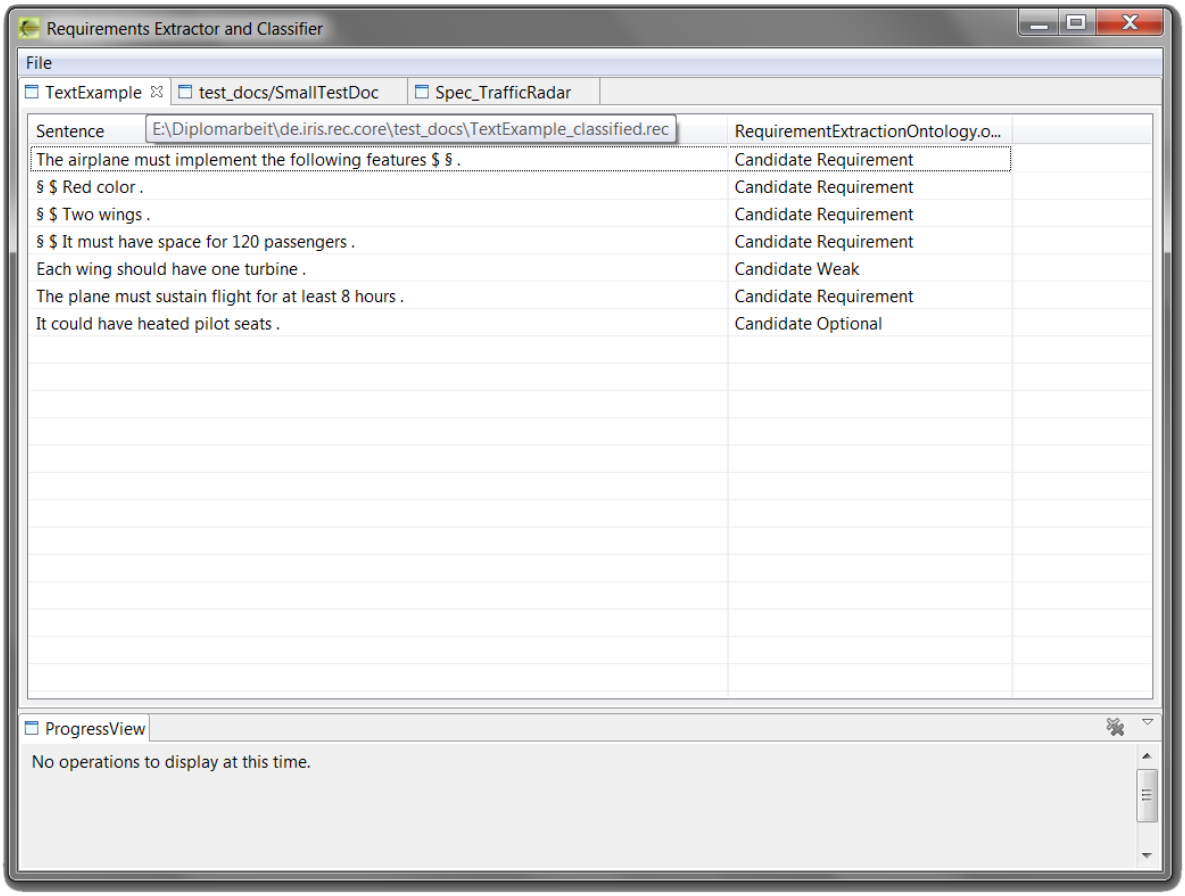


Abbildung 5.9.: Der Dokumentenbetrachter des REC

Für jedes einzelne Dokument wird ein eigener Reiter geöffnet. Jeder Reiter erhält den Namen des Dokumentes als Titel, und den Dateipfad als „Tooltip“. In der ersten Spalte werden die Sätze angezeigt, sie lassen sich durch klicken auf die „Sentence“ Schaltfläche in auf, oder absteigender Reihenfolge sortieren.

Danach wird eine Spalte pro angewendeter Klassifizierung eingeblendet. Der Titel ist die abgekürzte IRI der Ontologie, während als „Tooltip“ die komplette IRI eingeblendet wird. Es werden zu jedem Satz die Klassen angezeigt, zu denen der Satz gehört. Der jeweilige Spaltentitel dient als Schaltfläche für die Sortierung. Für die Sortierung werden die Klassen als String verwendet lexikalisch Sortiert. Dies wurde so gewählt, um ein einheitliches Sortierungsschema gewährleisten zu können, ungeachtet der verwendeten Klassifizierung.

6. Ergebnisse

Der REC wurde mit mehreren Spezifikationen der National Highway Traffic Safety Administration (NHTSA) getestet (Abbildung 6.1). Die NHTSA ist das Verkehrsministerium der USA.

Emergency Medical Technician: Basic Refresher Curriculum	141 Seiten
Research on Minimum Sound Specifications for Hybrid and Electric Vehicles	108 Seiten
Speed-Measuring Device Performance Specifications: Across-the-Road Radar Module	56 Seiten
Model Minimum Performance Specifications for Police Traffic Radar Devices	90 Seiten
Requirements for Trailer Manufacturers	42 Seiten

Abbildung 6.1.: Getestete NHTSA Spezifikationen

Die Metriken wurden von [CHSZS06] übernommen (siehe Abbildung 3.1). Überprüft wurde der REC auf Trefferquote „recall“ und Präzision „precision“.

Als Ziel für die Arbeit wurde eine Präzision von 100% und eine Trefferquote über 60% angestrebt. Dieses Ziel ergab sich aus folgender Überlegung. Geht man davon aus, dass es nicht möglich ist 100% der Anforderungen zuverlässig zu erkennen, so muss ein Nutzer eine Teilmenge des Textes manuell überarbeiten. Wenn das REC System garantieren kann, dass es sich bei gefundene Anforderungen nur um korrekt erkannte Anforderungen handelt, so können sie bei einer manuellen Überarbeitung ausgelassen werden. Vergleichbare Arbeiten erkennen 60 - 80% der Anforderungen, um also vergleichbar zu sein, muss der REC mindestens 60% der Anforderungen erkennen.

6.1. Testablauf

Der Testablauf (Abbildung 6.2) bestand darin ein Dokument durch den REC und von Hand zu klassifizieren. Danach wurden beide klassifizierten Dokumente verglichen.

Es wurde notiert, wie viele Anforderungen korrekt erkannt wurden. Dabei wurde auch eine Unterteilung in die verschiedenen Anforderungsklassen vorgenommen. Die fehlerhaft

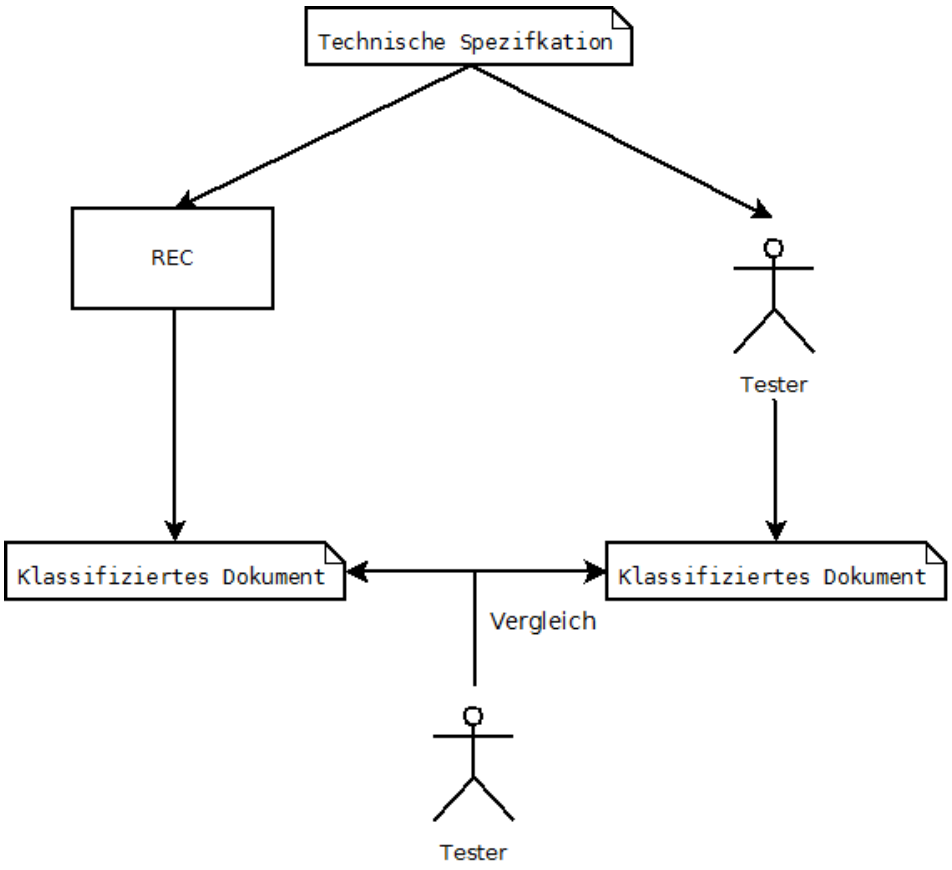


Abbildung 6.2.: Ablauf der REC Tests

identifizierten Anforderungen, sowie die inkorrekt als „Nicht"-Anforderungen klassifizierten Sätze, wurden ebenso festgehalten. Die Anzahl der korrekt als „Nicht"-Anforderung erkannten Sätze wurde durch die zuvor gesammelten Messdaten erhoben.

