Indikatorbasierte Erkennung und Kompensation von ungenauen und unvollständig beschriebenen Softwareanforderungen

Der Fakultät für Wirtschaftswissenschaften der

Universität Paderborn

zur Erlangung des akademischen Grades

Doktor der Wirtschaftswissenschaften

– *Doctor rerum politicarum* –

vorgelegte Dissertation

von

Frederik Simon Bäumer

geboren am 09. Februar 1988

in Aachen

Tag des Kolloquiums:	26. Juli 2017
Referentin:	Jun.-Prof. Dr. Michaela Geierhos
Korreferent:	Prof. Dr.-Ing. habil. Wilhelm Dangelmaier

Vorveröffentlichungen

Im Zusammenhang mit der vorliegenden Dissertation veröffentliche Beiträge:

- Michaela Geierhos & **Frederik S. Bäumer**: In Henning Christiansen, M. Dolores Jiménez López, Roussanka Loukanov & Larry Moss (Hrsg.): *Partiality and Underspecication in Information, Languages, and Knowledge*, S. 65–107. Cambridge Scholars Publishing.

- **Frederik S. Bäumer** & Michaela Geierhos: *Running Out of Words: How Similar User Stories Can Help to Elaborate Individual Natural Language Requirement Descriptions*. In: Dregvaite, Giedre, Damasevicius, Robertas (Hrsg.): Information and Software Technologies – 22nd International Conference, ICIST 2016, Druskininkai, Lithuania, October 13-15, 2016, Proceedings, CCIS 639, S. 549–558. Springer. ISBN 978-3-319-46253-0. doi:10.1007/978-3-319-46254-7.

- Michaela Geierhos & **Frederik S. Bäumer**: *How to Complete Customer Requirements Using Concept Expansion for Requirement Refinement*. In: Proceedings of the 21st International Conference on Applications of Natural Language to Information Systems, NLDB 2016, Springer, LNAI 9612, 2016, Salford, UK, Juni 2016, S. 37–47. ISBN 978-3-319-41753-0. doi:10.1007/978-3-319-41754-7_4.

- Lorijn van Rooijen, **Frederik S. Bäumer**, Marie Christin Platenius, Michaela Geierhos, Heiko Hamann & Gregor Engels: *From User Demand to Software Service: Using Machine Learning to Automate the Requirements Specication Process*. In: Proceedings of the 4th International Workshop on Articial Intelligence for Requirements Engineering (AIRE'17), 5. September 2017, Lissabon. (im Druck)

- **Frederik S. Bäumer**, Markus Dollmann & Michaela Geierhos: *Studying Software Descriptions in SourceForge and App Stores for a better Understanding of real-life Requirements*. In: Proceedings of the 2nd International Workshop on App Market Analytics (WAMA 2017), 5. September 2017, Paderborn. (im Druck)

- Michaela Geierhos, Sabine Schulze & **Frederik Bäumer**: *What did you mean? Facing the Challenges of User-generated Software Requirements*. In Proceedings of the 7th International Conference on Agents and Articial Intelligence (ICAART), Lissabon, Januar 2015. S. 277–283. ISBN 978-989-758-073-4

Zusammenfassung

Die vorliegende Dissertation ist im Rahmen des Sonderforschungsbereichs 901: *On-The-Fly Computing* (auch: OTF-*Computing*) entstanden. Die Vision des OTF-*Computings* sieht vor, dass zukünftig der individuelle Softwarebedarf von Endanwendern durch die automatische Komposition bestehender Softwareservices gedeckt wird. Im Fokus stehen dabei natürlichsprachliche Softwareanforderungen, die Endanwender formulieren und an OTF-Anbieter als Anforderungsbeschreibung übergeben. Sie dienen an dieser Stelle als alleinige Kompositionsgrundlage, können allerdings ungenau und unvollständig sein. Dies sind Defizite, die bislang durch Softwareentwickler im Rahmen eines bidirektionalen Konsolidierungsprozesses erkannt und behoben wurden. Allerdings ist eine solche Qualitätssicherung im OTF-*Computing* nicht mehr vorgesehen – der klassische Konsolidierungsprozess entfällt.

Hier setzt die Dissertation an, indem sie sich mit Ungenauigkeiten frei formulierter Anforderungsbeschreibungen beim Softwareentwurf auseinandersetzt. Hierfür wird mit CORDULA (*Compensation of Requirements Descriptions Using Linguistic Analysis*) ein System entwickelt, dass sprachliche Unzulänglichkeiten (Ambiguität, Vagheit sowie Unvollständigkeit) in den Formulierungen unerfahrener Endanwender erkennt und kompensiert. CORDULA unterstützt dabei die Suche nach geeigneten Softwareservices zur Komposition, indem Anforderungsbeschreibungen in kanonische Kernfunktionalitäten überführt werden.

Die vorliegende Arbeit leistet somit methodisch gesehen einen Beitrag zur ganzheitlichen Erfassung und Verbesserung sprachlicher Unzulänglichkeiten in nutzergenerierten Anforderungsbeschreibungen, indem erstmalig parallel und sequenziell Ambiguität, Unvollständigkeit und Vagheit behandelt werden. Erst durch den Einsatz linguistischer Indikatoren ist es möglich, datengetrieben und bedarfsorientiert die individuelle Textqualität zu optimieren, indem von der klassischen Textanalysepipeline abgewichen wurde: Die *ad hoc*-Konfiguration der Kompensationspipeline, ausgelöst durch die *On-The-Fly* festgestellten Defizite in den Anforderungsbeschreibungen der Endanwender, ist ein Alleinstellungsmerkmal.

Abstract

This dissertation has been written within the scope of Collaborative Research Centre 901: On-The-Fly Computing (also known as OTF Computing). The vision of OTF Computing is to have the software needs of end users in the future covered by an automatic composition of existing software services. Here we focus on natural language software requirements that end users formulate and submit to OTF providers as requirement specifications. These requirements serve as the sole foundation for the composition of software; but they can be inaccurate and incomplete. Up to now, software developers have identified and corrected these deficits by using a bidirectional consolidation process. However, this type of quality assurance is no longer included in OTF Computing – the classic consolidation process is dropped.

This is where this work picks up, dealing with the inaccuracies of freely formulated software design requirements. To do this, we developed the CORDULA (Compensation of Requirements Descriptions Using Linguistic Analysis) system that recognizes and compensates for language deficiencies (e.g., ambiguity, vagueness and incompleteness) in requirements written by inexperienced end users. CORDULA supports the search for suitable software services that can be combined in a composition by transferring requirement specifications into canonical core functionalities.

This dissertation provides the first-ever method for holistically recording and improving language deficiencies in user-generated requirement specifications by dealing with ambiguity, incompleteness and vagueness in parallel and in sequence. Using linguistic indicators makes it possible to optimize the individual text quality in a data-driven and needs-oriented manner by deviating from the classical text analysis pipeline: Its distinguishing feature is the ad hoc configuration of the compensating pipeline, triggered by the deficiencies that On-The-Fly Computing detected in the requirement specifications of end users.

Danksagung

An dieser Stelle möchte ich allen danken, die mich auf unterschiedlichste Weise bei der Erstellung dieser Arbeit unterstützt haben.

Zuerst möchte ich Jun.-Prof. Dr. phil. habil. Michaela Geierhos danken, die meine Dissertation betreut und mich dabei in einem beachtlichen Maße unterstützt hat. Sie hat mich motiviert, mir Etappenziele sowie Herausforderungen aufgezeigt und mir diese vor allem auch zugetraut. Für das in mich gesetzte Vertrauen, ihre Geduld und ihre Diskussionsbereitschaft bedanke ich mich vielmals.

Ich bedanke mich darüber hinaus bei Prof. Dr.-Ing. habil. Wilhelm Dangelmaier für die Übernahme des Zweitgutachtens sowie die konstruktiven Anmerkungen, mit denen er meine Dissertation in unseren Gesprächen bedacht hat. Mein Dank richtet sich auch an Prof. Dr. René Fahr und Prof. Dr.-Ing. Heiko Hamann für die Bereitschaft, als Mitglieder in meiner Promotionskommission zu fungieren.

Besonders danken möchte ich meinen Kollegen am Heinz Nixdorf Institut, Stephan Abke, Markus Dollmann, Nicolai Grote, Annette Steffens und Jens Weber, für all die hilfreichen und konstruktiven Ratschläge. Danke für die gute Zusammenarbeit! Unterstützt wurde ich während meiner Tätigkeit zusätzlich durch Edwin Friesen, Marcel Grawe und Joschka Kersting, die durch vielfältige Unterstützung zum Gelingen dieser Arbeit beigetragen haben. Weiterhin möchte ich mich bei Sven Heim bedanken, der eine große Hilfe, Kritiker und guter Diskussionspartner war.

Mein Dank geht auch an meine Kollegen des Sonderforschungsbereichs 901. Hier möchte ich mich insbesondere bei Dr. Marie Christin Platenius und Dr. Lorijn van Rooijen für die gute Zusammenarbeit im Teilprojekt bedanken.

Zum Schluss möchte ich meiner Familie für das Lektorat danken. Mein großer Dank geht an meine Eltern, die mich immer unterstützten und es auch bei dieser Arbeit taten. Ich habe zwei tolle Schwestern, die ich nicht missen möchte und eine wunderbare Freundin, die sich lange Monologe über indikatorbasierte Kompensation angehört und mir den Rücken freigehalten hat. Allen bin ich sehr dankbar.

Paderborn, Juli 2017 Frederik Simon Bäumer

Inhaltsverzeichnis

Abkürzungsverzeichnis

a *Accuracy*

ACE *Automatic Content Extraction (Evaluation)*

ADV Adverb

AIC *Ambiguity Indicator Corpus*

API *Application Programming Interface*

ASCII *American Standard Code for Information Interchange*

BART *Beautiful Anaphora Resolution Toolkit*

BLANC *BiLateral Assessment of Noun-phrase Coreference*

BNC *British National Corpus*

CAR *Completeness Assistant for Requirements*

CD *Cardinal Number*

CDC *Conan Doyle Corpus*

CEAF *Constrained Entity Aligned F-measure*

CoNLL *Conference on Natural Language Learning*

CORDULA *Compensation of Req. Descriptions Using Linguistic Analysis*

CSS *Cascading Style Sheets*

DELA *Dictionnaires Electroniques du LADL*

EL *Entity Linking*

FA Funktionale Anforderungen

FAQ *Frequently Asked Questions*

FN *False negative*

FP *False positive*

GUI *Graphical User Interface*

HOTCoref *Higher Order Tree Coreference*

HTTP *Hypertext Transfer Protocol*

HTTPS *Hypertext Transfer Protocol Secure*

HTML *Hypertext Markup Language*

IE Informationsextraktion

IMAP *Internet Message Access Protocol*

IR *Information Retrieval*

IT Informationstechnologie

IEEE *Institute of Electrical and Electronics Engineers*

JRE *Java Runtime Environment*

JSON *JavaScript Object Notation*

JSP *JavaServer Pages*

JWNL *Java WordNet Library*

ML Maschinelles Lernen

MUC *Message Understanding Conference*

NE *Named Entities*

NER *Named Entity Recognition*

NFA Nicht-funktionale Anforderungen

NLARE *Natural Language Automatic Requirement Evaluator*

NLTK *Natural Language Toolkit*

NLP *Natural Language Processing*

NL *Natural Language*

NN Nomen

NP Nominalphrase

NUC *Next Unit of Computing*

OCL *Object Constraint Language*

OTF *On-The-Fly*

p *Precision*

PAS Prädikat-Argument-Struktur

PDC *Prague Dependency Treebank*

PDF *Portable Document Format*

PGP *Pretty Good Privacy*

POS *Part of Speech*

PP Präpositionalphrase

PCFG *Probabilistic Context-Free Grammars*

QuARS *Quality Analyzer for Requirement Specifications*

r *Recall*

RAM *Random-Access Memory*

RASP *Robust Accurate Statistical Parsing*

RDF *Resource Description Framework*

RE *Requirements Engineering*

REaCT *Requirements Extraction and Classification Tool*

RegEx *Regular Expressions*

RESI *Requirements Engineering Specification Improver*

SBD *Sentence Boundary Disambiguation*

SBVR *Semantic Business Vocabulary and Rules*

SEI *Software Engineering Institute*

SPARQL *SPARQL Protocol And RDF Query Language*

SREE *Systemized Requirements Engineering Environment*

SRL *Semantic Role Labeling*

SRS *Software Requirement Specification*

SSL *Service Specification Language*

TLS *Transport Layer Security*

TN *True negative*

TP *True positive*

SVM *Support Vector Machine*

TüBa-D/Z Tübinger Baumbank des Deutschen / Zeitungskorpus

UGC *User Generated Content*

UML *Unified Modeling Language*

URL *Uniform Resource Locator*

V Verb

VP Verbalphrase

VPE *Verbal Phrase Ellipsis*

WSD *Word Sense Disambiguation*

WSI *Word Sense Induction*

WSJ *Wall Street Journal*

XML *Extensible Markup Language*

YAGO *Yet Another Great Ontology*

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Motivation, Herausforderungen und Ziele