6.1.1. Probleme beim Testen

Ein erster Testdurchlauf identifizierte mehrere Probleme: Zum Einen bereitete die Formatierung der Texte dem Stanford CoreNLP Schwierigkeiten, zum Anderen hatte der REC eine niedrige Performanz.

Formatierung der Texte

Sollten ursprünglich noch unbearbeitete Dokumente als Eingabe verwendet werden, musste festgestellt werden, dass dies leider nicht möglich ist. Probleme haben mathematische Formeln und Überschriften in den Texten bereitet. So wurde die folgende Textstelle nicht als Anforderung erkannt, da die Überschrift zu einem Bestandteil des Satzes wurde.

```
51221.21 Speed Display Requirements
The speed display characteristics of display readability,
display speed lock control, display clear function, internal
circuit test function, speed display transfer# signal processing
channel sensitivity, target channel speed displays, patrol
channel speed displays, and auxiliary displays shall be tested in
accordance with 51221.79 and shall meet the following
requirements:
```

Nach Entfernung der Überschrift wurde dieser Satz korrekt als Anforderung klassifiziert. Der Stanford Core NLP hat auch mit diversen Encodingzeichen Probleme (siehe 5.2.1).

Die Dokumente mussten entsprechend aufbereitet werden. Dazu wurden in den Dokumenten die Abschnitte, welche die meisten Anforderungen enthielten, identifiziert. Die Sätze dieser Abschnitte wurden ohne Überschriften, Formeln oder Tabellen in neue Textdokumente überführt.

Dauer der Tests

Aufgrund eines derzeitig noch nicht identifizierten Problems dauert die Klassifizierung großer Dokumente unverhältnissmäßig lang.

So dauert die Klassifizierung des Satzes „Each type I, III, and V radar device shall be accompanied by a minimum of one tuning fork.“ in einem Dokument mit 4 Seiten 3757 ms, wird ein Dokument mit nur diesem einem Satz erzeugt, so dauert die klassifizierung nur 3 ms. Dabei handelt es sich um die optimierte Klassifizierungsengine (5.2.3), die Satzweise klassifiziert. Die Klassifizierung eines 30 Seiten Dokumentes brauchte damit über 3 Tage.

Um, ungeachtet dieses Fehlers, in der verfügbaren Zeit eine möglichst brauchbare Testreihe aufzubauen, wurden deswegen aus jeder Spezifikation 20 Sätze ausgewählt und klassifiziert. Zusätzlich wurden aus eine Spezifikation 5 Seiten gewählt und ausgewertet.

6.2. Testergebnisse

Insgesamt wurden 227 Sätze klassifiziert. Die Analyse von Hand ergab, dass 152 davon Anforderungen sind. Abbildung 6.3 verdeutlicht die Ergebnisse der Klassifizierung mit der Erkennungsontologie von Anforderungen grafisch.

Der REC erkannte 109 Anforderungen, 7 schwach formulierte Anforderungen und 10 optionale Anforderungen. Somit wurden 25 Anforderungen nicht als solche identifiziert. Dies führt zu einer Trefferquote von 80%. Keine der erkannten Anforderungen wurde fehlerhaft erkannt. Die Präzision des REC liegt bei 100%. Es liegt also eine Ausfallrate von 0% bei den technischen Spezifikationen vor. Die Ausfallrate, auch Falsch-Positiv-Rate, gibt den Anteil der fälschlich als positiv (auf Anforderung) klassifizierten Sätze an, die in Wirklichkeit keine Anforderungen (also negativ) sind.

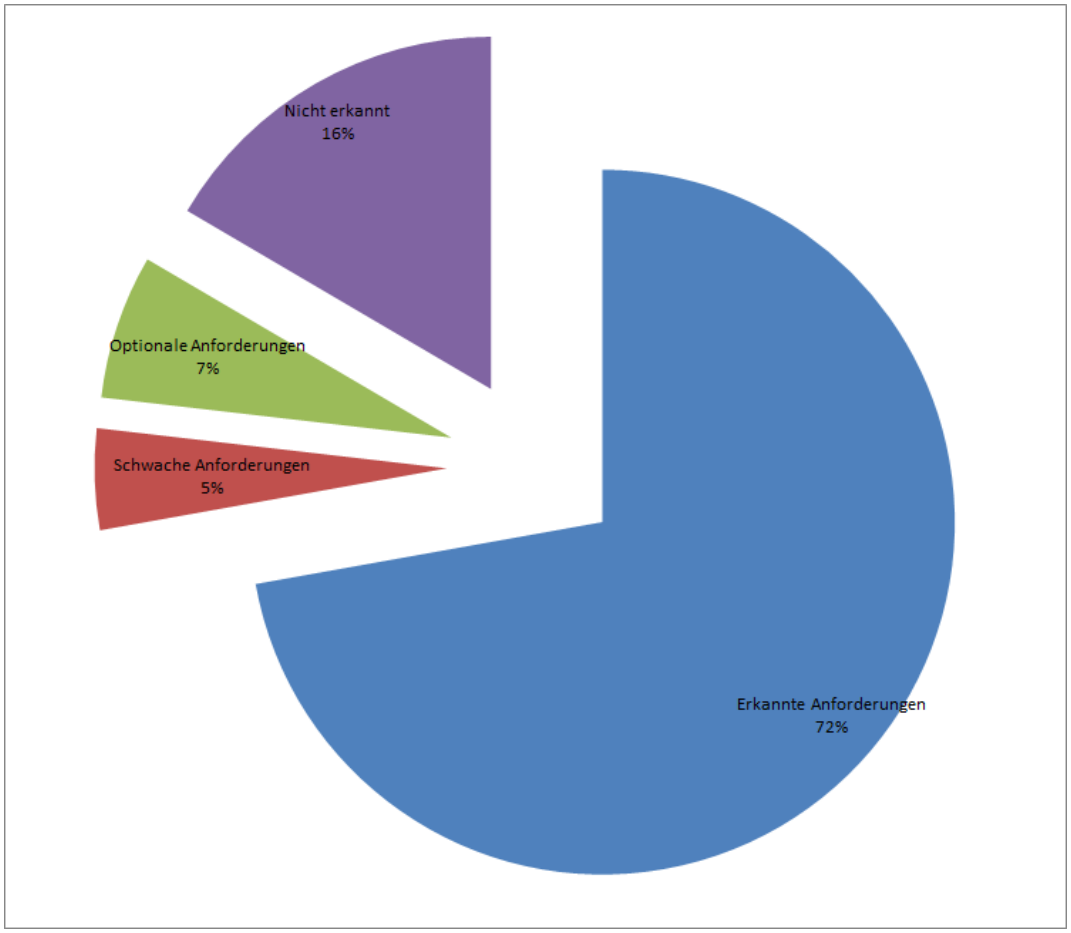


Abbildung 6.3.: Klassifizierung nach Anforderungsontologie.

6.2.1. Detailanalyse der Ergebnisse

Eine genauere Analyse der Dokumente und der Klassifzierungsergebnisse liefert ein besseres Bild über die Fähigkeiten des REC.

Wie Abbildung 6.4 zeigt, liegt die größte Fehlerrate beim „Emergency Medical Technician: Basic Refresher Curriculum“. Eine manuelle Analyse ergab, dass der Großteil der Anforderungen in diesem Dokument im Imperativ vorliegen. Die Klassifzierungsontologie des REC kann allerdings Imperative Aussagen nicht erkennen. Es verbleibt zu bemerken, dass es sich beim „Emergency Medical Technician: Basic Refresher Curriculum“ um einen Katalog von Anforderungen handelt, aber nicht um eine technische Spezifikation. In technischen Spezifikationen ist die Verwendung des Imperativ unüblich.