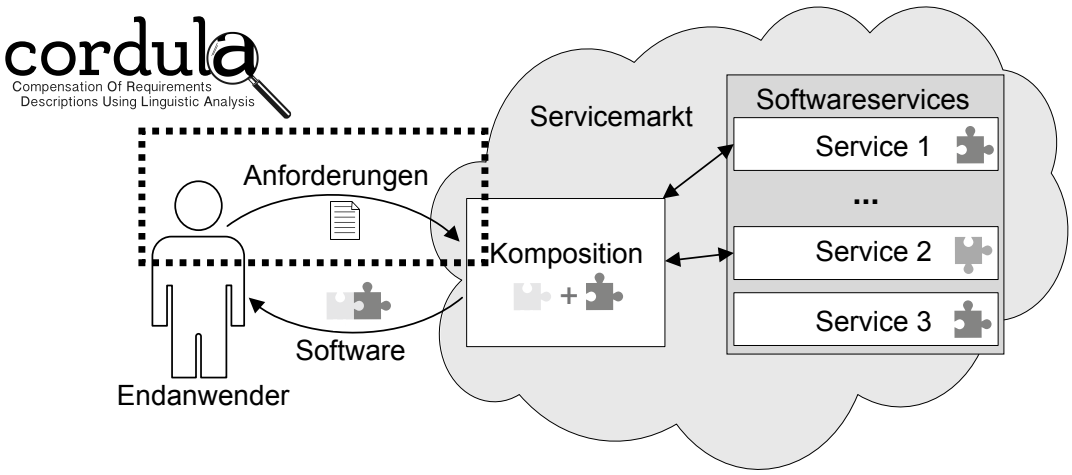
Im Projektmanagement besteht eine der zentralen Herausforderungen darin, dass Auftraggeber[1] und Auftragnehmer ein gemeinsames Verständnis für die Bestandteile des Projektauftrages entwickeln. Herausfordernd ist dabei insbesondere, dass Auftraggeber die Anforderungen überwiegend aus ihrer Perspektive beschreiben und damit einen hohen Freiheitsgrad in der Projektumsetzung zulassen. Darüber hinaus sind die Anforderungen natürlichsprachlich formuliert und daher oftmals mehrdeutig (auch: ambig) sowie in Teilen unvollständig. Dieser Herausforderung wird im Projektmanagement durch einen wechselseitigen Konsolidierungsprozess begegnet, in dem Rückfragen möglich sind und die Parteien sich auf ein gemeinsames Verständnis der Anforderungen einigen. Übertragen auf Softwareprojekte bedeutet das, dass *Stakeholder* (z. B. Endanwender) und Softwareentwickler sich hinsichtlich der Anforderungen an ein geplantes Softwareprodukt einig werden und damit gemeinsam die Gefahr einer nicht-deterministischen Entwicklung reduzieren. Dieser Konsolidierungsprozess wird dabei auch als Übersetzungsschritt zwischen der Anwender- und der Entwicklerperspektive bezeichnet und zielt insbesondere auf das Ausräumen von Ungewissheiten auf Seite der Entwickler ab. Er gilt allerdings auch als langwierig für *Stakeholder*, die sich Rückfragen gegenübersehen, von denen sie, zum Beispiel als resultatorientierte Endanwender, im Arbeitsalltag nicht direkt betroffen sind und deren Beantwortung sie leicht überfordern kann. Schlussendlich ist es aber aufgrund der direkten Kommunikation zwischen den Parteien ein zielführendes Vorgehen, das den Projekterfolg bereits in einem frühen Stadium sichern kann.

Es ist jedoch einschränkend zu sagen, dass ein solcher bidirektionaler Konsolidierungsprozess in der Softwareentwicklung nicht immer vorgesehen ist. So beispielsweise beim *On-The-Fly Computing* (auch: OTF-*Computing*), bei dem der individuelle Softwarebedarf von Endanwendern durch die automatische Komposition einzelner Softwareservices gedeckt wird. Die klassische Entwicklerrolle fällt somit faktisch weg, während die Notwendigkeit eines Konsolidierungsprozesses weiterhin besteht.

Im OTF-*Computing* werden Einzelservices von Softwareherstellern entwickelt und auf Servicemärkten bereitgestellt. Ziel ist es, geeignete (d. h. bedarfsgerechte und kompatible) Servicekompositionen für gegebene Softwareanforderungen zu generieren und diese den Endanwendern zur Verfügung zu stellen. Im Fokus dieser Arbeit stehen dabei natürlichsprachliche Softwareanforderungen, die von Endanwendern formuliert und an OTF-Anbieter als Anforderungsbeschreibung übergeben werden.

Hier setzt diese Arbeit an, indem sie sich mit Ungenauigkeiten frei formulierter Anforderungsbeschreibungen beim Softwareentwurf auseinandersetzt. Hierfür wird mit CORDULA (*Compensation of Requirements Descriptions Using Linguistic Analysis*) ein System entwickelt, dass sprachliche Unzulänglichkeiten in den Formulierungen unerfahrener Endanwender erkennt und ohne Rückfragen optimiert. Es ist dabei nicht das Ziel, aus Nutzereingaben direkt vollständige Softwarespezifikation abzuleiten, wie sie Programmierer erwarten würden. Vielmehr wird die Suche nach geeigneten Softwareservices zur Komposition unterstützt, indem CORDULA individuelle Anforderungsbeschreibungen in ihre kanonischen Kernfunktionalitäten überführt.

[1] Aus Gründen der Lesbarkeit wird auf eine geschlechtsspezifische Differenzierung verzichtet. Entsprechende Begriffe gelten im Sinne der Gleichbehandlung für beide Geschlechter.



Vision des OTF-*Computings*. In Anlehnung an Jungmann (2016, S. 20)

Diese Anbieter verarbeiten die Anforderungsbeschreibung und antworten mit einer Komposition von Softwareservices. Zwar existieren auch semi-formale und formale Möglichkeiten der Anforderungsspezifikation, diese stehen Endanwendern aber nicht zur Verfügung, da ihnen die Fachkenntnis fehlt um sie zu benutzen und sie demnach an der Teilname gehindert würden. Im Vergleich zu (semi-)formalen Ansätzen sind Ambiguitäten und Unvollständigkeit fester Bestandteil der natürlichen Sprache – sie dienen der Sprachökonomie, begünstigen aber Missverständnisse. So auch in der maschinellen Verarbeitung von Anforderungsbeschreibungen im OTF-*Computing*: Ambiguitäten und Unvollständigkeit sind in der Vision einer automatisierten, hochperformanten Komposition hinderlich, da sie Ungewissheit erzeugen und damit den Gesamtprozess verlangsamen, wenn nicht sogar schädigen. Mangels klassischem Konsolidierungsprozess sind integrierte Vorgehensweisen erforderlich, die sich der Kompensation dieser Ungewissheiten im OTF-*Computing* widmen.

Herausforderungen

Die von Endanwendern übermittelten Anforderungsbeschreibungen sind hinsichtlich Informationsgehalt und -güte zu beurteilen. Zum einen ist bezüglich des Informationsgehalts zu erwarten, dass (aus der Entwicklungsperspektive) notwendige Angaben fehlen während nebensächliche Angaben vorliegen, welche es aber im Sinne einer performanten Servicekomposition frühzeitig zu erkennen und zu filtern gilt:

Beispiel (Anforderungsbeschreibung)
„Since I want to listen to music on the go, I need a software which can be only play mp3 files on Android but the user not allow to copy or send with bluetooth“[2]

Zum anderen müssen vorhandene, notwendige Angaben wiederum hinsichtlich der Informationsgüte bewertet werden, wobei der Fokus in dieser Arbeit auf ambigen („*send*“ hat acht Lesarten[3]) und unvollständigen Angaben (Was soll wohin nicht

[2] In Anlehnung an `http://qr.ae/TbZ3yi` (Stand: 19.05.2017).
[3] Siehe weiterführend: `http://wordnetweb.princeton.edu/perl/webwn?s=send` (Stand: 19.05.2017).

kopiert werden?) und deren Erkennung und Kompensation liegt. Bestehende Ansätze zur Prävention, Erkennung und Kompensation konzentrieren sich dabei zumeist auf einzelne Ausprägungen von Ambiguitäten bzw. Unvollständigkeit und sind oftmals nicht als softwaregestützte Verfahren konzipiert, sondern existieren als Lesetechniken, Checklisten oder *Review*-Prozesse. Softwaregestützte Verfahren setzten vielfach Einschränkungen voraus, die den Umfang der natürlichen Sprache, beispielsweise den Wortschatz, begrenzen oder zusätzliche Ressourcen, wie Korpora oder strukturierte Anforderungsdokumente, benötigen, die jedoch nicht existieren. Für das Anwendungsszenario des OTF-*Computings* sind diese bestehenden Ansätze ungeeignet, da sie nicht ohne Benutzerinteraktion anzuwenden und nicht für die Anbindung an Drittsysteme vorgesehen sind sowie oftmals nicht ohne weitere Einschränkungen auf natürlicher Sprache arbeiten können. Auch erforderliche Zusatzinformationen wie Klassendiagramme und umfangreiche domänenspezifische Korpora sind im OTF-Szenario seitens der Endanwender nicht zu erwarten. Darüber hinaus ist die Performanz bisheriger Ansätze zu hinterfragen, da keine der Methoden prüft, ob die eigene Anwendung überhaupt notwendig ist. So könnte die Anwendung der Unvollständigkeitskompensation beispielsweise übersprungen werden, wenn keine Hinweise auf Unvollständigkeit vorliegen. Darüber hinaus fehlt bisher eine Interaktion zwischen den Erkennungs- und Kompensationsansätzen sowohl hinsichtlich Synergien als auch schädlicher Auswirkungen der eigenen Aktivität auf die Anforderungsbeschreibung: Hier fehlt es an definierten prozeduralen Abläufen, welche die Notwendigkeit der Methodenausführung erkennen und diese so flexibel steuern, dass Informationen untereinander geteilt werden können.

Zielsetzung

Ziel ist die automatische Erkennung und Kompensation von Ambiguität und Unvollständigkeit in natürlichsprachlichen Anforderungsbeschreibungen. Dies geschieht unter Berücksichtigung der Anforderungen des OTF-*Computings* bezüglich hoher Performanz, Flexibilität und niedriger Benutzerinteraktion. Diesbezüglich ist die Identifikation geeigneter Verfahren zur Erkennung und Kompensation lexikalischer, syntaktischer und referentieller Ambiguität sowie Unvollständigkeit in natürlichsprachlichen Anforderungsbeschreibungen ein vorgelagerter Schritt (s. Kapitel 3). Auf diesen Schritt folgt die Entwicklung von Strategien zur bedarfsgerechten und performanten Steuerung der geeigneten Erkennungs- und Kompensationsverfahren (s. Abschnitt 5.2). Diese Strategien ermöglichen es, einzelne Verfahren bzw. deren Ausführung zu überwachen, Ergebnisse abzugleichen und Synergieeffekte zu nutzen. Hierzu ist, im Sinne einer hohen Performanz, die Entwicklung von kontextsensitiven Indikatoren zur Ermittlung des Kompensationsbedarfs erforderlich, die einzelne Strategien aktivieren können (s. Abschnitt 5.3). Die Besonderheit bei der Indikatorenentwicklung ist, dass diese nicht auf die Ergebnisse der nachgelagerten Erkennungs- und Kompensationsverfahren zurückgreifen können und daher überwiegend Textmerkmale heranziehen, die ebenfalls erst im Rahmen dieser Arbeit zu identifizieren und zu systematisieren sind. Dies wiederum setzt domänenspezifische Ressourcen voraus, die in Teilen noch nicht existieren und daher erstellt (z. B. PAS-Korpus) bzw. zusammengetragen (z. B. Anforderungsbeschreibungskorpus) werden müssen (s. Kapitel 6).

Die Anwendbarkeit der Verfahren, Indikatoren und Strategien gilt es darüber hinaus anhand eines Prototyps zu evaluieren, wobei dies die Konzeption (s. Kapitel 5) und Implementierung (s. Kapitel 7) eines strategiebasierten Anforderungskompensationssystems (CORDULA) zur Aufnahme, Verarbeitung, Kompensation und Strukturierung unstrukturierter Anforderungsbeschreibungen voraussetzt. Im Folgenden werden diesbezüglich zunächst bestehende Definitionen und Ansätze diskutiert, um die eigene Arbeit in den Kontext der existierenden Forschung einzubetten (s. Teil I).

Teil I

Grundlagen und Stand der Forschung

Anforderungserhebung und Dokumentation

1

In diesem Kapitel werden grundlegende Begrifflichkeiten wie *Stakeholder* (s. Abschnitt 1.1) sowie „Anforderung“ erläutert (s. Abschnitt 1.2), wobei insbesondere eine Unterteilung in funktionale und nicht-funktionale Anforderungen vorzunehmen ist. Darauf folgt in Abschnitt 1.3 die Betrachtung von Methoden und Techniken der Anforderungsdokumentation. Abschließend wird in Abschnitt 1.4 der Begriff der „Anforderungsbeschreibung“ definiert, welcher für den weiteren Verlauf der Arbeit als Sonderform der Anforderungsdokumentation von Bedeutung ist.

1.1 Anforderungsquellen

Im betriebswirtschaftlichen Kontext werden alle „internen und externen Personengruppen, die von den unternehmerischen Tätigkeiten gegenwärtig oder in Zukunft direkt oder indirekt betroffen sind“ (Springer Gabler, 2015) als *Stakeholder* bezeichnet. Dieses Begriffsverständnis lässt sich auch auf Softwareentwicklungsprojekte anwenden, da hier ebenfalls die direkten sowie indirekten Interessen und Bedürfnisse mehrerer natürlicher und juristischer Personen(-gruppen) zu berücksichtigen sind[4] (Balzert, 2009; Fahney et al., 2012). Somit sind *Stakeholder* wichtige Informationsquellen für Anforderungen (sog. Anforderungsquellen) und definieren die Rahmenbedingungen eines zu entwickelnden Systems (Pohl und Rupp, 2015, S. 21 ff.).

Die Interessen und damit auch die Anforderungen der jeweiligen *Stakeholder* sind untereinander nicht immer zu vereinbaren (Pohl und Rupp, 2015, S. 22). Grechenig (2010) weist daher darauf hin, dass es wichtig ist, Anforderungen zu priorisieren und einzelne *Stakeholder* in Leitungspositionen mit einem Mandat zur Konfliktlösung zu versehen. Dieser Aspekt des Rangs wird auch von Schwinn (2011, S. 170) aufgegriffen, der die Rolle eines „Chef-Planers“ empfiehlt, welcher im Wesentlichen inhaltliche und formelle Gesamtverantwortung[5] für ein IT-Projekt übernimmt.

Darüber hinaus können sich Anforderungen verändern, wegfallen oder hinzukommen. Ebenso werden *Stakeholder* unter Umständen im Projektverlauf ausscheiden bzw. erst später identifiziert werden. Das Übersehen von *Stakeholdern* hat hierbei „häufig zur Konsequenz, dass Anforderungen an das System lückenhaft sind“ (Pohl und Rupp, 2015, S. 22) oder sogar gänzlich fehlen.

[4] Eine ähnliche Auffassung findet sich im *Systems Engineering* (Haberfellner et al., 1994, S. 186 f.).