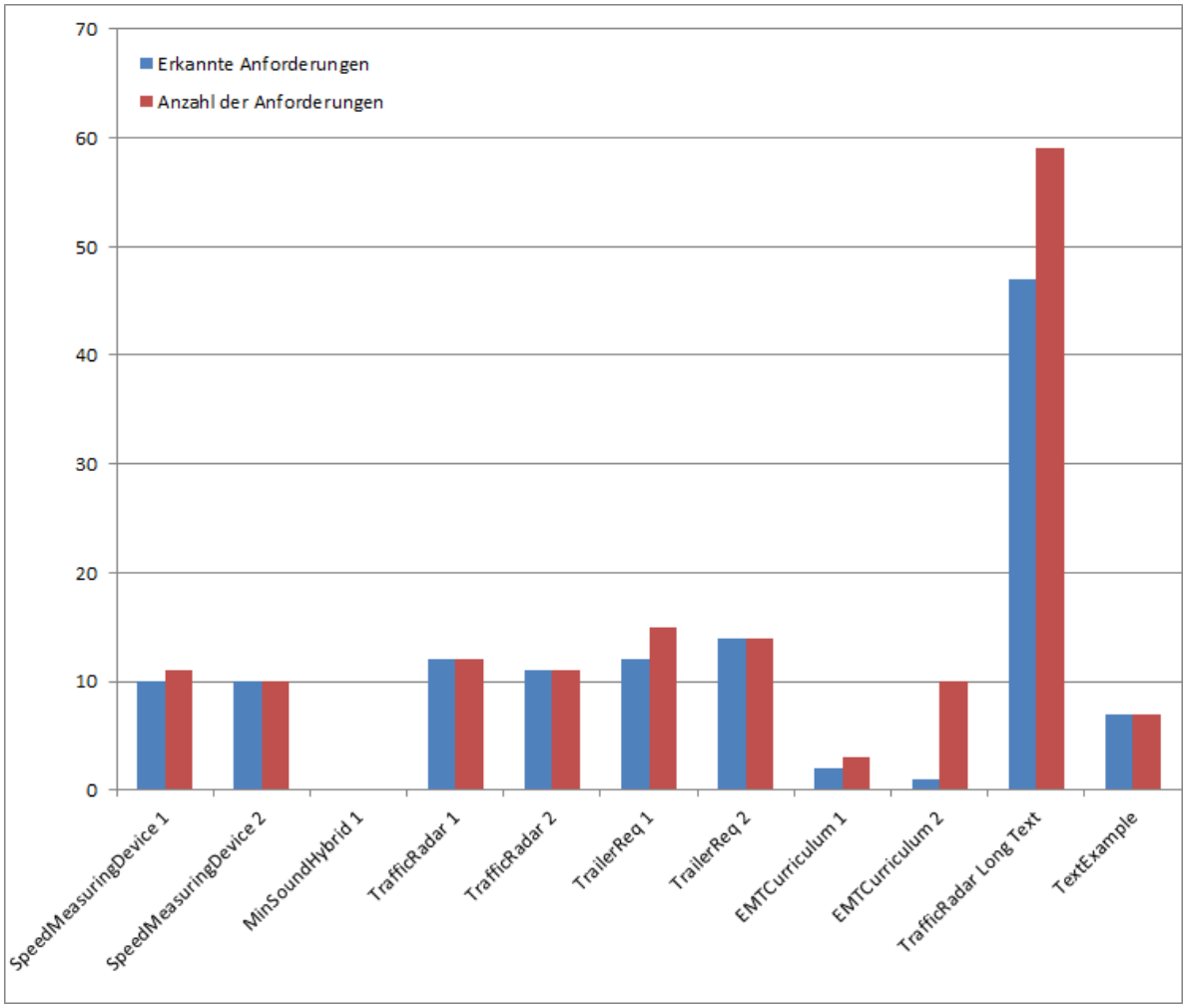


Abbildung 6.4.: Erkennung nach Dokumenten aufgeschlüsselt.

6.2.2. Analyse von sonstigen Textdokumenten

Im Zuge der Klassifizierung von Spezifikationen wurden auch andere Text Dokumente analysiert. Gewählt wurden die Funktionsbeschreibungen der Betriebssysteme Microsoft Windows und Ubuntu Linux. Abbildung 6.5 präsentiert die Ergebnisse dieser Analysen. Es ergab sich eine erhebliche Falscherkennung von Anforderungen, die in technischen Spezifikationen nicht vorlag. In der Tat lag die Präzision nur noch bei 50% anstatt bei den ursprünglichen 100%. Die falsch klassifizierten Sätze wurden alle, fälschlicherweise, in die Klasse der optionalen Anforderungen eingeordnet.

Grund hierfür ist, dass Sätze mit dem Verb „can“ als optionale Anfoderungen klassifiziert werden. Betrachtet man das Verb „können“ genauer, so wird klar, dass es mehrere Bedeutungen haben kann. Es beschreibt zum einerseits die Möglichkeit einer Tätigkeit.

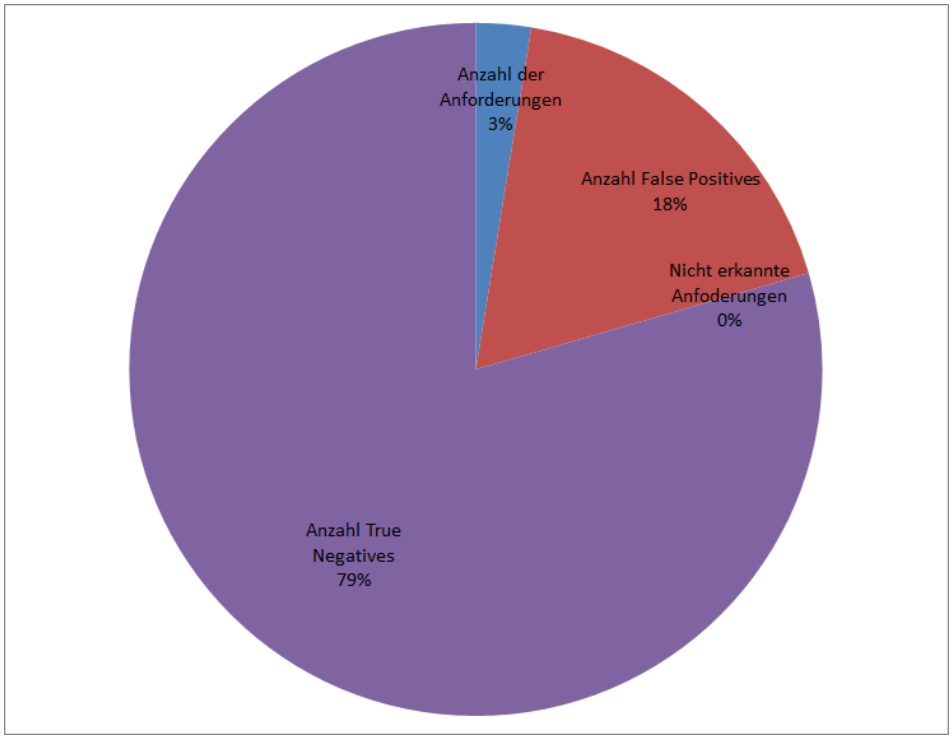


Abbildung 6.5.: Klassifizierung von Texten, die keine technischen Spezifikationen sind.

```
And when you preview an app in the Ubuntu Software Centre,
you can install it with just one more click.
```

Alternativ beschreibt es die Möglichkeit auf das Vorhandensein einer Sache.

```
The car can be of red color.
```

Während der erste Satz keine Anforderung darstellt, handelt es sich beim Zweiten um eine optionale Anforderungen. Interessanter Weise kamen in den analysierten technischen Spezifikationen keine Sätze der ersten Art vor. Nutzer des REC Systems sollten also optionale Anforderungen erneut überprüfen. Nur so können sie sicher stellen, dass keine Sätze falsch klassifiziert werden.

6.2.3. Klassifizierung mit anderen Ontologie

Der REC erlaubt es Texte mit anderen Ontologien zu klassifizieren. Auch die mehrfach Klassifizierung eines Dokumentes mit verschiedenen Ontologien wird unterstützt (siehe Abbildung 6.6). Wie in Kapitel 5.2.2 erwähnt müssen alle Ontologien die Basisontologie erweitern. Da bereits die Erkennung von Anforderungen mit einer Ontologie erfolgt, wurden nur minimale Tests vollzogen. Es sollte gewährleistet werden, dass keine kritischen programmatischen Fehler vorhanden sind.

Es wurden Ontologien erzeugt, die das Vorkommen bestimmter Wörter, zum Beispiel „temperature“ oder „frequency“, hervorheben sollten. Dabei wurden auch die Koreferenzen miteinbezogen.