[5] Das bedeutet nach Tiemeyer (2013, S. 246 f.) unter anderem, Verantwortung für das Erreichen der formulierten Projektziele und das Einhalten definierter Zeit- und Kostenrahmen zu übernehmen. Darüber hinaus gilt es, den effizienten Einsatz der Projektressourcen sowie die Einhaltung der gesetzten Qualitätsanforderungen zu überwachen.

Weiterhin ist zu beachten, dass jede *Stakeholder*-Gruppe eine eigene Sicht auf Funktionen hat (Grechenig, 2010, S. 143 f.). Diese Sichtweise geht einher mit unterschiedlichen Rollen, die innerhalb eines Softwareentwicklungsprojekts von *Stakeholdern* eingenommen werden (Robertson und Robertson, 2012, S. 44 ff.). Ebenso wird dieses Rollenverständnis in der Begriffsdefinition von Pohl (2007, S. 65) bzw. Robertson und Robertson (2006) deutlich, an der sich diese Arbeit orientiert:

Definition 1.1.1 (Stakeholder)
Ein Stakeholder ist eine Person oder eine Organisation, die ein potenzielles Interesse an dem zukünftigen System hat und somit auch Anforderungen an das System stellt. Eine Person kann dabei die Interessen von mehreren Personen oder Organisationen vertreten und somit gleichzeitig mehrere Rollen einnehmen (z. B. Kunde und Endanwender).

Die verschiedenen *Stakeholder* und deren Wirkungsbereiche sind in Abbildung 1.1 dargestellt. Eine der bekannteren Rollen ist beispielsweise die des Endanwenders (engl. *user*), der mit der Software arbeiten wird und an ihrer benutzerfreundlichen Bedienung interessiert ist. Weniger offensichtlich ist die Rolle des betriebsinternen Datenschutzbeauftragten, dessen Fokus auf der rechtskonformen Datenspeicherung und -verarbeitung liegt (Robertson und Robertson, 2012, S. 44 ff.).

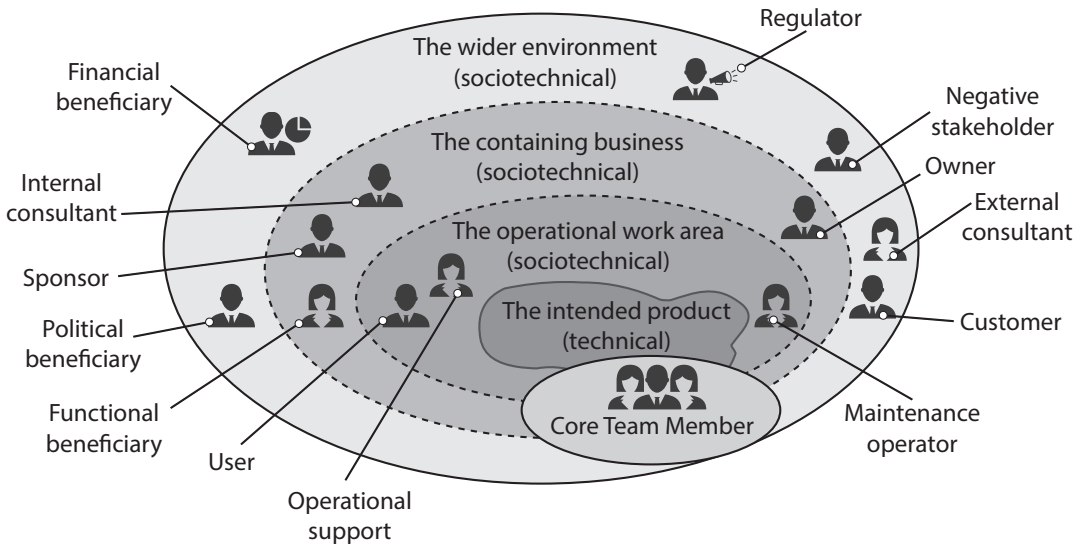


Abbildung 1.1: *Stakeholder map.*
In Anlehnung an Robertson und Robertson (2012, S. 45)

Im Zentrum von Abbildung 1.1 steht das Softwareprodukt, wobei die gewölbte Umrandung verdeutlicht, dass sich die Form eines Produkts im Laufe des Projektzeitraums ändern kann bzw. zum Zeitpunkt der Projektinitiierung nicht abschließend zu definieren ist. Um das Produkt herum befinden sich drei Wirkungsbereiche, wovon der erste Bereich („*The operational work area*“) alle *Stakeholder* enthält, die direkt mit dem Produkt agieren. Der zweite Bereich („*The containing business*“) enthält Gruppen, die von dem Produkt in einer beliebigen Art profitieren und der dritte Bereich („*The wider environment*“) vereinigt *Stakeholder*, die darüber hinaus Einfluss auf oder Interesse an dem Produkt haben (Robertson und Robertson, 2012, S. 45).

Die Tatsache, dass *Stakeholder* mehrere Rollen innehaben können[6], wird in Abbildung 1.1 deutlich: So werden Kunden (engl. *customer*) nach Fertigstellung der Software zu Eigentümern (engl. *owner*) und darüber hinaus, sofern die Software zur eigenen Benutzung angeschafft wird, auch zum Endanwender. Grechenig (2010, S. 143) weist in diesem Zusammenhang darauf hin, dass viele *Stakeholder* nicht zwangsläufig Techniker sind und somit weder den „typischen Techniker-lingo und technische Systembeschreibungen“ (Grechenig, 2010, S. 143) verstehen, noch ihre Anforderungen in dieser Form verschriftlichen können – im Gegensatz zum „*Core Team*“ aus Abbildung 1.1, welches an den Entwicklungsarbeiten des Produkts beteiligt ist und deshalb dieselbe Fachsprache spricht. Aus diesem Grund werden oftmals für die unterschiedlichen *Stakeholder* spezielle Anforderungsdokumentations- und Erhebungstechniken genutzt (Grechenig, 2010, S. 143).

In dieser Arbeit wird, der Terminologie des Sonderforschungsbereichs 901 folgend, der Begriff „Endanwender“ wie folgt verwendet:

Definition 1.1.2 (Endanwender)
Endanwender sind Stakeholder, die ein Anwendungsinteresse an Softwareprojekten haben. Sie haben eine vage Vorstellung ihrer individuellen Softwareanforderungen, können diese aber aufgrund fehlender Erfahrung und ohne Expertenhilfe nicht formal dokumentieren.

Die Endanwender werden besonders hervorgehoben, da sie den Ausgangspunkt der informalen Anforderungsdokumentation darstellen und damit auch im Fokus dieser Arbeit stehen. Im weiteren Verlauf wird sowohl der Begriff der *Stakeholder* im Allgemeinen, als auch der des Endanwenders im Speziellen genutzt – Letzterer insbesondere im Zusammenhang mit den in Abschnitt 1.4 erläuterten Anforderungsbeschreibungen.

1.2 Anforderungen an Softwaresysteme

Für den Begriff der „Anforderung“ (engl. *requirement*) existieren im Informatikkontext mehrere Definitionsansätze (z. B. Rupp, 2014; Sommerville, 2011; Balzert, 2009; Pohl, 2007). Nach Balzert (2009, S. 455) legen Anforderungen fest, „was man von einem Softwaresystem als Eigenschaften erwartet“. Unter „man“ sind dabei alle *Stakeholder* zu verstehen.

Als „Eigenschaften“ identifiziert Balzert (2009) neben den primären funktionalen und nicht-funktionalen Anforderungen die Visionen und Ziele, die am Anfang einer Produktspezifikation stehen und welche die Rahmenbedingungen für das System und die Entwicklung definieren. Laut IEEE (1991) ist eine Anforderung[7]:

1. „Eine Eigenschaft oder Fähigkeit, die von einem Benutzer (Person oder System) zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird.

[6] Eine ähnliche Auffassung wird auch im *Systems Engineering* vertreten (Haberfellner et al., 1994, S. 186 f., 311; Gausemeier et al., 2013, S. 29).
[7] Übersetzung entnommen aus Rupp (2014, S. 13 f.).

2. Eine Eigenschaft oder Fähigkeit, die ein System oder Teilsystem erfüllen oder besitzen muss, um einen Vertrag, eine Norm, eine Spezifikation oder andere, formell vorgegebene Dokumente zu erfüllen.

3. Eine dokumentierte Repräsentation einer Eigenschaft oder Fähigkeit gemäß (1) oder (2)“.

Diese Definition ergänzt die allgemeine Auffassung einer Anforderung von Balzert (2009) um einen wesentlichen Aspekt: Die „dokumentierte Repräsentation“ (s. Abschnitt 1.3). Eine isolierte dokumentierte Anforderung – das „erreichte inhaltliche Verständnis über eine Anforderung“ (Pohl, 2007, S. 47) – wird auch als Anforderungsartefakt bezeichnet und zusammen mit anderen derartigen Artefakten in Anforderungsdokumenten verwaltet (Pohl, 2007, S. 14).

1.2.1 Funktionale Anforderungen

Funktionale Anforderungen (FA) ergeben sich aus dem gewünschten Softwarenutzen der jeweiligen *Stakeholder*. Demnach ist eine funktionale Anforderung das, was eine Software im individuellen (Geschäfts-)Kontext nützlich macht (Rupp, 2014) bzw. können muss (Schneider, 1998, S. 33) und somit auch das, was eine Software in der Lage sein muss, an Funktionalität zu erbringen (Sommerville, 2011; Balzert, 2009; Schienmann, 2002; IEEE, 1991).

Beispiel 1.2.1 (Funktionale Anforderung)

(a) „Alle Druckaufträge an das System werden vom Benutzer getätigt.“

(b) „Der Benutzer kann Druckaufträge erstellen und konfigurieren.“

(c) „Wird die ‚Drucken‘-Option gewählt, öffnet das System den ‚Drucken‘-Dialog und fordert den Benutzer zur Eingabe der Seitenzahlen auf.“

Das Beispiel 1.2.1 zeigt drei frei formulierte FA, die sich hinsichtlich des Abstraktionsgrades unterscheiden. Bray (2002, S. 15 ff.) weist darauf hin, dass Anforderungen auf verschiedenen Abstraktionsebenen ausgedrückt werden können, die Wahl der Ebene aber dem Verfasser obliegt. Eine Endanwenderanforderung kann zum Beispiel sehr allgemein gehalten sein (Pohl, 2007, S. 15).

1.2.2 Nicht-funktionale Anforderungen

Anforderungen, die keine FA darstellen, werden in der Fachliteratur unterschiedlich behandelt: Traditionell ist der Begriff „nicht-funktionale Anforderungen“ (NFA) in der Literatur etabliert (Rupp, 2014; Sommerville, 2011; Balzert, 2009; Pohl, 2007). Vereinfacht ausgedrückt sind NFA damit alle Anforderungen, die nicht funktional sind (Rupp, 2014). So zählt IEEE (1991) beispielsweise „*design requirement*, *interface requirement*“ und „*performance requirement*“ als kontrastierende Begriffe zu funktionalen Anforderungen auf und macht sie damit zum Bestandteil der NFA. Allerdings widersprechen beispielsweise Balzert (2009) und Sommerville (2011) dieser

strikten Definition, indem sie anführen, dass NFA auch die FA betreffen können (beispielsweise Zuverlässigkeit, Sicherheit oder Internationalisierung). Denn oftmals beziehen sich NFA auf das System als Ganzes (vgl. Beispiel 1.2.2).

Beispiel 1.2.2 (NFA) *„Das System muss sicher und schnell sein.“*

Pohl (2007) sieht ebenfalls in NFA Eigenschaften, die das Gesamtsystem betreffen und widerspricht dennoch diesem Begriffsverständnis entschieden, indem er anführt, „dass es sich bei vielen [...][NFA] um unterspezifizierte Anforderungen handelt“[8] (Pohl, 2007, S. 16). Demnach ist die Klasse der NFA in zwei Unterklassen aufzuteilen (Pohl, 2007, S. 16):

1. **Unterspezifizierte funktionale Anforderungen**
   Durch Spezifizierung lässt sich dieser Anforderungstyp in FA überführen.

2. **Qualitätsanforderungen**
   Dieser Anforderungstyp bezieht sich auf qualitative Eigenschaften, die sowohl das Gesamtsystem, als auch einzelne Funktionen und Funktionsgruppen betreffen können. Sie können in der Regel nicht durch FA spezifiziert werden.

Im weiteren Verlauf der Arbeit wird bei Qualitätsanforderungen dem Begriffsverständnis von Pohl (2007) gefolgt.

1.2.3 Rahmenbedingungen

Neben der inhaltlichen Betrachtung von Anforderungen ist beispielsweise die Perspektive der „Veränderbarkeit“ zu berücksichtigen: So können organisatorische oder technologische Anforderungen (z. B. seitens des Gesetzgebers) existieren, die als Rahmenbedingungen (engl. *constraints*) zu berücksichtigen sind, Einfluss auf die Funktionen nehmen und dennoch nicht verändert werden können (Rupp, 2014; Balzert, 2009). Diesbezüglich kann nach Pohl (2007) generell zwischen Rahmenbedingungen, die das zu entwickelnde System betreffen und jenen, die den Entwicklungsprozess tangieren, unterschieden werden (vgl. Beispiel 1.2.3).

Beispiel 1.2.3 (Rahmenbedingung)
„Die Entwicklung des Gesamtsystems darf einen maximalen Personalaufwand von 24 Monaten nicht überschreiten.“

1.3 Anforderungsdokumentation

Unter dem Begriff „Anforderungsdokumentation“ wird ein Dokument verstanden, das als Teil der Vertragsgrundlage eines Softwareprojekts existiert und die Ergebnisse der Anforderungsanalyse – unabhängig von den genutzten Methoden – kumuliert (Grechenig, 2010, S. 175 ff.). Eine Anforderungsdokumentation stellt somit eine Zusammenfassung aller identifizierten Anforderungen dar. In der Literatur existiert

[8] Der Begriff der Ungenauigkeit wird in Kapitel 2 definiert.

keine einheitliche Begriffsverwendung, sodass auch „Spezifikation“, „Anforderungsanalysedokument“, „*Software-Requirements-Specification* (SRS)“ und „Lastenheft/Pflichtenheft“ sowohl in verschiedenen Kontexten als auch synonym verwendet werden (Baumgartner et al., 2013; Grande, 2011; Grechenig, 2010). Im Rahmen dieser Arbeit wird deshalb die Anforderungsdokumentation in Anlehnung an Pohl (2007, S. 43, 229 ff.) sowie Grechenig (2010, S. 175) wie folgt definiert:

Definition 1.3.1 (Anforderungsdokumentation)
Als Anforderungsdokumentation wird die Tätigkeit bezeichnet, informal vorliegende Informationen (z. B. Interviewprotokolle, Notizen, Skizzen) mittels (vordefinierter) Dokumentationstechniken festzuhalten (z. B. schriftlich). Die Art und Weise der Übertragung, die Methode sowie die Qualitätskriterien ergeben sich aus vorgegebenen Dokumentationsvorschriften, die wiederum aus Normen ableitbar sind oder von leitenden Stakeholdern initial formuliert werden.