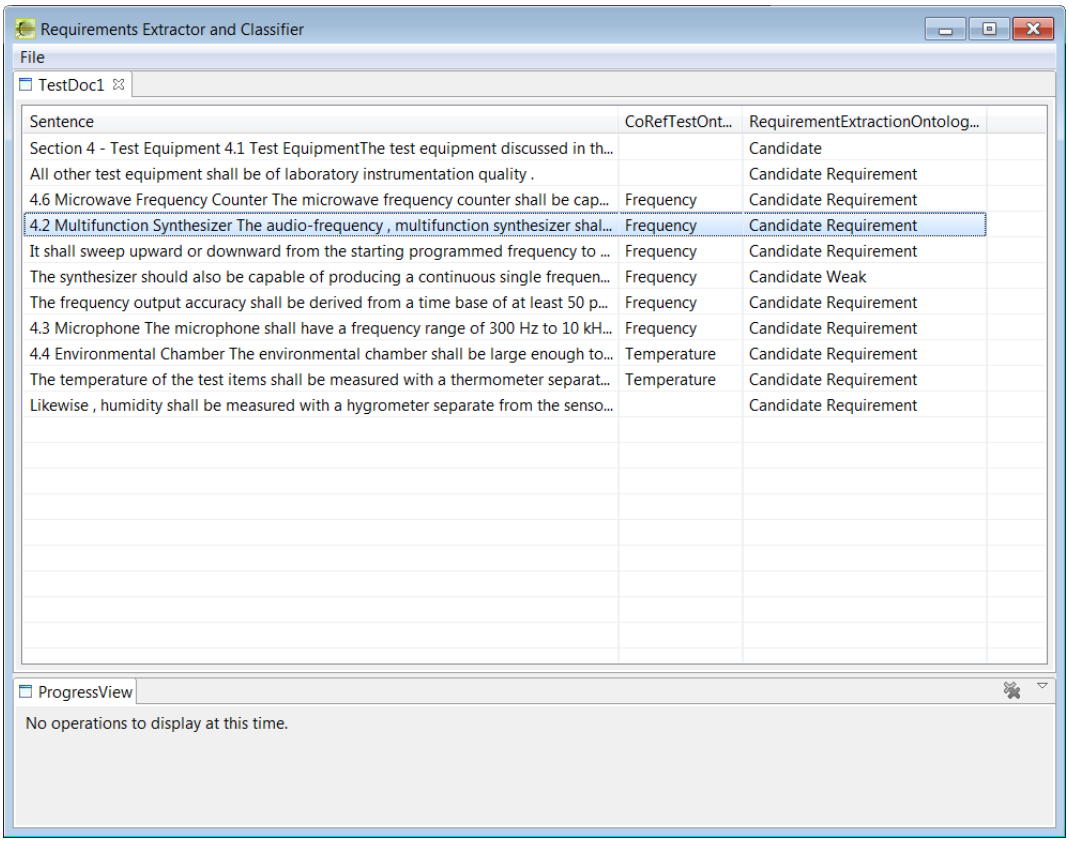


Abbildung 6.6.: Ein mit mehreren Ontologien klassifiziertes Dokument.

Die Tests zeigten, dass sich Dokumente ohne Wechselwirkung von mehreren Ontologien klassifizieren ließen. Desweiteren traten keine programmatischen Fehler zu Tage.

6.3. Zusammenfassung

Mit dem Requirement Extractor and Classifier (REC) System können Dokumente mit verschiedenen Ontologien klassifiziert werden. Für die Gewinnung von Anforderungen wurde eine Ontologie erzeugt, die bei technischen Spezifikationen eine Erkennungsrate von 70% lieferte. Zusätzlich lag die Präzision des REC bei 100%. Es wurde also kein Satz fälschlicherweise als Anforderung klassifiziert. Die Trefferquote ist stark von der Qualität des Textes abhängig. Anforderungen die im Imperativ vorliegen oder unvollständige Sätze eignen sich nicht für den REC. Auch müssen Auflistungen erst überarbeitet werden, bevor sie das System erkennen und klassifizieren kann. Da Überschriften vom Stanford CoreNLP nicht als Sätze

erkannt werden, stören sie die Erkennung von Anforderungen. Diese Störung kann behoben werden, indem jede Überschrift mit einem Satzendzeichen („.“, „!“ oder „?“) beendet wird.

Geht man von den technischen Spezifikationen weg, so liegen bei den optionalen Anforderungen eine beträchtliche Anzahl an falsch klassifizierten Sätzen vor. Dies führt zu einer Präzision von 50% bei allgemeinen Texten.

Die Klassifizierung mit anderen Ontologie erfolgt ohne Wechselwirkung und ohne bemerkte programmatische Fehler. Detailiertere Tests würden die Ontologien und nicht die Funktionsweise des REC überprüfen und wurden somit nicht durchgeführt.

7. Kontemplation

In einer abschließenden Reflexion wird in diesem Kapitel die vorliegende Arbeit in ihrer Vollständigkeit bewertet. Dazu werden in einem ersten Absatz erneut die größten Schwierigkeiten dieser Arbeit aufgeführt. Anschließend wird ein Fazit über diese Arbeit geschlossen. Den Abschluss bildet der Ausblick, in dem auf die Möglichkeiten eingegangen wird, mit denen diese Arbeit erweitert werden kann.

7.1. Hindernisse

Die vom Abschnitt der Analyse, bis hin zur Implementierung, aufgetretenen Probleme werden nun aufgeführt und untersucht werden. Die Überlegungen bei der Wahl der Architektur des REC werden dabei nicht nähe betrachtete.

Wahl der Extraktionsmethode Die beiden Extreme der Extraktion von Anforderungen standen frühzeitig fest. Auf der einen Seite die komplett über NLP Werkzeuge realisierte Lösung. Das überlegte Vorgehen bestand darin, Dokumente einzulesen und anhand komplexer Filtermethoden, die Anforderungen zu identifizieren. Auf der anderen Seite waren die Lernalgorithmen. Das überlegte Vorgehen bestand darin, mithilfe von Data Mining in den Texten nach Gemeinsamkeiten zu suchen. Das Ziel war es durch diese Gemeinsamkeiten Eigenschaften von Anforderungen zu bestimmen. Anschließend sollten die Eigenschaften genutzt werden um die Anforderungen zu extrahieren.

Von Anfang an sollten Ontologien genutzt werden, um nach der Extraktion, die Anforderungen zu klassifizieren. Die Wahl Ontologien auf für die Extraktion zu nutzen, war deswegen naheliegend. Zudem sie die Extraktion über Ontologien ein Mittelweg zwischen beiden Extremen. Es werden NLP Methoden genutzt um die Bestandteile zu identifizieren und anschließend werden, über die Inferenzberechnung, die logischen Verknüpfungen geschlossen.

Erzeugung der Datenstruktur Die nächste Hürde bot die Erzeugung eine Datenstruktur und die Wahl der Speicherung. Von Anfang an stand die persistente Haltung der Daten im Vordergrund. Da sowohl der Stanford CoreNLP als auch die OWL API Methoden für den XML Export lieferten, bot sich dieses Format an. In anbetracht potentiell großer Dokumentenbestände wurde alternativ die Speicherung in einer Datenbank in Erwägung

gezogen. Um eine größt mögliche Portabilität zu gewährleisten wurde schließlich XML als Datenformat gewählt.

Die Datenstruktur selbst sollte nun das Dokument möglichst getreu wiedergeben, alle relevanten Daten mit sich führen aber die daraus resultierenden Dateien möglichst klein sein. Vorallem die Informationen zur Darstellung der verschiedenen Klassifizierungsschemata stellten ein Problem dar. Eine seperate Speicherung der Darstellung, würde zu einer zusätzlichen Datei für jede angewendete Klassifizierung führen. Damit ein klassifiziertes Dokument bereits alle Informationen mit sich führt mussten sie als Metainformationen in die Datenstruktur mit einbezogen werden. Dies führte dazu, dass die Ontologien auch die Metainformationen mit sich führen muss, damit sie während der Klassifizierung nicht verlohren gehen.

Probleme der Implementierung Im Zuge der Implementierung wurden die Performanzprobleme der Inferenz offensichtlich. Die Anforderungsextraktionsontologie besteht aus 13 Klassen, was zu einem Zeitaufwand von $(13)^n$, mit $n = Anzahl\ Wörter + Anzahl\ Sätze + 1$ führt. Eine Verbesserung auf Seite der Ontologie war somit nicht zielführend. Eine Reduzierung der Eingabe Menge, insbesondere der Wörter, wurde als sinnvollste Option für eine Geschwindigkeitszunahme identifiziert. Daraus resultierte der OptimizedRequirementExtractor (siehe Abschnitt 5.2.3).