Der primäre Nutzen der Anforderungsdokumentation ist die systematische, strukturierte Verwaltung von Anforderungen, die aus verschiedenen Quellen und in unterschiedlicher Qualität gewonnen wurden. Sie werden somit strukturell und inhaltlich aufgewertet und stehen als Informations- und Wissensbasis aber auch als Konstruktions-, Verhandlungs- und Vertragsgrundlage zur Verfügung. Auch dann, wenn einzelne *Stakeholder* gegebenenfalls nicht mehr greifbar sind.

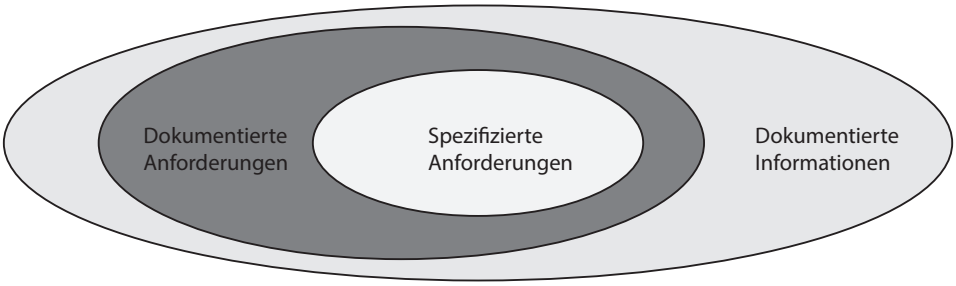


Abbildung 1.2: Dokumentation vs. Spezifikation. Laut Pohl (2007, S. 220)

Wie Abbildung 1.2 zeigt, stellen dokumentierte Informationen die Ausgangslage dar. Sie wurden beispielsweise während Interviews mit *Stakeholdern* verschriftlicht.

Definition 1.3.2 (Dokumentierte Anforderung)
Dokumentierte Informationen, die den Dokumentationsvorschriften entsprechen, stellen dokumentierte Anforderungen dar. Die Dokumentationsvorschriften ergeben sich wiederum aus Normen oder werden von leitenden Stakeholdern initial formuliert.

Analog verhält es sich mit dokumentierten und spezifizierten Anforderungen. Anforderungsdokumentationen sind demnach abzugrenzen von Anforderungsspezifikationen, die sich aus Anforderungsdokumenten ergeben, wenn sie den vorgegebenen Spezifikationsvorschriften entsprechen (Pohl, 2007, S. 44).

Definition 1.3.3 (Spezifizierte Anforderung)
Dokumentierte Anforderungen, die den Spezifikationsvorschriften entsprechen, stellen spezifizierte Anforderungen dar. Die Spezifikationsvorschriften ergeben sich wiederum aus Normen oder werden initial von leitenden Stakeholdern formuliert.

Zur Dokumentation bzw. Spezifikation von Anforderungen stehen Methoden zur Verfügung, die sich unter anderem im Formalisierungsgrad stark unterscheiden (Wiegers, 2005, S. 153). So können sowohl informale Methoden, wie die natürliche Sprache, als auch semi-formale und formale Verfahren eingesetzt werden (Tiemeyer, 2013, S. 327 f.). Eine strikte Trennung zwischen den Ansätzen existiert nicht, sodass in Anforderungsdokumentationen oftmals mindestens zwei Varianten zur Anwendung kommen (Laplante, 2013, S. 83).

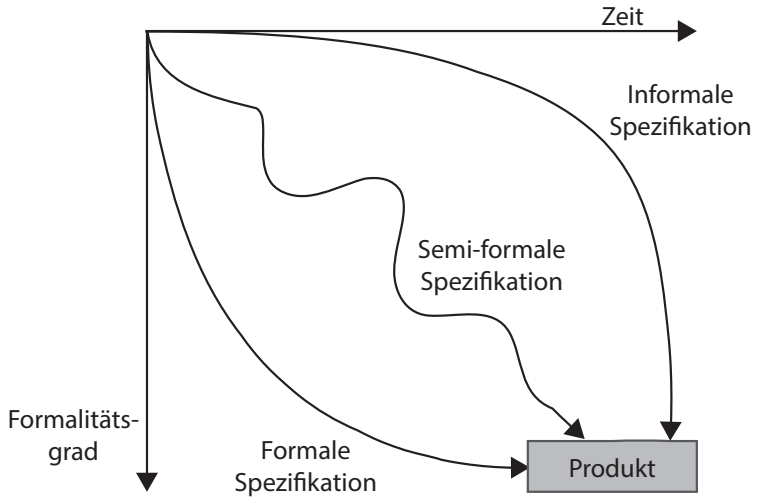


Abbildung 1.3: Methoden und ihre Formalisierungsgrade.
In Anlehnung an Beneken (o. D.)

Wie Abbildung 1.3 illustriert, sind alle genannten Methoden – ausgehend von einer Idee oder Problemstellung – für die angestrebte Produktentwicklung zielführend. Dabei wird ersichtlich, dass im Laufe der Zeit final ein einheitlicher Formalisierungsgrad beim Endprodukt erreicht wird.

Trotz höherem Zeitaufwand ist es laut Brugger (2009, S. 237) wünschenswert und der Kreativität zuträglich, dass zu Beginn eines Projekts informale Methoden eingesetzt werden. Erst wenn konkrete Handlungspläne vorliegen, kann in Abhängigkeit der Projektgröße zu formalen oder semi-formalen Methoden gewechselt werden (Brugger, 2009, S. 237). Wann genau dieser Paradigmenwechsel von der informalen Produktidee bzw. Problemstellung zur formalen Spezifikation des Produkts vollzogen werden kann, hängt von projektspezifischen Faktoren ab (z. B. Gesamtprojektkomplexität).

Am Beispiel der formalen Spezifikation kann aufgezeigt werden, dass ein hoher Grad an Formalisierung schon zu einem sehr frühen Zeitpunkt erreicht wird. Dies ist bei informalen Spezifikationen erst wesentlich später der Fall (vgl. Abbildung 1.3), da beim Einsatz informaler Methoden kein komplettes Regelwerk bereitsteht (Tiemeyer, 2013, S. 328). Vielmehr werden die Ausdrucksmöglichkeiten der *Stakeholder* bewusst kaum eingeschränkt. Beispiele für die informale Methode sind die natürliche Sprache und Schaubilder wie sogenannte *Box-and-Arrow*-Diagramme (Tiemeyer, 2013, S. 328). Im Gegensatz dazu unterliegen semi-formale Verfahren einer vordefinierten, eindeutigen Syntax. Die Repräsentation kann eine „graphische Notation sein, mit präzisen Regeln zur Erstellung der Diagramme oder eine textuelle Notation mit ähnlichen Regeln“ (Tiemeyer, 2013, S. 328). Ein Beispiel für semi-formale Sprachen ist die

Unified Modeling Language (UML), bei der „die Syntax [...] größtenteils formal, die Semantik jedoch zum überwiegenden Teil natürlichsprachlich spezifiziert ist“ (Pohl, 2007, S. 290). Formale Verfahren übersteigen diesen Ansatz, indem Anforderungen mittels formal spezifizierter Syntax und Semantik modelliert werden (Pohl, 2007, S. 290). Nach Hußmann (1993, S. 5) sind solche Dokumentationstechniken kaum intuitiv zu verstehen, bieten aber wesentliche Vorteile, insbesondere was ihre Präzision und Verifizierbarkeit betrifft.

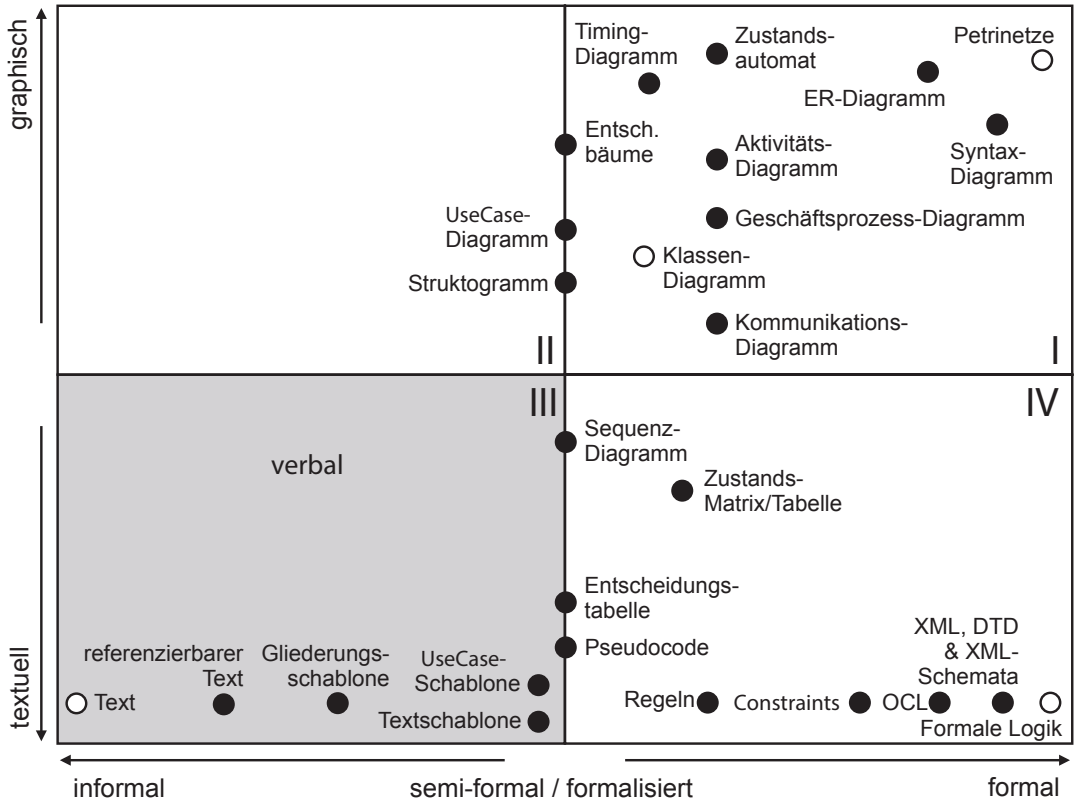


Abbildung 1.4: Klassifikation der Dokumentationstechniken.
In Anlehnung an Balzert (2009, S. 101)

Balzert (2009, S. 100 f.) klassifiziert verschiedene Dokumentationstechniken mittels der konträren Kategorien „textuell – graphisch“ und „informal – formal“. Textuell umfasst dabei die Darstellung durch natürlichsprachliche Texte – das heißt durch eine „schriftlich fixierte im Wortlaut festgelegte, inhaltlich zusammenhängende Folge von Aussagen“ (Dudenredaktion, 2017c). Werden Informationen graphisch dargestellt, geschieht dies durch Symbole, Linien und zusätzliche, textuelle Annotationen (Balzert, 2009, S. 109). Der Grad der Formalisierung gibt die Formalisierung mittels definierter Strukturvorgaben an. Diese Vorgaben existieren sowohl für textuelle als auch für graphische Dokumentationstechniken (Balzert, 2009, S. 109). Die sich durch die Kategorien ergebenden vier Quadranten sind in Abbildung 1.4 dargestellt.

Die natürliche Sprache („Text“) ist als textuelle, informale Technik im dritten Quadranten abgebildet. Im Gegensatz dazu ist die formale Logik im vierten Quadranten anzusiedeln. Sie weist einen hohen Formalisierungsgrad auf, basiert aber ebenfalls auf einer textuellen Darstellung. Klassendiagramme und Petrinetze sind als Beispiele für

graphische Techniken im ersten Quadranten anzuführen, wobei Petrinetze wesentlich formaler definiert sind.

In den folgenden Abschnitten werden ausgewählte Dokumentationstechniken unter dem Aspekt ihres methodischen Formalisierungsgrades vorgestellt und ihre jeweiligen Vor- und Nachteile diskutiert.

1.3.1 Informale Anforderungsdokumentation

Die Anforderungsdokumentation mittels natürlicher Sprache ist eine in der Praxis weit verbreitete Technik (Rupp, 2014; Sommerville, 2011; Balzert, 2009; IEEE, 1998). Unter „natürlicher Sprache“ wird dabei die „Umgangssprache als Kommunikationsmittel“ (Lewandowski, 1994, S. 740 f.) verstanden, die sowohl Hochsprache, Alltagssprache sowie Dialekte und Sprachvarianten umfasst – eine „historisch entwickelte, regionale und sozial geschichtete Sprache“ (Bußmann, 1983, S. 342), die von künstlichen Sprachsystemen (Kunstsprachen, Weltsprachen) abzugrenzen ist. Im Gegensatz zu Kunstsprachen ist die natürliche Sprache durch ihre historische Wandelbarkeit und lexikalische sowie strukturelle Ambiguität geprägt (Bußmann, 1983, S. 279, 342 f.). Natürliche Sprache wird im weiteren Verlauf dieser Arbeit in Anlehnung an Lewandowski (1994, S. 740) wie folgt definiert:

Definition 1.3.4 (Natürliche Sprache)
Natürliche Sprache als Kommunikationsmittel ist die Umgangssprache. Eine historisch gewachsene Sprache, die mehr oder weniger standardisierte Varietäten wie Hochsprache vs. Umgangssprache, Dialekte oder Regionalsprachen aufweist. Aus logischer Sicht ist sie voll von historischen Zufälligkeiten, von Mehrdeutigkeiten und Inkonsequenzen; sie ist plastisch und variabel, pragmatisch, offen und dynamisch.

Wie deutlich wird, besitzt die natürliche Sprache Eigenschaften, die insbesondere im Kontext der Anforderungsdokumentation zu nennen sind. Natürliche Sprache ...

- „tritt in zeitlicher, regionaler und sozialer Variation auf;
- erfüllt eine Reihe von Funktionen und dient nicht nur der Repräsentation von Sachverhalten;
- ist nicht explizit, sondern implizit in dem Sinne, [dass] beim Verstehen von Äußerungen besonders beim Übergang von der wörtlichen Bedeutung zu einer intendierten Bedeutung Präsuppositionen und Implikationen gelten [...];
- ist in ihrem Gebrauch abhängig von Kontexten aller Art, von räumlicher und zeitlicher Deixis;
- ist [...] oft in der Weise mehrdeutig, [dass] Ausdrücke in verschiedenen Kontexten Verschiedenes bedeuten;
- ist in ihren referierenden Ausdrücken bis zu einem gewissen Ausmaß vage [...];
- ist syntaktisch nicht immer wohlgeformt (Ellipsen usw.)“.
  (Lewandowski, 1994, S. 740 f.)

Wird die **natürliche Sprache** differenzierter betrachtet, kristallisieren sich Teilmengen heraus, die sich zum Beispiel in der Entstehung, Verwendung und Verbreitung voneinander unterscheiden (vgl. Abbildung 1.5).

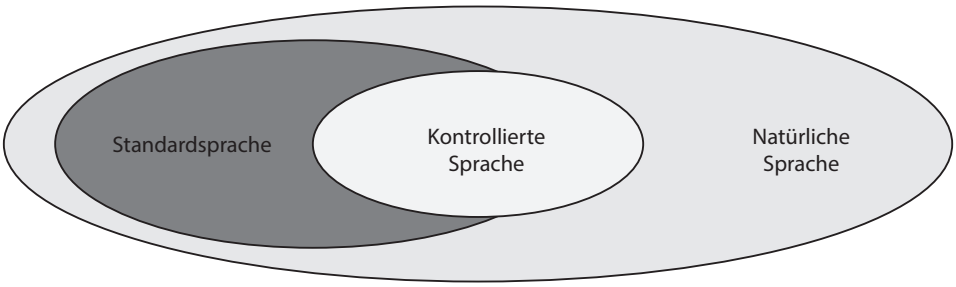


Abbildung 1.5: Teilmengen natürlicher Sprache.
In Anlehnung an Schwitter (1998, S. 57)

Die größte Teilmenge der natürlichen Sprache stellt die zur öffentlichen Kommunikation genutzte **Standardsprache** (auch: Gemeinsprache) dar. Hierbei handelt es sich um eine „deskriptive Bezeichnung für die historisch legitimierte, überregionale, mündliche und schriftliche Sprachform der sozialen Mittel- beziehungsweise Oberschicht“ (Bußmann, 1983, S. 502) und somit um eine „über den Mundarten, lokalen Umgangssprachen und Gruppensprachen stehende, allgemein verbindliche Sprachform“ (Dudenredaktion, 2017b). Ihre Normierung wird durch das Bildungssystem, Medien und Institutionen kontrolliert (Bußmann, 1983, S. 502)[9].

Fachsprachen unterscheiden sich von der Standardsprache „vor allem durch einen fachspezifischen differenzierten Wortschatz mit Tendenz zu fester bzw. normierter Terminologie“ (Bußmann, 1983, S. 137). Allerdings weist Lehrndorfer (1996, S. 37) darauf hin, dass die „langezeit populäre These, das Wesentliche einer Fachsprache liege in den Fachworten und nicht in der Syntax, [...] inzwischen relativiert werden [kann]“. Darüber hinaus ist die Fachsprache nicht disjunkt von der Standardsprache, da sie sich zum einen der Wörter und Grammatik der Standardsprache bedient und zum anderen ein Austausch mit ihr stattfindet, „da häufig Fachgebiete von gestern zum Populärwissen von heute avancieren“ (Lehrndorfer, 1996, S. 25).

Hingegen ist **kontrollierte Sprache** (auch: Normsprache) der Versuch, Fachsprache einer „Kontrolle“ zu unterwerfen und sie damit der „Intuition, Situation und Sprachregister“ (Lehrndorfer, 1996, S. 40) zu entziehen. Dabei sind „kontrollierte Sprachen [...] keine Kunstsprachen, wie zum Beispiel das Esperanto, sondern der Zuschnitt einer bestehenden Sprache auf eine bestimmte Anwendung und seine Benutzer“ (Ferlein und Hartge, 2008, S. 40). Sie werden genutzt, um den Austausch über komplexe Themen zu erleichtern, technische Dokumentationen verständlicher zu gestalten oder Aussagen leichter überprüfen zu können.

Kontrollierte Sprachen stellen somit eine sehr kleine Teilmenge dar, die eine große Schnittmenge mit Fachsprachen aufweist und einzelne Charakteristika der Standardsprache, insbesondere den normativen Charakter, besitzt (vgl. Abbildung 1.5).

[9] Der Begriff der Standardsprache ist in der Varietätenlinguistik umstritten und die angeführte Definition nicht abschließend. Im Rahmen dieser Arbeit ist die Definition hinreichend, da sie den normativen Charakter sowie ihre gesellschaftliche Relevanz aufzeigt.

Kontrollierte Sprache lässt sich in Anlehnung an Pohl (2008, S. 710) und Lehrndorfer (1996, S. 40 ff.) wie folgt definieren:

Definition 1.3.5 (Kontrollierte Sprache)
Eine kontrollierte Sprache ist eine echte Teilmenge der natürlichen Sprache. Sie besitzt eine in Bezug auf eine spezifische Domäne eingeschränkte Grammatik (Syntax) und definiert eine Menge von Begriffen (Lexik), die zur Konstruktion von Aussagen über die Domäne verwendet werden können. Ihr Ziel ist die Dokumentation komplexer thematischer Zusammenhänge.

Kontrollierte Sprache eignet sich als Fachsprache zur vereinfachten Formulierung von Aussagen über eine Domäne. Die begrenzte Ausdrucksmöglichkeit ist eine wesentliche Eigenschaft der kontrollierten Sprache und führt aufgrund der „Quasi-Standardisierung“ und einem gemeinsamen Vokabular zu einer besseren Lesbarkeit und einem höheren Verständnis bei den *Stakeholdern*. Damit eignen sich kontrollierte Sprachen insbesondere für die Spezifikation von Anforderungen, da einerseits der Interpretationsspielraum verkleinert wird und andererseits die Möglichkeit besteht, Widersprüche schneller aufzuspüren (Pohl, 2007, S. 247).

Bei der Erhebung von Anforderungen sind kontrollierte Sprachen weniger geeignet, da sie zum einen die Ausdrucksfähigkeit der *Stakeholder* einschränken und zum anderen umfangreiche Leitfäden voraussetzen, in denen der Umgang mit der kontrollierten Sprache erläutert wird. Indem nicht nur die Menge an zulässigen Wörtern und deren Bedeutung definiert wird, sondern auch eine Einschränkung in der Grammatik der jeweiligen Sprache erfolgt, ist sie wesentlich restriktiver als beispielsweise ein Glossar.

1.3.1.1 Glossare

Ein Glossar vermeidet durch die Identifikation sowie Definition wesentlicher fachlicher und technischer Begriffe nicht nur Ambiguitäten, sondern bildet eine gemeinsame Sprachgrundlage unter den *Stakeholdern* (Grechenig, 2010, S. 195). Möglich wird das durch die strukturierte Auflistung aller Termini eines Fachgebiets (Terminologie) und deren Definitionen (Balzert, 2009, S. 482). In dieser Arbeit wird ein Glossar nach Pohl (2008, S. 707) wie folgt definiert:

Definition 1.3.6 (Glossar)
Ein Glossar legt die spezifische Bedeutung einer Menge von Fachbegriffen einer Domäne (d. h. eine Fachterminologie) fest. Neben den Begriffsdefinitionen kann ein Glossar Verweise zwischen verwandten Begriffen sowie Beispiele zur Erläuterung der Begriffe beinhalten.

Ein Glossar verhindert somit, dass beispielsweise synonyme Begriffe in der Dokumentation verwendet werden können. Auch, weil alle *Stakeholder* dazu aufgerufen sind, sich bei ihrer Anforderungsformulierung auf die im Glossar befindlichen Termini zu beschränken. Um ein gemeinsames Begriffsverständnis zu fördern, können unter anderem Beispiele und Gegenbeispiele darin aufgeführt werden (Pohl, 2007, S. 244).

Syntaktische Strukturen für die Dokumentation von natürlichsprachlichen Anforderungen werden von einem Glossar nicht vorgegeben, können aber durch ergänzende

Vorgehensweisen, wie zum Beispiel syntaktischen Anforderungsmustern, vorgegeben werden. Die hierbei eingeführten Limitationen syntaktischer Strukturen werden im Folgenden diskutiert.

1.3.1.2 Syntaktisches Anforderungsmuster

Ein vorgegebener Lückentext in Form von syntaktischen Anforderungsmustern dient der Vermeidung von häufig auftretenden Fehlern, wobei die Semantik dabei bewusst nicht eingeschränkt wird. Rupp (2014, S. 217 f.) bezeichnet diese Muster als Baupläne einzelner qualitativ hochwertiger Anforderungen, in denen aber durchaus Änderungen und Variationen gestattet sind. Dabei existieren verschiedene Schablonen, die sowohl für FA als auch für NFA eingesetzt werden. In Anlehnung an Rupp (2014, S. 218) wird in dieser Arbeit das syntaktische Anforderungsmuster wie folgt definiert.

Definition 1.3.7 (Syntaktische Anforderungsmuster)
Syntaktische Anforderungsmuster (Anforderungsschablone) sind Sprachfragmente, die sowohl Satzstellung als auch Wortwahl bei einzelnen Anforderungsformulierungen festlegen.

Eine mögliche Schablone zur natürlichsprachlichen Dokumentation funktionaler Anforderungen ist in Abbildung 1.6 dargestellt. Das Prozesswort beschreibt eine zu erbringende Systemfunktion durch ein Verb (z. B. „*anzeigen*“, „*exportieren*“). Darüber hinaus ist die Angabe von Bedingungen und Qualitätsanforderungen möglich, die an das entsprechende Prozesswort gebunden sind. Als Prozesswort dient im Beispiel 1.3.1 das Verb „*weiterleiten*“, an das die Qualitätsanforderung „*maximal zwei Versuche*“ und die Bedingung „*entgegengenommener Druckauftrag*“ gekoppelt sind.

Abbildung 1.6: Syntaktisches Anforderungsmuster.
In Anlehnung an Pohl (2007, S. 220) bzw. Rupp (2014, S. 218 ff.)

Beispiel 1.3.1 (Ausgefülltes Anforderungsmuster)
„Wird ein Druckauftrag entgegengenommen, soll das System fähig sein, den Auftrag an den Drucker in maximal zwei Versuchen weiterleiten.“

Die Vorgabe konkreter, ausgewählter Grammatikregeln rückt die natürliche Sprache mit ihrem ursprünglichen informalen Charakter als Modellierungssprache näher an künstlich definierte, (semi-)formale Modellierungssprachen. Im Folgenden werden deshalb sowohl textuelle als auch visuelle Modellierungssprachen im Sinne einer Abgrenzung betrachtet.

1.3.2 Semi-formale Anforderungsdokumentation

Semi-formale Methoden stellen nach Brugger (2009, S. 234) einen „Kompromiss zwischen Formalität und Verständlichkeit dar“. Grundsätzlich basieren sie auf einer vorgegebenen Struktur bzw. einer präzise definierten Syntax und auf graphischer Darstellung (Kurth, 1991, S. 47). Es ist möglich, Ergänzungen in natürlicher Sprache vorzunehmen, die unter anderem als Träger der Semantik fungieren (Brugger, 2009, S. 234). Brugger (2009, S. 234 f.) weist darüber hinaus darauf hin, dass nicht alle semi-formalen Techniken den gleichen Formalisierungsgrad aufweisen. Der semi-formale Charakter sorgt aber letztendlich dafür, dass nicht alle *Stakeholder* in der Lage sind, ihre Softwareanforderungen modellbasiert zu dokumentieren. Dabei haben semi-formale Methoden entschiedene Vorteile für *Stakeholder*, wie beispielsweise die Strukturiertheit und Übersichtlichkeit durch verschiedene Sichten auf das zu spezifizierende Problem.

Für jede Perspektive existiert eine geeignete Modellierungssprache, mit der die betrachteten Informationen zweckmäßig dokumentiert werden können (Rupp, 2012, S. 16 ff.). Traditionell stellen „Struktur“, „Funktion“ und „Verhalten“ nach Pohl und Rupp (2015, S. 37, 75) die „komplementären Perspektiven zur Beschreibung funktionaler Anforderungen“ dar (Pohl, 2007, S. 184 ff.):

- **Strukturperspektive**
  Betrachtung statischer System- und Datenstrukturen unter Ausblendung dynamischer Aspekte wie Zustandsänderungen (Rupp, 2012, S. 17).

- **Funktionsperspektive**
  Der Fokus liegt auf bereitzustellenden Systemfunktionen, wobei auch die Ein- und Ausgaben (Daten / Informationen) und deren Manipulation durch Systemfunktionen betrachtet werden (Rupp, 2012, S. 17).

- **Verhaltensperspektive**
  Betrachtet werden Zustände, Zustandswechsel und erzeugte Ausgaben, die ein System bzw. einzelne Komponenten und Objekte einnehmen können. Es besteht eine enge Verknüpfung mit der Funktionsperspektive (Rupp, 2012, S. 17).

Pohl (2007, S. 186) weist auf zwei wichtige Eigenschaften dieser Techniken hin: Zum einen sind die dargestellten Perspektiven nicht disjunkt. Dies führt dazu, dass eine Integration der verschiedenen Perspektiven notwendig ist, was wiederum die wechselseitige Prüfung einzelner Modelle auf Konsistenz und Vollständigkeit ermöglicht (Pohl und Rupp, 2015, S. 76). Zum anderen sind bislang keine Qualitätsanforderungen berücksichtigt. Diese können durch textuelle Annotationen in den Modellen hinzugefügt werden. Eine Integration der verschiedenen Perspektiven ist teilweise mittels objektorientierten Modellierungsprachen möglich (Pohl, 2007, S. 186).