7.2. Zusammenfassung

Die Erhebung von Anforderungen aus natürlich sprachlichen Texten ist ein wichtiger Aufgabenteil eines jeden technischen Projektes. Die Extraktion von Anforderungen von Hand ist allerdings langwierig und anfällig für Fehler. Für diese Arbeit wurde deswegen ein System entworfen, dass gewichtete Anforderungen analysiert und auswertet. Es werden vom REC System Sätze danach untersucht, ob es sich bei ihnen um Anforderungen handelt. Dafür werden sie mithilfe einer Ontologie in Klassen unterteilt. Die betrachteten Klassen sind Anforderungen, schwache Anforderungen, optionale Anforderungen, Kandidaten für Anforderungen und keine Anforderungen. Die Ontologie nutzt (Subjekt, Verb, Objekt)-Tripel (SVO-Tripel) um Kandidaten zu finden und überprüft diese Kandidaten anschließend auf Schlüsselwörter.

Die Erkennungsontologie hat bei technischen Spezifikationen eine Trefferquote von über 80% bei einer Präzision von 100%. Es werden also keine Sätze fälschlicherweise als Anforderungen klassifiziert. Betrachtungen von anderen Texten ergaben, dass viele optionale Anforderungen tatsächliche keine Anforderungen waren. Dies führte zu einer Präzision von 50% bei Dokumenten, die keine technische Spezifikation sind. Die Qualität der Klassifizierung hängt somit stark von der Struktur des eingehenden Dokumentes ab.

Der REC kann auch genutzt werden, um Dokumente mit anderen Ontologien zu klassifizieren. Die Ontologien müssen eine Basisontologie (Abbildung 5.6) erweitern. Ein Dokument lässt sich ohne Wechselwirkung von mehreren Ontologien klassifizieren.

7.3. Ausblick

Das System ist an mehreren Stellen ausbaufähig. Da die Struktur der Dokumente die Qualität der Klassifizierung stark beeinflusst, ist es sinnvoll Nutzer bei der Erzeugung der Dokumente zu unterstützen. Wie genau diese Unterstützung aufgebaut sein sollte, muss in weiteren Arbeiten geklärt werden. Da die Klassifizierung großer Dokumente sehr lange dauert, muss für den praktischen Einsatz des REC die Performanz verbessert werden. Zu den denkbaren Ansätzen gehört dabei der Entwurf einer multithreading-fähigen Inferenzmaschine. Interessant wäre es auch zu sehen wie eine starke Nutzung von Algorithmen aus der KI die Erkennung von Anforderungen beeinflusst.

7.3.1. Erweiterungen des REC

Die Wahl der Eclipse RCP Architektur ermöglicht ein einheitliches Vorgehen, bei der Erweiterung des REC. Die folgenden drei Bereiche des REC haben besondere Aufmerksamkeit verdient.

NLP Analyse

Für die NLP Analyse wird derzeitig der Stanford CoreNLP genutzt. Wie bereits erwähnt bietet der Stanford CoreNLP eine schnelle NLP Lösung, die nicht trainiert werden muss.

Derzeitig wird von ihm allerdings nur die Englische Sprache erkannt. Da Module für andere Sprachen, so zum Beispiel Deutsch, vorliegen besteht die Möglichkeit den REC zu einem multilingualen Werkzeug zu erweitern. Eine Ontologie für die Extraktion von Anforderungen in Deutsch ist bereits vorhanden. Für eine erfolgreiche Anwendung dieser Ontologie müssen sowohl POS Analyse, als auch Lemmatisierung für Deutsch unterstützt werden.

Die Wahl des Stanford CoreNLP beruht darauf, dass er nicht trainiert werden muss. Eine weitere Arbeit bezüglich der NLP Analyse besteht darin ein Plugin zu implementieren, das statt dem Stanford CoreNLP ein anderes Programm nutzt.

Die von der Apache Software Foundation verwalteten Programme OpenNLP ([The12a]) und UIMA ([The12b]) bieten sich als Alternativen zu dem in Stanford entwickelten CoreNLP. Beide haben den Nachteil, dass sie trainiert werden müssen. Es lassen sich zwar Trainingsdaten im Internet (auch für verschiedene Sprachen) finden, aber da müssten auch erst Analysen durchgeführt werden, welcher Trainingsdatensatz am besten zu nutzen ist. Alternativ könnte auch eine neue Trainingsmenge erzeugt werden. Beim Apache UIMA handelt es sich um ein Framework mit dem große Mengen an unstructurierten Informationen verwaltet werden können. Die Apache OpenNLP Bibliothek ist ein Werkzeug für die Verarbeitung von natürlicher Sprache, dafür nutzt sie maschinelles Lernen.

Die Geschwindkeit des REC ist verbesserungswürdig. Da die NLP Analyse nur einen Bruchteil der Laufzeit in Anspruch nimmt, wird eine Performanzoptimierung des NLP Moduls nur geringe Auswirkungen auf die gesamt Performanz bieten.

Anforderungsextraktion

In dieser Arbeit wurde erörtet, in wie weit sich Ontologien für eine domänenunspezifische Extraktion, von Anforderungen, nutzen lassen. Eine mögliche Alternative besteht darin Algorithmen der künstlichen Intelligenz zu nutzen. Eine geziehlte Nutzung von maschinellem Lernen im Umfeld der domänenspezifischen Anforderungserhebung ist untersuchungswürdig. Insbesondere die Erkennungsrate, in zuvor trainierten Domänen, lässt sich möglicherweise durch Lernalgorithmen verbessern. Über den Extension Point für Klassifizierungen kann eine Lösung die auf Lernalgorithmen beruht im REC eingebunden werden.

Ontologien Insbesondere der Aspekt der Ontologien liefert einige Ansatzpunkte für die Erweiterung dieser Arbeit. Unter anderem ist die Performanz des Reasoners bei der Klassifizierung für einen produktiv Einsatz nicht akzeptabel. Durch ein satzweises Anwenden der Ontologie, bei der Extraktion, konnte die Performanz verbessert werden. Jedoch untersützt diese Lösung keine Koreferenzen mehr.

Für eine Steigerung der Performanz stehen mehrere Alternativen zur Verfügung. Unter anderem unterstützt der verwendete HermiT Reasoner kein Multithreading. Eine moderne Mehrkernarchitektur wird somit nicht vollständig ausgenutzt. Zwei verschiedene Optionen stehen hierbei zur Wahl. Zum Einen beschäftigen sich Koutsomitropoulos et. al. [KSP+10] damit, wie im Umfeld des semantischen Internets, die Rechenlast der Inferenzberechnung in verteilte Anwendungen ausgelagert werden kann. Die Andere Alternative besteht darin einen multithreadingfähigen Reasoner zu entwerfen. Es besteht Klärungsbedarf, ob dafür ein neuer Reasoner entworfen und implementiert wird, oder ob ein bestehender Reasoner erweitert wird.

Optimierung der Anzeige

Die Anzeige für die extrahierten Texte ist derzeitig funktional gestaltet. Eine Optimierung dieser Anzeige, für bessere Bedienbarkeit durch die Anwender, ist ein wichtiger Schritt zu einem Produktivsystem.

Möglichkeiten für Optimierung bestehen in der Darstellung der Tabelle. Die Präsentation der Klassifizierungsergebnisse sollte bei jeglicher Verbesserung Vorrang haben. Um die Übersicht zu verbessern, könne jede angewendete Ontologie ein eigenes Farbschema verwenden. Durch den Einsatz von Filterfunktionen würde das Arbeiten mit dem REC erleichtert werden. Bisher kann ein Nutzer die Klassifizierungen nach Klassen sortieren. Bei komplexeren Ontologien, oder großen Dokumenten, würde eine Möglichkeit nur bestimmte klassifizierte Sätze anzuzeigen, die Darstellung verbessern.

7.3.2. Fazit

Eine allgemeingültige, automatische Extraktion von Anforderungen ist nur schwer zu bewerkstelligen. Die automatisierte Extraktion von Anforderungen mit Ontologien ist um so erfolgreicher, je strukturierter die Anforderungen formuliert sind.

Eine mögliche Lösungbesteht im Ansatz von Farfeleder et. al. 3.1.3. Jedoch wäre eine domänenunspezifische Lösung dieses Ansatzes von Vorteil. Die daraus resultierenden semistrukturierten Anforderungen lassen sich dann mit Ontologien extrahieren und klassifizieren.

Die beim Entwurf des REC gewohnenen Erkenntnisse führen zum Schluss, dass eine Verbesserung der textuellen Anforderungen am zielführendsten ist. Werkzeuge wie der REC sollten im Kern domänenunspezifisch arbeiten, damit sie schnell eingesetzt werden können. Für eine qualitative Verbesserunge der Anforderungsextraktion, sollen sie jedoch zusätzlich mit domänenspezifischen „Profilen“ arbeiten. Ein solchen „Profil“ würde einer domänenspezifischen Ontologie, wie sie von Farfeleder et. al. vorgeschlagen wird, entsprechen. Das Werkzeug müsste mit Lernalgorithmen ausgestattet sein um mit der Zeit diese Ontologie zu erzeugen und zu verbessern. Im produktiven Einsatz kann eine Firma zu jedem Kunden, oder jedem thematisch verwandtem Projekt, ein seperates Profil verwenden. Durch einen kontinuierlichen Einsatz würde sich die Erkennungsrate permanent verbessern.