Objektorientierte Modellierungssprachen stellen nach Fettke (2012) „eine Alternative zur strukturierten Systemanalyse und zum strukturierten Systementwurf“ dar und ermöglichen eine weitreichende Integration der genannten Perspektiven (Pohl, 2007, S. 200). Im Zentrum dieser Modellierungssprachen stehen Objekte bzw. Klassen, die „durch eine Datenstruktur, Funktionen zur Manipulation der Daten sowie

durch ein spezifisches Verhalten definiert werden“ (Pohl, 2007, S. 200). Mit diesen Eigenschaften ermöglichen objektorientierte Modellierungssprachen nach Schwinn (2011, S. 51) eine „durchgängigen Sichtweise in der Software-Entwicklung – von den Systemanforderungen, dem Bauplan, bis zum Code“ (Schwinn, 2011, S. 51). Bereits während der Erhebung und Dokumentation von Anforderungen können somit relevante Systemeigenschaften bestimmt und hinsichtlich der oben genannten Perspektiven präzise spezifiziert werden (Schwinn, 2011, S. 51).

Im Unterschied zu „traditionellen Ansätzen“ (Pohl, 2007, S. 200) ist durch die Integration dieser Perspektiven die Notwendigkeit und die Gefahr von Methodenbrüchen wesentlich niedriger (Schwinn, 2011, S. 51). Zur objektorientierten Modellierung eignet sich beispielsweise die UML, die mittlerweile als Standard der Softwaresystem-Modellierung im *Software Engineering* gilt (Schwinn, 2011, S. 47) und sich in der objektorientierten Architekturentwicklung etabliert hat (Tiemeyer, 2013, S. 327).

Diagrammtypen

Bei der UML handelt es sich um eine graphische Modellierungssprache, die nach Rupp und Queins (2012, S. 4 f.) zur Modellierung, Visualisierung sowie Spezifikation und Dokumentation komplexer Systeme eingesetzt wird (Schwinn, 2011, S. 48, 53). Sie wird gemeinhin als eine semi-formale Methode verstanden, auch wenn sie sowohl natürlichsprachliche als auch formale Elemente[10] aufweist (Laplante, 2007, S. 58, 103; Laplante, 2013, S. 83 f.).

Zu den Vorteilen der UML zählen ihre Standardisierung und Verbreitung: So führt die Standardisierung zur eindeutigen Definition einzelner Diagramme (vgl. Tabelle 1.1), die aufgrund der hohen Praxisrelevanz in vielen Unternehmen weit verbreitet sind. Ihr Gebrauch führt erwartungsgemäß im Vergleich zu informalen Dokumentationstechniken zu weniger Missverständnissen und Fehlinterpretationen.

UML-Diagramme		
Struktur	**Verhalten**	**Interaktion**
Klassen-, Objekt-, Kompositionsstruktur-, Komponenten-, Verteilungs-, Paket-, Profil-Diagramm	*Use-Case*-, Aktivitäts-, Zustandsdiagramm	Sequenz-, Kommunikations-, Interaktionsübersichts-, *Timing*-Diagramm

Tabelle 1.1: Diagrammtypen der UML 2.
In Anlehnung an Rupp und Queins (2012, S. 7)

[10] Laplante (2007, S. 58) nennt explizit die *Object Constraint Language*, mit der Randbedingungen in der Softwareentwicklung formal beschrieben werden (z. B. *Preconditions / Postconditions*).

Perspektiven

Die UML besitzt in der zweiten Version insgesamt 14 Diagrammtypen (vgl. Tabelle 1.1), von denen zwar nicht alle für Neuentwicklung eines Softwaresystems hilfreich sind, die verschiedenen Perspektiven (Struktur-, Verhalten- und Funktionsperspektive) aber gut abdecken (Schwinn, 2011, S. 51). Nach Pohl und Rupp (2015) eignen sich zur Modellierung der **Strukturperspektive** beispielsweise UML-Klassendiagramme, die als „statische Basis der Anwendungssysteme“ gelten (Schwinn, 2011, S. 55). Sie beschreiben die statische Struktur eines Systems und beantworten damit folgende Frage: „Aus welchen Klassen besteht [das] System und wie stehen diese untereinander in Beziehung?“ (Rupp und Queins, 2012, S. 11).

Abbidung 1.7 zeigt ein Modellierungsbeispiel, welches zwei Klassen (Person, E-Mail-Konto) umfasst. Neben Namen, Merkmalen (Attributen) und Operationen, die in den Klassen selbst untergebracht sind, ist auch eine Kante zur Darstellung der Beziehung (Assoziation) abgebildet (vgl. Abbildung 1.7). Als optional gelten im Klassendiagramm die Rollen (z. B. „Besitzer“) und die Multiplizitäten (z. B. „0..*“). Letztere geben an, „wie viele Instanzen einer Klasse in Bezug auf die betrachteten Assoziationen mit wie vielen Instanzen der assoziierten Klassen in Beziehung stehen können“ (Pohl und Rupp, 2015, S. 80).

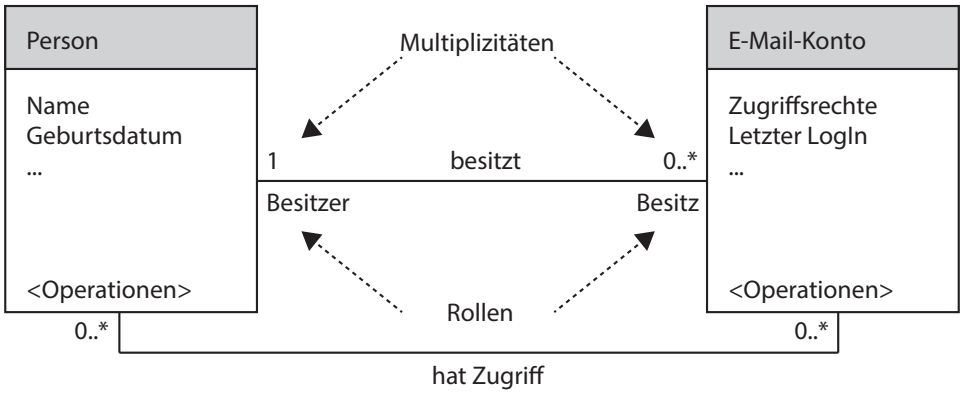


Abbildung 1.7: Semi-formale Dokumentation mittels Klassendiagramm. In Anlehnung an Pohl und Rupp (2015, S. 80)

Rupp (2007, S. 194 f.) weist im Kontext der Anforderungsdokumentation auf die Möglichkeit hin, Klassendiagramme auch als Begriffsmodelle zu nutzen und damit – ergänzend zu einem Glossar – Begriffe und deren Beziehungen zu beschreiben.

Anders als die statische Strukturperspektive ist die **Verhaltensperspektive** dynamisch (Pohl und Rupp, 2015, S. 89) und umfasst die Darstellung von Objekten, die „ihren Zustand als Reaktion auf Ereignisse und Zeitablauf ändern“ (Schmuller, 2003, S. 120). Um dieses „reaktive Verhalten eines Systems“ (Pohl und Rupp, 2015, S. 91) abzubilden, werden UML-Zustandsdiagramme genutzt. Sie ermöglichen es, Objekte, Zustände und Übergänge zwischen Zuständen sowie die Start- und Endpunkte einer Reihe von Zustandsänderungen darzustellen (Schmuller, 2003, S. 120).

Die Funktionalität eines Systems und die damit einhergehende Transformation von Eingaben in definierte Ausgaben kann durch die **Funktionsperspektive** dargestellt werden (Pohl und Rupp, 2015, S. 82). UML-Aktivitätsdiagramme werden einge-

setzt, da sie sich „wie kaum eine andere Dokumentationstechnik [dazu eignen,][...] Abläufe jeglicher Art und deren Regeln darzustellen“ (Rupp, 2007, S. 205 f.). Ein Aktivitätsdiagramm stellt beispielsweise die Konstellation von Aktionen (kleinste ausführbare Einheit innerhalb einer Aktivität) und deren Verbindungen (gerichtete Kanten) mit Kontroll- und Datenflüssen dar, wobei das Kontrollflussmodell die Reihenfolge von Aktionen spezifiziert und das Datenmodell die Daten angibt, die zwischen den Aktionen ausgetauscht werden (Balzert, 2009, S. 236 ff.). Die zentrale Frage, die mit diesem Diagramm beantwortet werden kann, ist, wie bestimmte flussorientierte Prozesse oder Algorithmen ablaufen (Rupp und Queins, 2012, S. 12).

Diese Sichtweise ermöglicht es, durch Systematisierung und Strukturierung der Anforderungen, fehlende Aktionen oder Denkfehler zu erkennen, die in natürlicher Sprache gegebenenfalls übersehen worden wären. Diese Überprüfbarkeit und das mathematische Konkretisieren von Anforderungen (als Kalkül) wird in formalen Methoden weitergeführt, indem die formale Syntax um eine semantische Interpretation erweitert wird (Kurth, 1991, S. 48 f.).

1.3.3 Formale Anforderungsdokumentation

Anders als informale und semi-formale Dokumentationstechniken, haben formale Sprachen eine eindeutig definierte Syntax und Semantik (Schneider, 1998, S. 809). Für die Spezifikation von Softwaresystemen bedeutet das, dass „die ungenaue menschliche Sprache durch die präzisen Mittel der Mathematik [...] ersetzt“ (VSEK Konsortium, 2007b) wird und fehlerhafte Spezifikationen zu einem möglichst frühen Zeitpunkt vermieden werden. „Qualität wird [somit] in formal spezifizierte Systeme hineinkonstruiert“ (VSEK Konsortium, 2007b).

Der formale Charakter ermöglicht einerseits die exakte Dokumentation sowie (Teil-)Verifikation (Tiemeyer, 2013, S. 238), führt aber andererseits zum Ausschluss vieler *Stakeholder* (Wiegers, 2005, S. 153) und geringerer Akzeptanz (Hood und Wiebel, 2005, S. 38), da das zugrundeliegende logische Modell von mathematischer Natur ist und damit Fachwissen zur Interpretation voraussetzt (Kurth, 1991, S. 48).

Nach Brugger (2009, S. 233) sind formale Sprachen damit „ungeeignet für die Kommunikation auf breiter Basis“, was aber nicht bedeutet, dass formale Methoden nicht zu erlernen sind oder dass das notwendige mathematische Grundverständnis ihren Einsatz verhindert (Hall, 1990, S. 16 ff.).

Formale Methoden, wie beispielsweise die Prädikatenlogik (Stang, 2002, S. 120), werden in vielen Softwarebereichen zur Spezifikation genutzt (Hall, 1990, S. 16). Sie sind, unter anderem bei der Entwicklung sicherheitskritischer Systeme (z. B. Hall und Chapman, 2002), insbesondere im Hinblick auf vermeidbare Widersprüche, Inkonsistenzen und Fehler, oftmals unverzichtbar und der Dokumentation mittels natürlicher Sprache, insbesondere wegen der Verifikationsmöglichkeiten, überlegen. Beispielsweise befasst sich die Prädikatenlogik mit Aussagen, die, im Gegensatz zu Fragen, Ausrufen usw., in der modelltheoretischen Semantik Wahrheitswerte annehmen – sie können demnach wahr oder falsch hinsichtlich einer bestimmten Variablenbelegung werden (Stang, 2002, S. 120).

Beispiel 1.3.2 (Prädikatenlogische Aussagen)
(a) $\exists x, y(person(x) \wedge emailkonto(y) \wedge besitzen(x, y))$
(b) $\forall x(person(x) \rightarrow \exists y(emailkonto(y) \wedge besitzen(x, y)))$
(c) $\forall y(emailkonto(y) \rightarrow \exists x(person(x) \wedge besitzen(x, y)))$

Beispiel 1.3.2 enthält drei prädikatenlogische Aussagen: (a) gibt an, dass mindestens eine Person mindestens ein E-Mail-Konto besitzt. Nun ist in dieser Arbeit die modelltheoretische Überprüfbarkeit der Aussagen von besonderem Interesse, welche sich bei der Interpretation einer formalen Grammatik auf einem vorgegeben Modell nachweisen lässt (Pohl, 2007, S. 247). So sagt zum Beispiel (b) aus, dass jede Person ein E-Mail-Konto besitzt. Diese Aussage lässt sich falsifizieren, sobald mindestens eine Person genannt wird, die kein E-Mail-Konto besitzt. Anders verhält es sich bei (c), wo die Aussage getroffen wird, dass jedes E-Mail-Konto mindestens einen Besitzer hat. Diese Aussage ließe sich nur dadurch falsifizieren, wenn mindestens ein herrenloses E-Mail-Konto existieren würde.

Wie Beispiel 1.3.2 zeigt, können Anforderungen an ein Softwaresystem auch im prädikatenlogischen Sinne interpretiert werden, was bedeutet, dass Anforderungsdokumentationen „mit den gängigen Instrumenten von Aussagen- und Prädikatenlogik bearbeitet werden [können]“ (Stang, 2002, S. 120). So können beispielsweise Anforderungen, die nicht gleichzeitig in einem System realisiert werden können, durch die logische Analyse *ad absurdum* geführt werden.

Hood und Wiebel (2005, S. 38) führen darüber hinaus an, dass formale Methoden und Notationen in der Einführung mit kostenintensivem Mehraufwand einhergehen. Allerdings lässt sich auch dahingehend argumentieren, dass eine hohe Qualität, das heißt die frühzeitige Entdeckung von Fehlern, zur Vermeidung von kostenintensiven Garantie- und Gewährleistungsfällen beiträgt (Hall und Chapman, 2002, S. 24).

1.3.4 Gegenüberstellung

Informale Anforderungsbeschreibungen der *Stakeholder* dienen oftmals als Grundlage für einen initialen Softwareentwurf, der in einer späteren Projektphase in eine formalere Spezifikation überführt werden kann. In Abbildung 1.8 ist der reflektierte Charakter dieses Vorgehens zu erkennen, da in diesen Schritten explizit eine Rückkopplung zum vorherigen Spezifikationsschritt vorgesehen ist. Dieser iterative Evaluationsprozess zwischen Anforderungsbeschreibungen und Spezifikation trägt wesentlich zur Qualitätssicherung bei.