Die mit einem sochen System erzeugten Spezifikationen hätten genügend Struktur, dass eine anschließende Klassifizierung zielführend erfolgt. Wenn die Sätze klassifiziert sind können sie mit zusätzlichen Werkzeugen ([Lan10], [Tun12]) mit semantischen Informationen bereichert werden.

Mit all diesen Informationen (Semantik, Klassifizierung) sollte es möglich sein, im Sinne der MDA sinnvolle Modelle zu erzeugen. Anhand dieser Schritte wird eine weitere Automatisierung der MDA erzielt.

A. Anhang

A.1. Abkürzungsverzeichnis

CIM Computation Independant Model

IRI Internationalized Ressource Identifier

JAXB Java Architecture for XML Binding

KI Künstlichen Intelligenz

MDA Model Driven Architecture

NER Erkenner von benannten Entitäten (Named-entity recognition)

NFR Nicht-funktionalen Anforderungen (Non-functional Requirements)

NHTSA National Highway Traffic Safety Administration

NLP Natural Language Processing

OMG Object Managment Group

OSSD Open Source Software Entwicklung (Open Source Software Development)

OWL Web Ontology Language

PIM Platform Independent Model

POS Part-of-Speech

PSM Platform Specific Model

RCP Rich Client Platform

RDF Ressource Description Framework

REC Requirement Extractor and Classifier

SVO-Tripel (Subjekt, Verb, Objekt)-Tripel

SWT Standard Widget Toolkit

UML Unified Modeling Language

W3C World Wide Web Consortium

XML Extensible Markup Language

A.2. Großbeispiele

A.2.1. Stanford CoreNLP XML Ausgabe

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="CoreNLP-to-HTML.xsl" type="text/xsl"?>
<root>
  <document>
    <sentences>
      <sentence id="1">
        <tokens>
          <token id="1">
            <word>Stanford</word>
            <lemma>Stanford</lemma>
            <CharacterOffsetBegin>0</CharacterOffsetBegin>
            <CharacterOffsetEnd>8</CharacterOffsetEnd>
            <POS>NNP</POS>
            <NER>ORGANIZATION</NER>
          </token>
          <token id="2">
            <word>University</word>
            <lemma>University</lemma>
            <CharacterOffsetBegin>9</CharacterOffsetBegin>
            <CharacterOffsetEnd>19</CharacterOffsetEnd>
            <POS>NNP</POS>
            <NER>ORGANIZATION</NER>
          </token>
          <token id="3">
            <word>is</word>
            <lemma>be</lemma>
            <CharacterOffsetBegin>20</CharacterOffsetBegin>
            <CharacterOffsetEnd>22</CharacterOffsetEnd>
            <POS>VBZ</POS>
            <NER>O</NER>
          </token>
          <token id="4">
            <word>located</word>
            <lemma>locate</lemma>
            <CharacterOffsetBegin>23</CharacterOffsetBegin>
            <CharacterOffsetEnd>30</CharacterOffsetEnd>
            <POS>VBN</POS>
            <NER>O</NER>
          </token>
          <token id="5">
            <word>in</word>
            <lemma>in</lemma>
            <CharacterOffsetBegin>31</CharacterOffsetBegin>
            <CharacterOffsetEnd>33</CharacterOffsetEnd>
            <POS>IN</POS>
            <NER>O</NER>
          </token>
          <token id="6">
            <word>California</word>
            <lemma>California</lemma>
```

```
    <CharacterOffsetBegin>34</CharacterOffsetBegin>
    <CharacterOffsetEnd>44</CharacterOffsetEnd>
    <POS>NNP</POS>
    <NER>LOCATION</NER>
  </token>
  <token id="7">
    <word>.</word>
    <lemma>.</lemma>
    <CharacterOffsetBegin>44</CharacterOffsetBegin>
    <CharacterOffsetEnd>45</CharacterOffsetEnd>
    <POS>.</POS>
    <NER>O</NER>
  </token>
</tokens>
<parse>(ROOT (S (NP (NNP Stanford) (NNP University)) (VP (VBZ is) (VP (VBN located)
                    (PP (IN in) (NP (NNP California))))) (. .))) </parse>
<basic-dependencies>
  <dep type="nn">
    <governor idx="2">University</governor>
    <dependent idx="1">Stanford</dependent>
  </dep>
  <dep type="nsubjpass">
    <governor idx="4">located</governor>
    <dependent idx="2">University</dependent>
  </dep>
  <dep type="auxpass">
    <governor idx="4">located</governor>
    <dependent idx="3">is</dependent>
  </dep>
  <dep type="prep">
    <governor idx="4">located</governor>
    <dependent idx="5">in</dependent>
  </dep>
  <dep type="pobj">
    <governor idx="5">in</governor>
    <dependent idx="6">California</dependent>
  </dep>
</basic-dependencies>
<collapsed-dependencies>
  <dep type="nn">
    <governor idx="2">University</governor>
    <dependent idx="1">Stanford</dependent>
  </dep>
  <dep type="nsubjpass">
    <governor idx="4">located</governor>
    <dependent idx="2">University</dependent>
  </dep>
  <dep type="auxpass">
    <governor idx="4">located</governor>
    <dependent idx="3">is</dependent>
  </dep>
  <dep type="prep_in">
    <governor idx="4">located</governor>
    <dependent idx="6">California</dependent>
  </dep>
```

```
    </collapsed-dependencies>
    <collapsed-ccprocessed-dependencies>
      <dep type="nn">
        <governor idx="2">University</governor>
        <dependent idx="1">Stanford</dependent>
      </dep>
      <dep type="nsubjpass">
        <governor idx="4">located</governor>
        <dependent idx="2">University</dependent>
      </dep>
      <dep type="auxpass">
        <governor idx="4">located</governor>
        <dependent idx="3">is</dependent>
      </dep>
      <dep type="prep_in">
        <governor idx="4">located</governor>
        <dependent idx="6">California</dependent>
      </dep>
    </collapsed-ccprocessed-dependencies>
  </sentence>
  <sentence id="2">
    <tokens>
      <token id="1">
        <word>It</word>
        <lemma>it</lemma>
        <CharacterOffsetBegin>46</CharacterOffsetBegin>
        <CharacterOffsetEnd>48</CharacterOffsetEnd>
        <POS>PRP</POS>
        <NER>O</NER>
      </token>
      <token id="2">
        <word>is</word>
        <lemma>be</lemma>
        <CharacterOffsetBegin>49</CharacterOffsetBegin>
        <CharacterOffsetEnd>51</CharacterOffsetEnd>
        <POS>VBZ</POS>
        <NER>O</NER>
      </token>
      <token id="3">
        <word>a</word>
        <lemma>a</lemma>
        <CharacterOffsetBegin>52</CharacterOffsetBegin>
        <CharacterOffsetEnd>53</CharacterOffsetEnd>
        <POS>DT</POS>
        <NER>O</NER>
      </token>
      <token id="4">
        <word>great</word>
        <lemma>great</lemma>
        <CharacterOffsetBegin>54</CharacterOffsetBegin>
        <CharacterOffsetEnd>59</CharacterOffsetEnd>
        <POS>JJ</POS>
        <NER>O</NER>
      </token>
      <token id="5">
```

```
    <word>university</word>
    <lemma>university</lemma>
    <CharacterOffsetBegin>60</CharacterOffsetBegin>
    <CharacterOffsetEnd>70</CharacterOffsetEnd>
    <POS>NN</POS>
    <NER>O</NER>
  </token>
  <token id="6">
    <word>.</word>
    <lemma>.</lemma>
    <CharacterOffsetBegin>70</CharacterOffsetBegin>
    <CharacterOffsetEnd>71</CharacterOffsetEnd>
    <POS>.</POS>
    <NER>O</NER>
  </token>
</tokens>
<parse>(ROOT (S (NP (PRP It)) (VP (VBZ is) (NP (DT a) (JJ great)
                    (NN university))) (. .))) </parse>
<basic-dependencies>
  <dep type="nsubj">
    <governor idx="5">university</governor>
    <dependent idx="1">It</dependent>
  </dep>
  <dep type="cop">
    <governor idx="5">university</governor>
    <dependent idx="2">is</dependent>
  </dep>
  <dep type="det">
    <governor idx="5">university</governor>
    <dependent idx="3">a</dependent>
  </dep>
  <dep type="amod">
    <governor idx="5">university</governor>
    <dependent idx="4">great</dependent>
  </dep>
</basic-dependencies>
<collapsed-dependencies>
  <dep type="nsubj">
    <governor idx="5">university</governor>
    <dependent idx="1">It</dependent>
  </dep>
  <dep type="cop">
    <governor idx="5">university</governor>
    <dependent idx="2">is</dependent>
  </dep>
  <dep type="det">
    <governor idx="5">university</governor>
    <dependent idx="3">a</dependent>
  </dep>
  <dep type="amod">
    <governor idx="5">university</governor>
    <dependent idx="4">great</dependent>
  </dep>
</collapsed-dependencies>
<collapsed-ccprocessed-dependencies>
```

```
        <dep type="nsubj">
          <governor idx="5">university</governor>
          <dependent idx="1">It</dependent>
        </dep>
        <dep type="cop">
          <governor idx="5">university</governor>
          <dependent idx="2">is</dependent>
        </dep>
        <dep type="det">
          <governor idx="5">university</governor>
          <dependent idx="3">a</dependent>
        </dep>
        <dep type="amod">
          <governor idx="5">university</governor>
          <dependent idx="4">great</dependent>
        </dep>
      </collapsed-ccprocessed-dependencies>
    </sentence>
  </sentences>
  <coreference>
    <coreference>
      <mention representative="true">
        <sentence>2</sentence>
        <start>3</start>
        <end>6</end>
        <head>5</head>
      </mention>
      <mention>
        <sentence>2</sentence>
        <start>1</start>
        <end>2</end>
        <head>1</head>
      </mention>
    </coreference>
  </coreference>
 </document>
</root>
```

Listing A.1: Stanford CoreNLP XML Ausgabe der Sätze „Stanford University is located in California. It is a great university.“

A.2.2. OWL Ontologie XML

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
```

```
 xml:base="http://www.rec.iris.uni-stuttgart.de/SampleOntologie.owl"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
 xmlns:xml="http://www.w3.org/XML/1998/namespace"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 ontologyIRI="http://www.rec.iris.uni-stuttgart.de/SampleOntologie.owl">
<Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
<Prefix name="" IRI="http://www.w3.org/2002/07/owl#"/>
<Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
<Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
<Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
<Declaration>
    <Class IRI="#Document"/>
</Declaration>
<Declaration>
    <Class IRI="#Sentence"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasSentence"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasName"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasOriginalSentence"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#Example_1"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#Sentence_1_1"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#Sentence_1_2"/>
</Declaration>
<ClassAssertion>
    <Class IRI="#Document"/>
    <NamedIndividual IRI="#Example_1"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Sentence"/>
    <NamedIndividual IRI="#Sentence_1_1"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Sentence"/>
    <NamedIndividual IRI="#Sentence_1_2"/>
</ClassAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#hasSentence"/>
    <NamedIndividual IRI="#Example_1"/>
    <NamedIndividual IRI="#Sentence_1_1"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#hasSentence"/>
    <NamedIndividual IRI="#Example_1"/>
```

```
        <NamedIndividual IRI="#Sentence_1_2"/>
    </ObjectPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#hasName"/>
        <NamedIndividual IRI="#Example_1"/>
        <Literal datatypeIRI="&rdf;PlainLiteral">XML Example 1</Literal>
    </DataPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#hasOriginalSentence"/>
        <NamedIndividual IRI="#Sentence_1_1"/>
        <Literal datatypeIRI="&rdf;PlainLiteral">Stanford University is located in
            California.</Literal>
    </DataPropertyAssertion>
    <DataPropertyAssertion>
        <DataProperty IRI="#hasOriginalSentence"/>
        <NamedIndividual IRI="#Sentence_1_2"/>
        <Literal datatypeIRI="&rdf;PlainLiteral">IIt is a great university.</Literal>
    </DataPropertyAssertion>
    <InverseFunctionalObjectProperty>
        <ObjectProperty IRI="#hasSentence"/>
    </InverseFunctionalObjectProperty>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#hasSentence"/>
        <Class IRI="#Document"/>
    </ObjectPropertyDomain>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#hasSentence"/>
        <Class IRI="#Sentence"/>
    </ObjectPropertyRange>
    <FunctionalDataProperty>
        <DataProperty IRI="#hasName"/>
    </FunctionalDataProperty>
    <DataPropertyDomain>
        <DataProperty IRI="#hasName"/>
        <Class IRI="#Document"/>
    </DataPropertyDomain>
    <DataPropertyDomain>
        <DataProperty IRI="#hasOriginalSentence"/>
        <Class IRI="#Sentence"/>
    </DataPropertyDomain>
    <DataPropertyRange>
        <DataProperty IRI="#hasName"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataPropertyRange>
    <DataPropertyRange>
        <DataProperty IRI="#hasOriginalSentence"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataPropertyRange>
</Ontology>


<!-- Generated by the OWL API (version 3.2.3.1824) http://owlapi.sourceforge.net -->
```

Listing A.2: XML Darstellung einer stark reduzierten Dokumentenontologie.

A.2.3. REC Dokumentenbeispiel

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<document fileName="TextExample">
  <sentences>
    <Sentence id="TextExample_0">
      <ClassificationMapping>
        <entry>
          <key>
            http://www.rec.iris.uni-stuttgart.de/RequirementExtractionOntology.owl
          </key>
          <value>
            <Classifications>
              <Classification>
                <ontologyClass>#Candidate</ontologyClass>
              </Classification>
              <Classification>
                <ontologyClass>#Requirement</ontologyClass>
              </Classification>
            </Classifications>
          </value>
        </entry>
      </ClassificationMapping>
      <Dependencies>
        <Dependency dependencyType="dobj">
          <dependant>TextExample_0_5</dependant>
          <governor>TextExample_0_3</governor>
        </Dependency>
        <Dependency dependencyType="num">
          <dependant>TextExample_0_4</dependant>
          <governor>TextExample_0_5</governor>
        </Dependency>
        <Dependency dependencyType="det">
          <dependant>TextExample_0_0</dependant>
          <governor>TextExample_0_1</governor>
        </Dependency>
        <Dependency dependencyType="nsubj">
          <dependant>TextExample_0_1</dependant>
          <governor>TextExample_0_3</governor>
        </Dependency>
        <Dependency dependencyType="aux">
          <dependant>TextExample_0_2</dependant>
          <governor>TextExample_0_3</governor>
        </Dependency>
      </Dependencies>
      <originalSentence>The airplane must have two wings .</originalSentence>
      <Words>
        <Word id="TextExample_0_0">
          <word>The</word>
          <lemma>the</lemma>
          <pos>DT</pos>
          <ner>O</ner>
          <CharacterOffsetBegin>0</CharacterOffsetBegin>
          <CharacterOffsetEnd>3</CharacterOffsetEnd>
        </Word>
```

```
      <Word id="TextExample_0_1">
        <word>airplane</word>
        <lemma>airplane</lemma>
        <pos>NN</pos>
        <ner>O</ner>
        <CharacterOffsetBegin>4</CharacterOffsetBegin>
        <CharacterOffsetEnd>12</CharacterOffsetEnd>
      </Word>
      <Word id="TextExample_0_2">
        <word>must</word>
        <lemma>must</lemma>
        <pos>MD</pos>
        <ner>O</ner>
        <CharacterOffsetBegin>13</CharacterOffsetBegin>
        <CharacterOffsetEnd>17</CharacterOffsetEnd>
      </Word>
      <Word id="TextExample_0_3">
        <word>have</word>
        <lemma>have</lemma>
        <pos>VB</pos>
        <ner>O</ner>
        <CharacterOffsetBegin>18</CharacterOffsetBegin>
        <CharacterOffsetEnd>22</CharacterOffsetEnd>
      </Word>
      <Word id="TextExample_0_4">
        <word>two</word>
        <lemma>two</lemma>
        <pos>CD</pos>
        <ner>NUMBER</ner>
        <CharacterOffsetBegin>23</CharacterOffsetBegin>
        <CharacterOffsetEnd>26</CharacterOffsetEnd>
      </Word>
      <Word id="TextExample_0_5">
        <word>wings</word>
        <lemma>wing</lemma>
        <pos>NNS</pos>
        <ner>O</ner>
        <CharacterOffsetBegin>27</CharacterOffsetBegin>
        <CharacterOffsetEnd>32</CharacterOffsetEnd>
      </Word>
      <Word id="TextExample_0_6">
        <word>.</word>
        <lemma>.</lemma>
        <pos>.</pos>
        <ner>O</ner>
        <CharacterOffsetBegin>32</CharacterOffsetBegin>
        <CharacterOffsetEnd>33</CharacterOffsetEnd>
      </Word>
    </Words>
  </Sentence>
</sentences>
<classificationOntologies>
  <ClassificationOntologies>
    http://www.rec.iris.uni-stuttgart.de/RequirementExtractionOntology.owl
  </ClassificationOntologies>
```