Eine Prüfung der Anforderungen findet zum Beispiel bei der formalen Spezifikation statt, wo Entwurfsfehler (z. B. Widersprüche) entdeckt und in den zugrundeliegenden Spezifikationen behoben werden können. Je spezifischer die Spezifikation wird, desto weniger *Stakeholder* werden noch miteinbezogen - außer den Softwareentwicklern, die den gesamten Prozess aktiv mitgestalten müssen. Es ist aber auch herauszustellen, dass formale und informale Vorgehensweisen als komplementär zu betrachten sind: Die angeführten Methoden mit ihren jeweiligen technischen Ausprägungsformen (z. B. kontrollierte Sprachen oder UML) haben unterschiedliche Anwendungsfälle, Nutzer sowie Stärken und Schwächen. Sie können daher in der gemeinsamen Nutzung Synergieeffekte erzielen.

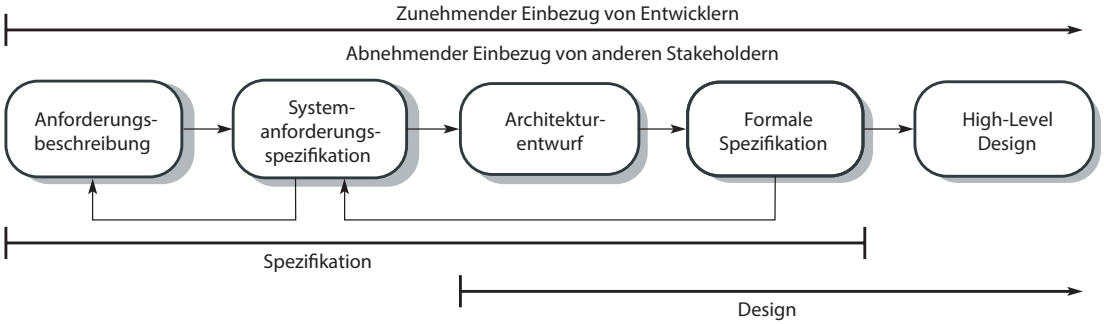


Abbildung 1.8: Formale Spezifikation und Design.
In Anlehnung an Sommerville (2009, S. 4)

Im Fokus der Bewertung dieser Methoden stehen oftmals die Begriffe Präzision und Anwendbarkeit, die in der folgenden Gegenüberstellung unter den Aspekten Formalitätsgrad sowie Benutzerakzeptanz aufgegriffen werden.

Formalitätsgrad

Als Stärke der semi-formalen und formalen Methoden wird oftmals ihre hohe **Präzision** genannt (Rupp 2014, S. 214; Hood und Wiebel 2005, S. 37; Sommerville 2009, S. 3). Diese ist auch bei komplexen Sachverhalten zu erreichen, die unter Umständen in natürlicher Sprache nur schwer konsistent und strukturiert zu fassen sind (Pohl, 2007, S. 298). Hierzu wird unter anderem auf die Darstellung von diskreten Perspektiven, die den Ausdruck eines bestimmten Blickwinkels ermöglichen, zurückgegriffen (Pohl, 2007, S. 298).

Bei der Verwendung natürlicher Sprache können Perspektiven (s. Abschnitt 1.3.2) unbewusst vermischt werden und damit, durch eine geringe Strukturierung sowie unklare Perspektivenzuteilung, die Qualität der Dokumentation mindern (Pohl und Rupp, 2015, S. 38). Sehr genaue Anforderungensdokumentationen sind mit informalen Methoden nur unter erheblichem Mehraufwand zu erreichen (Hsia et al., 1993).

Neben der Präzision ist noch die **Eindeutigkeit** ein wichtiges Kriterium. Denn formale Dokumentationstechniken ermöglichen eindeutige Spezifikationen, da sie die erneute Prüfung bestehender Anforderungen während der Formalisierung erfordern (Sommerville 2009, S. 3 f.; Kurth 1991, S. 48 f.). Wichtig ist hierbei die Erkenntnis, dass formale Methoden zwar die Möglichkeit einer fehlerfreien Spezifikation eröffnen, Entwurfsfehler aber weiterhin möglich sind (Hall, 1990, S. 12). Prinzipiell kann jedoch ein hoher Formalisierungsgrad und die Möglichkeiten der Verifikation und Validierung die Identifikation von Fehlern unterstützen (Sommerville, 2009, S. 3 ff.).

Im Gegensatz dazu ermöglicht die informale Dokumentationstechnik eine ungenaue Verwendung der natürlichen Sprache (Pohl 2007, S. 239 ff.; Balzert 2009, S. 481), unter anderem dadurch, dass sie auf mehreren Ebenen hochgradig ambig ist (IEEE, 1998, S. 4 f.) und schnell unstrukturiert wird, wodurch Redundanzen und Widersprüche schwerer zu erkennen sind (Rupp, 2013, S. 79, 83). Die fehlende Präzision erschwert die Kommunikation zwischen den *Stakeholdern* und ergibt sich insbesondere aus

den Phänomenen der Ungenauigkeit und Unvollständigkeit (Pohl 2007, 239 ff.; Rupp 2014, S. 214; Grechenig 2010, S. 153).

Hinsichtlich der universellen **Anwendbarkeit** sind formale Methoden oftmals gegenüber der natürlichen Sprache unterlegen (Hood und Wiebel 2005, S. 37; Sommerville 2009, S. 3; Pohl und Rupp 2015, S. 38). Es besteht eine Einschränkung bei den abzubildenden Problembereichen (z. B. geringe Skalierbarkeit, daher Konzentration auf kritische Systembestandteile) oder Systembestandteilen (z. B. schlechte Anwendbarkeit auf Gestaltung von GUI), was ein deutlicher Nachteil ist. Die natürliche Sprache hingegen ist universell anwendbar und flexibel im Grad der Detaillierung sowie Abstrahierung (Pohl 2007, S. 239; Balzert 2009, S. 481; Kalenborn 2014, S. 73).

Benutzerakzeptanz

Die Benutzerakzeptanz einzelner Dokumentationstechniken wird oftmals hinsichtlich ihrer **Benutzerfreundlichkeit** und möglicher **Einstiegsbarrieren** bewertet (vgl. Tabelle 1.3), wobei dies im Falle der natürlichen Sprache vor allem den geringen Schulungsaufwand und die damit vergleichsweise kurze Einarbeitungszeit umfasst (Pohl 2007, S. 239; Rupp 2012, S. 16; Rupp 2013, S. 79 ff.; Kurth 1991, S. 47; Ka-

	Vorteile	Nachteile
Informal	• Mehrere Abstraktionsebenen [b] • Flexibler Detailgrad [b, c, a] • Universelle Anwendbarkeit [b, e, a]	• Ungenauigkeit [a, b] • Vermischung von Perspektiven [d]
Semi-formal	• Hohe Präzision [e] • Komplexe Sachverhalte [a] • Strukturierte Darstellung [f] • Kompakte Darstellung [f]	• Nicht universell anwendbar [d] • Ggf. Softwaretools notwendig [e]
Formal	• Vermeidung von Entwurfsfehlern [g] • Sehr hohe Präzision [h, i] • Verifikation / Validierung [h]	• Limitierte Skalierbarkeit [g] • Nicht universell anwendbar [h, g] • Ggf. Softwaretools notwendig [j]

Tabelle 1.2: Vor- und Nachteile von Dokumentationstechniken

[a] vgl. Pohl (2007)
[b] vgl. Balzert (2009)
[c] vgl. Kalenborn (2014)
[d] vgl. Pohl und Rupp (2015)
[e] vgl. Rupp (2014)
[f] vgl. Rupp (2013)
[g] vgl. Sommerville (2009)
[h] vgl. Hood und Wiebel (2005)
[i] vgl. Sommerville (2011)
[j] vgl. Kurth (1991)

lenborn 2014, S. 73). Im Vergleich zu semi-formalen und formalen Methoden ist dies ein starkes Argument, da diese sich nicht nur dem ungerechtfertigten Vorwurf unnötiger Schulungen zur Methodenkompetenz ausgesetzt sehen (Kurth 1991, S. 48 f.; Sommerville 2009, S. 4), sondern auch Mehraufwand für die *Stakeholder* bei der Erstellung der Modelle entsteht (Rupp 2014, S. 213; Kurth 1991, S. 48 f.; Hood und Wiebel 2005, S. 37; Sommerville 2011, S. 336).

Dass zwangsläufig **Mehrkosten** bei der Verwendung formaler Methoden entstehen, ist umstritten (z. B. Hall, 1990, S. 17 ff.). So können Schulungskosten beispielsweise als einmalige Investition und nicht als projektspezifischer Mehraufwand gesehen werden. Ferner gibt Hall (1990, S. 16 f.) zu bedenken, dass Unterweisungen auch unabhängig von der gewählten Methode notwendig sind.

Trotzdem bleibt die **limitierte Ausdrucksfähigkeit** als Gegenargument für formale Techniken bestehen. Hierbei besteht das Risiko, dass *Stakeholder* zwar Modellierungskenntnisse aufweisen, beispielsweise aber die eigenen Arbeitsschritte und Softwareanforderungen nicht modellieren können, da sie in ihrer Ausdrucksfähigkeit limitiert sind (Pohl, 2007, S. 298). Allerdings kann diese Restriktion auch als Chance begriffen werden, sofern diese zu einer Verringerung der thematischen Komplexität führt. So kann auch in der natürlichsprachlichen Anforderungsdokumentation die Ausdrucksfähigkeit eingeschränkt werden. Beispielsweise schränken Glossare die zulässige Terminologie ein und syntaktische Anforderungsmuster limitieren die Syntax. Dies geschieht mit dem Ziel, das Risiko von Redundanzen, Widersprüchen und Unübersichtlichkeit zu minimieren (Pohl, 2007, S. 239 ff.).

Zwar ist es weitestgehend möglich, Anforderungen frei von Ungenauigkeiten zu verfassen, es stellt *Stakeholder* aber vor erhebliche Herausforderungen (Kamsties, 2005). Dies erscheint vor dem Hintergrund fehlender Restiktionen und Vorgaben insbesondere dann unrealistisch, wenn viele *Stakeholder* an der Anforderungsdokumentation beteiligt sind, was mit zunehmender Projektgröße der Regelfall ist (Grechenig, 2010, S. 143). Der resultierende **Interpretationsspielraum** stellt ein erhebliches Projektrisiko dar (Pohl, 2007, S. 239 ff.).

Schlussendlich weist Rupp (2013, S. 83) am Beispiel von Klassendiagrammen darauf hin, dass selbst *Stakeholder*, die über die notwendigen **Modellierungskenntnisse** verfügen und eine Anforderungsdokumentation prinzipiell gutheißen, die modellbasierte Dokumentation ablehnen können, wenn der notwendige Modellierungswille fehlt, bzw. eine Methode, losgelöst vom jeweiligen Formalisierungsgrad, keine Akzeptanz unter den *Stakeholdern* findet (Hall 1990, 18 f.; Kurth 1991, S. 46; Hood und Wiebel 2005, S. 38). Endanwender, die nicht über ausreichende Methodenkompetenz verfügen, sind daran interessiert, ohne Einschränkungen in ihrer Ausdrucksfähigkeit die individuellen Anforderungen an ein Softwareprojekt zu beschreiben. Gegenstand dieser Arbeit sind daher informale Anforderungsbeschreibungen, mit allen Vor- und Nachteilen der natürlichsprachlichen Anforderungsdokumentation.

1.4 Anforderungsbeschreibungen

Der Begriff der „Anforderungsbeschreibung“ steht in dieser Arbeit für eine Menge informal formulierter Anforderungen an ein Softwareprodukt (Leistungsumfang). Die individuellen Anforderungen werden von Endanwendern mit der Intention gestellt,

bei einer zukünftigen Softwareentwicklung berücksichtigt zu werden und damit den gewünschten Funktionsumfang abzudecken. Die Art und Weise der softwareseitigen Umsetzung ist dabei nicht zwangsläufig vorgegeben (vgl. Beispiel 1.4.1).

Zwar existiert keine eindeutige Definition von „Anforderungsbeschreibung“, allerdings wird der Terminus bereits in der Literatur mit Bezug auf die Entwicklung von Softwareprodukten verwendet. So nutzen beispielsweise Schneider und Vecellio (2011, S. 106) den Begriff „Anforderungsbeschreibung“ im Kontext strukturierter Dokumentation von Anforderungen und mit Hinweis auf IEEE 830 Standard zur Spezifikation von Anforderungen (IEEE, 1998). Auch Gumm und Sommer (2012, S. 837) sehen in Anforderungsbeschreibungen ein strukturiertes Dokument, welches das Resultat einer Anforderungsanalyse ist und verweisen auf den IEEE 830 Standard. „Herzstück“ einer solchen Beschreibung ist nach Gumm und Sommer (2012, S. 837) die „Beschreibung der funktionalen und nicht-funktionalen Anforderungen“.

Demgegenüber bezeichnet Schienmann (2002) nur den Teil einer strukturierten Anforderung als „Anforderungsbeschreibung“, der eine Beschreibung der jeweiligen

	Chancen	**Risiken**
Informal	• Hohe Verständlichkeit [a, b] • Volle Ausdrucksfähigkeit [c] • Geringer Schulungsaufwand [d, g] • Kurze Einarbeitungszeit [d, e]	• Redundanzen [a] • Widersprüche [a] • Unübersichtlichkeit [a] • Interpretationsspielraum [f, g, h]
Semi-formal	• Verringerung der Komplexität [i, d] • Schnelle Memorisierung [d]	• Limitierte Ausdrucksfähigkeit [d] • Mehraufwand [g] • Komplizierung [g] • Modellierungskenntnisse [g] • Fehlender Modellierungswillen [a]
Formal	• Eindeutige Spezifizierung [j] • Hochgradig reflektiv [j, b]	• Methodenkenntnisse [b, j, k] • Mehraufwand [b, l] • Gefahr von Mehrkosten [j, k] • Komplizierung [b] • Fehlende Akzeptanz [b, l, k]

Tabelle 1.3: Chancen und Risiken einzelner Dokumentationstechniken (Benutzersicht)

[a] vgl. Rupp (2013)
[b] vgl. Kurth (1991)
[c] vgl. Rupp (2012)
[d] vgl. Pohl (2007)
[e] vgl. Kalenborn (2014)
[f] vgl. IEEE (1998)
[g] vgl. Rupp (2014)
[h] vgl. Grechenig (2010)
[i] vgl. Rupp und Queins (2012)
[j] vgl. Sommerville (2009)
[k] vgl. Hall (1990)
[l] vgl. Hood und Wiebel (2005)

Anforderung enthält, die „möglichst kurz und prägnant in einigen Sätzen formuliert werden [sollte]“ (Schienmann, 2002, S. 152). Die wesentlichen Aspekte „Umfang“ und „Prägnanz“ sind im Folgenden zu berücksichtigen:

Definition 1.4.1 (Anforderungsbeschreibung)
Bei Anforderungsbeschreibungen handelt es sich um informale Fließtexte in natürlicher Sprache, die eine Menge von Anforderungen an ein Softwareprodukt beinhalten. Sie sind in der Ausdrucksfähigkeit unbeschränkt und können sowohl in der Länge als auch im Detailgrad und Abstraktionsniveau stark variieren.