```
  </classificationOntologies>
  <Dependencies/>
</document>
```

Listing A.3: Beispiel des REC XML Dokuments zum klassifizierten Satz „The airplane must have two wings.“

Literaturverzeichnis

[AHH06] Y. Alkhader, A. Hudaib, B. Hammo. Experimenting With Extracting Software Requirements Using NLP Approach. 2006. (Zitiert auf Seite 25)

[BH11] E. Boutkova, F. Houdek. Semi-Automatic Identification of Features in Requirement Specifications. *19th International Requirements Engineering Conference*, 2011.

[CHLL10] H. Chen, K. He, P. Liang, R. Li. Text-based Requirements Preprocessing using Nature Language Processing Techniques. *International Conference On Computer Design and Applications*, 2010.

[CHSZS06] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc. The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. *14th IEEE International Requirements Engineering Conference*, 2006. (Zitiert auf den Seiten 10, 14, 25, 26, 27, 39 und 53)

[CNRW06] F. Chantree, B. Nuseibeh, A. de Roeck, A. Willis. Identifying Nocuous Ambiguities in Natural Language Requirements. *14th IEEE International Requirements Engineering Conference*, 2006.

[Dep12] Department of Computer Science University of Oxford. HermiT OWL Reasoner 1.3.6, 2012. URL http://hermit-reasoner.com/. (Zitiert auf Seite 31)

[EAAG08] M. Eaddy, A. V. Aho, G. Antoniol, Y.-G. Gueheneuc. CERBERUS: Tracing Requirements to Source Code Using Information Retriecal, Dynamic Analysis, and Program Analysis. *16th IEEE International Conference on Program Comprehension*, 2008.

[Ecl12] Eclipse Foundation. Eclipse, 2012. URL http://www.eclipse.org/. (Zitiert auf den Seiten 9 und 32)

[FIC08] S. A. Fahmi, A. Ibrahim, H.-J. Choi. An Approach to Support and Partially Automate Requirements Engineering Activities. *8th International Conferrence on Computer and Information Technology Workshops*, 2008.

[FMK+11] S. Farfeleder, T. Moser, A. Krall, T. Stalhane, I. Omoronyia, H. Zojer. Ontology-Driven Guidance for Requirements Elicitation. *Extended Semantic Web Conference*, 2011. (Zitiert auf den Seiten 14 und 27)

[GK11] F. Gharehchopogh, Z. Khalifelu. Analysis and Evaluation of Unstructured Data: Text Mining versus Natural Language Processing. 2011. (Zitiert auf Seite 39)

[IA10] M. Ibrahim, R. Ahmad. Class diagram extraction from textual requirements using Natural language processing (NLP) techniques. *Second International Conference on Computer Research and Development*, 2010.

[KS06] H. Kaiya, M. Saeki. Using Domain Ontology as Domain Knowledge for Requirements Elicitation. *14th IEEE International Requirements Engineering Conference*, 2006.

[KSP+10] D. Koutsomitropoulos, G. Solomou, T. Pomonis, P. Aggelopoulos, T. Papatheodorou. Developing Distributed Reasoning-Based Applications for the Semantic Web. 2010. (Zitiert auf Seite 64)

[Lan10] M. Landhaeusser. *Automatische Auszeichnung von Semantik in natuerlichsprachlichen Spezifikationen*. Diplomarbeit, Karlsruher Institut fuer Technologie, 2010. (Zitiert auf den Seiten 20 und 65)

[LL07] J. Ludewig, H. Lichter. *Software Engineering*. dpunkt.verlag, 2007. (Zitiert auf den Seiten 12 und 19)

[LNN11] K.-K. Lau, A. Nordin, K.-Y. Ng. Extracting Elements of Component-based Systems from Natural Language Requirements. *37th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2011.

[LP94] J. Liang, J. D. Palmer. A Pattern Matching and Clustering Based Approach for Supporting Requirements Transformation. 1994.

[MFI04] L. Mich, M. Franch, P. N. Inverardi. Market research for requirements analysis using linguistic tools. *Requirements Engineering 9*, 2004. URL http://dx.doi.org/10.1007/s00766-003-0179-8. (Zitiert auf Seite 20)

[MM03] J. Mukerji, J. Miller. Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1, 2003. URL http://www.omg.org/cgi-bin/doc?omg/03-06-01. (Zitiert auf Seite 17)

[Mun12] R. Munroe. xkcd: A webcomic of romance, sarcasm, math, and language., 2012. URL xkcd.com. (Zitiert auf den Seiten 9 und 11)

[MWG09] Y. Mu, Y. Wang, J. Guo. Extracting Software Funcional Requirements from Free Text Documents. *International Conference on Information and Multimedia Technology*, 2009.

[NE08] N. Niu, S. Easterbrook. Extracting and Modeling Product Line Funcional Requirements. *16th IEEE International Requirements Engineering Conference*, 2008.

[RAC11] R. Rauf, M. Antkiewicz, K. Czarnecki. Logical Structure Extraction from Software Requirements Documents. *19th International Requirements Engineering Conference*, 2011.

[RSN03] P. Roßbach, T. Stahl, W. Neuhaus. Model Driven Architecture - Grundlegende Konzepte und Einordnung der Model Driven Architecture. 2003. URL http://it-republik.de/jaxenter/artikel/Model-Driven-Architecture-0408.html.

[SA 12] SA Depot, 2012. URL http://www.sa-depot.com. (Zitiert auf den Seiten 9 und 17)

[SMA+08] J. Santos, A. Moreira, J. Araujo, V. Amaral, M. Alferez, U. Kulesza. Generating Requirements Analysis Models from Textual Requirements. *First International Workshop on Managing Requirements Knowledge*, 2008.

[Spe12] S. Speirs, 2012. URL http://blogs.cisco.com/datacenter/how-a-customer-crisis-ten-years-ago-helped-me-understand-the-challenges-of-cloud-service-creation-today/. (Zitiert auf den Seiten 9 und 35)

[SSG09] S. K. Sigh, S. Sabharwal, J. Gupta. E-XTRACT: A Tool for Extraction, Analysis and Classification of Events from Textual Requirements. *International Conference on Advances in Recent Technologies in Communication and Computing*, 2009.

[Sta] Stanford Natural Language Processing Group. The Stanford NLP Group. URL http://nlp.stanford.edu/. (Zitiert auf Seite 28)

[Sta12a] Stanford Center for Biomedical Informatics Research. Protege 4.1, 2012. URL http://protege.stanford.edu/.

[Sta12b] Stanford Natural Language Processing Group. Stanford CoreNLP v1.3.3, 2012. URL http://nlp.stanford.edu/software/corenlp.shtml. (Zitiert auf den Seiten 10 und 29)

[The12a] The Apache Software Foundation. Apache OpenNLP, 2012. URL http://opennlp.apache.org/. (Zitiert auf Seite 63)

[The12b] The Apache Software Foundation. Apache UIMA, 2012. URL http://uima.apache.org/. (Zitiert auf Seite 63)

[Tun12] M. Tuncer. *Implementierung von Methoden zur Auswertung von textuellen Anforderungen im WRSPM-Umfeld*. Diplomarbeit, Universitaet Stuttgart, 2012. (Zitiert auf Seite 65)

[Uni90] University of Pennsylvania. The University of Pennsylvania (Penn) Treebank Tag-set, 1990. URL http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html. (Zitiert auf Seite 47)

[Uni12a] University of Manchester. FaCT++, 2012. URL http://code.google.com/p/factplusplus/. (Zitiert auf Seite 31)

[Uni12b] University of Manchester. OWL API Version 3.4.1, 2012. URL http://owlapi.sourceforge.net/. (Zitiert auf Seite 31)

[VR11] R. Vlas, W. N. Robinson. A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. *Proceedings of the 44th Hawaii International Conference on System Sciences*, 2011. (Zitiert auf den Seiten 14 und 25)

[W3C12] W3C, 2012. URL http://www.w3.org. (Zitiert auf den Seiten 9 und 30)

[Wit07] S. Withall. *Software Requirement Patterns (Best Practices)*. Microsoft Press, 2007. (Zitiert auf Seite 20)

[XLQY07] J. Xiang, L. Liu, W. Qiao, J. Yang. SREM: A Service Requirements Elicitation Mechanism based on Ontology. *31st Annual International Computer Software and Applications Conference*, 2007.

[ZJ00] H. Zhu, L. Jin. Automating Scenario-Driven Structured Requirements Engineering. 2000.

[ZYZXYY+07] L. Zong-Yong, W. Zhi-Xue, Y. Ying-Ying, W. Yue, L. Ying. Towards a Multiple Ontology Framework for Requirements Elicitation and Reuse. *31st Annual International Computer Software and Applications Conference*, 2007.

[ZZ04] X. Zhou, N. Zhou. Auto-generation of Class Diagram from Free-text Functional SPecifications and Domain Ontology. 2004.

Alle URLs wurden zuletzt am 08. 03. 2013 geprüft.

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, Unterschift