Demnach wird Anforderungsbeschreibungen in dieser Arbeit kein hoher Grad an Strukturierung und Formalisierung sowie Prägnanz zugesprochen. Vielmehr handelt es sich um Freitexte (vgl. Beispiel 1.4.1), die oftmals „[...] schwammige, unvollständige, widersprüchliche Anforderungen auf unterschiedlichem Abstraktionsniveau – von Beispielen bis hin zu globalen Aussagen [beinhalten]“ (Balzert, 2009, S. 507).

Auch der Umfang der Anforderungsbeschreibungen schwankt stark. Dabei kann es sich um einen einzigen, aber prägnanten Satz mit zentralen Softwareanforderungen bis hin zu einem sehr umfangreichen Text mit vielen Details handeln, der aber mit unwichtigen Angaben (*Off-Topic*) versetzt ist (vgl. „*like my movies from my summer holidays in ultra hd*“ in Beispiel 1.4.1).

Beispiel 1.4.1 (Anforderungsbeschreibung)

> I want an application with that i can write, read and mark my e-mails (as read/unread). Also, i$_0$ want to mark important$_1$ e-mails with stars. The application must handle big$_2$ attachements$_3$, like my movies from my summer holidays in ultra hd. Of course, the application must filter undesired emails. It would be great, if e-mails could be marked as read or may also be deleted automatically by defined rules. A nice and intuitive$_4$ user interface to sort e-mails in folders$_5$ would be desirable. Of course, the application must be able to format texts (bold, italic, underline, ...). e-mails that I have not yet completed, should be stored in a separate folder. It would be great, if i could specify a due date for these draft e-mails, and also for incoming e-mails. Also reminders for e-mails, for example location based, would be great thing.

Beispiel 1.4.1 enthält mehrere problematische Textstellen. Die Problematik bezieht sich dabei sowohl auf das Textverständnis, als auch auf die Qualität der Anforderung. So werden Angaben wie „*important*“ (1) und „*big*“ (2) genutzt, die als vage zu bezeichnen sind. Darüber hinaus erschweren Rechtschreibfehler („*attachements*“, (3) bzw. „*i*“ (0)) sowie Ambiguitäten und Schreibvarianten (4) einzelner Wörter („*e-mails*“, „*emails*“) die maschinelle Textverarbeitung. Auch vermeintlich vollständige Anforderungen sind vage („*stored in a separate folder*“ (5) → Was für ein Ordner?). Anforderungsbeschreibungen als vorläufiges Ergebnis der informalen Anforderungsdokumentation (s. Abschnitt 1.3.1) weisen, wie in Beispiel 1.4.1 ersichtlich, eine Vielzahl an Defiziten auf und müssen vor einer (maschinellen) Weiterverarbeitung überarbeitet werden. Phänomene wie Ungenauigkeit und Unvollständigkeit werden daher im folgenden Kapitel 2 als Schwerpunkt dieser Arbeit gesondert diskutiert.

Ungenauigkeit und Unvollständigkeit

2

Anforderungen sind „[...] naturgemäß zunächst vage, verschwommen, mehrdeutig, unzusammenhängend, unvollständig und gelegentlich sogar widersprüchlich“ (Partsch, 2010, S. 18). Unvollständigkeit und Ungenauigkeit sind dabei Phänomene, die insbesondere in der informalen Anforderungsbeschreibung auftreten und die Anforderungsqualität erheblich mindern können (s. Abschnitt 1.3.4). Dabei kann die Entstehungsursache variieren, wie Berry (2000) anhand eines „Anforderungseisbergs“ (engl. *requirements iceberg*) darstellt (vgl. Abbildung 2.1).

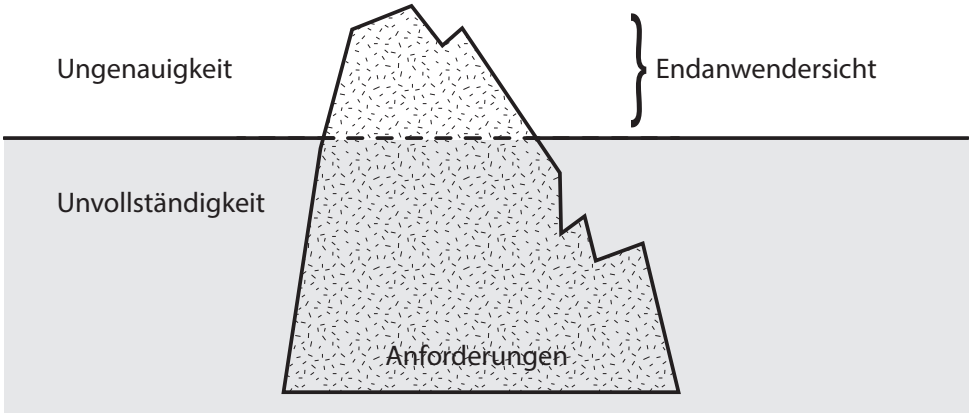


Abbildung 2.1: *Requirements Iceberg.* In Anlehnung an Berry (2000)

Ein *Stakeholder*, in diesem Fall ein Endanwender, kennt nur einen Teil der Anforderungen, die an eine Softwareapplikation gestellt werden (Spitze des Eisbergs) und kann demnach auch nur diese beschreiben. In diesen Anforderungsbeschreibungen können Ambiguität und Vagheit als Formen von Ungenauigkeit auftreten. Darüber hinaus gibt es Anforderungen, die dem *Stakeholder* (z. B. Endanwender, Entwickler) nicht bekannt sind (Rumpf des Eisbergs) und daher auch nicht beschrieben werden können (Unvollständigkeit). Sie können aber wesentlichen Charakter für die Gesamtfunktionalität der Applikation haben (Pohl, 2008, S. 9).

Unvollständigkeit und Ungenauigkeit finden sich auch in der Fehlertypologie von Avci (2008, S. 93) wieder. Als Anforderungsfehler bezeichnet Avci (2008, S. 93) „[...] ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Anforderungsanalyse, sofern es eine Änderung in diesem Ergebnis notwendig macht“. Dabei werden zwei Arten von Fehlern unterschieden: Diejenigen, die in einzelnen Anforderungen auftreten und jene, die in oder durch eine Beziehung zwischen zwei oder mehreren Anforderungen entstehen (vgl. Abbildung 2.2).

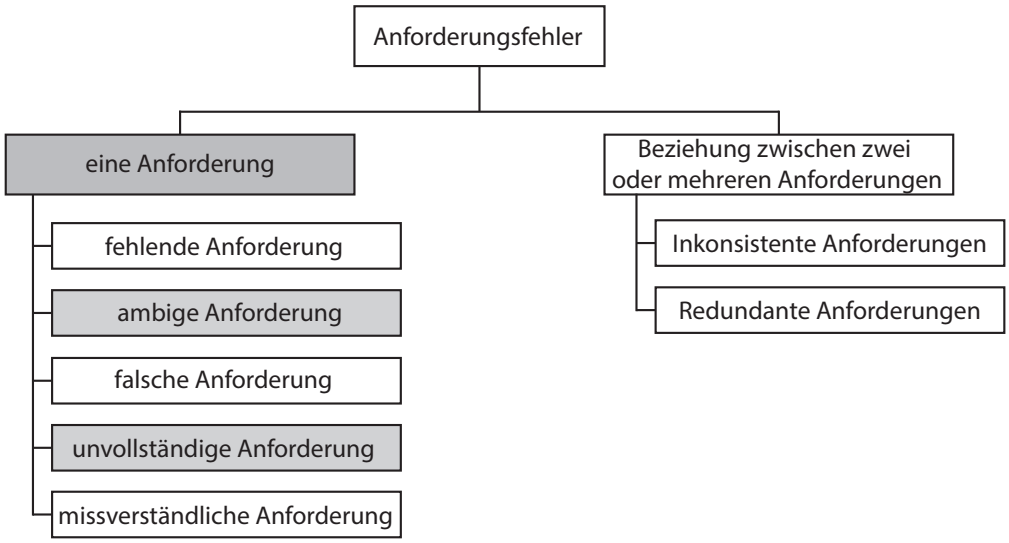


Abbildung 2.2: Typologie von Anforderungsfehlern.
In Anlehnung an Avci (2008, S. 93)

In dieser Arbeit liegt der Fokus auf Unvollständigkeit und Ambiguität in der Beschreibung einzelner Anforderungen. Unvollständigkeit bezeichnet dabei das Fehlen einer Ausprägung einer vorhandenen Anforderung. Es wird demnach von einer unvollständigen Anforderung oder auch *incomplete individual requirement* nach Firesmith (2005, S. 35 ff.) gesprochen. Demnach ist nicht das gänzliche Fehlen einer obligatorischen Anforderung gemeint (fehlende Anforderung). Dies ist aufgrund der Fokussierung auf natürlichsprachliche Anforderungen auch nicht umsetzbar, da – im Gegensatz zu formalen Anforderungsspezifikationen – keine Konsistenzprüfung und Validierung ermöglicht wird.

Die von Avci (2008, S. 93) als problematisch aufgeführte ambige Anforderung wird im Folgenden besonders behandelt. Zum einen, weil bereits der Ambiguitätsbegriff, der von Avci (2008) nicht definiert wird, umstritten ist und einer Erläuterung bedarf. Zum anderen, weil für eine differenziertere Betrachtung zwischen mehreren Formen der Ambiguität in natürlicher Sprache zu unterscheiden ist. Um dies zu strukturieren, wird die Bezeichnung der „Ungenauigkeit“[11] verwendet, die als Oberbegriff die verwandten Phänomene Ambiguität und Vagheit[12] zusammenführt (vgl. Abbildung 2.3). Dies geschieht wohl wissend, dass es keine allgemeingültige Definition und Aufteilung des Mehrdeutigkeitsphänomens gibt.

Anders als Abbildung 2.3 suggeriert, handelt es sich bei Ambiguität und Vagheit nicht um isolierte Phänomene, sondern um Ausprägungen von Ungenauigkeit, die auch gleichzeitig auftreten können. Beide eint, dass sie unvollständiges Wissen darstellen und damit eine gewisse Unsicherheit bei der Interpretation erzeugen. Nichtsdestotrotz gelten sie insofern als eigenständige Phänomene, als dass sie unterschiedlich zu kompensieren sind.

[11] Siehe auch Dudenredaktion (2016, S. 1847).
[12] Das Phänomen der Vagheit ist nur begrenzt Gegenstand dieser Arbeit. Mehr Informationen zur Vagheit im Anforderungskontext geben Geierhos und Bäumer (2017).

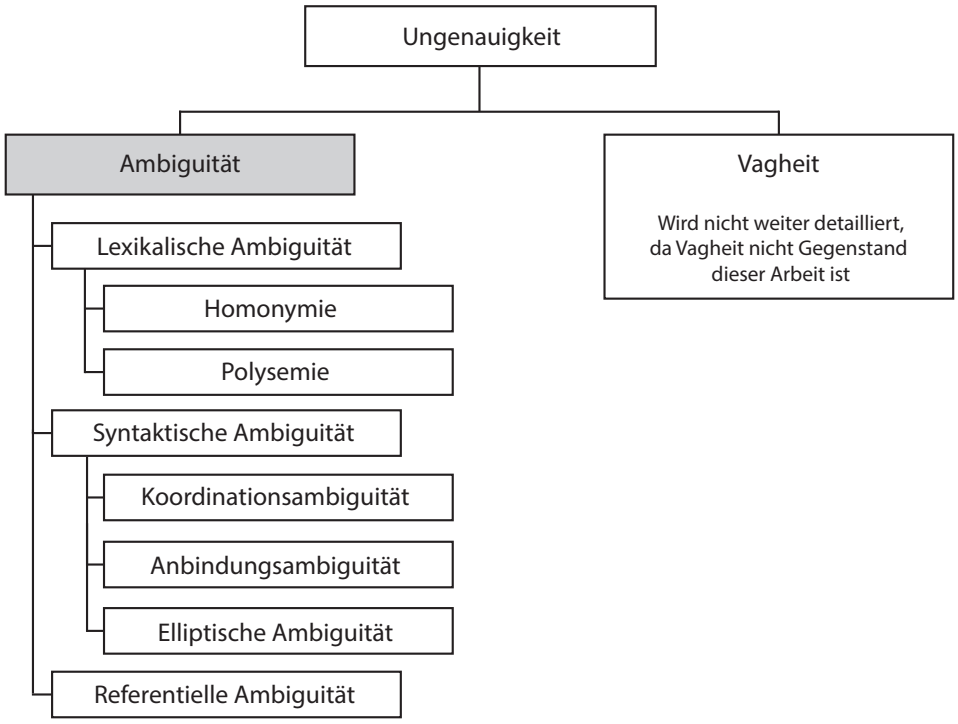


Abbildung 2.3: Der Begriff der Ungenauigkeit

Im Folgenden werden sowohl die Ausprägungen von Ambiguität als auch Unvollständigkeit im Kontext dieser Arbeit und im Hinblick auf Kompensationsmöglichkeiten beschrieben.

2.1 Ambiguität

In der modernen Sprachwissenschaft steht der Begriff der Ambiguität für die Mehrdeutigkeit sprachlicher Äußerungen (Pfeifer, o. D.), die bewusst oder unbewusst verwendet werden kann (Berghuber, 2008). Nach Löbner (2003, S. 53 ff.) können Wörter zum Beispiel mehrere Bedeutungen innehaben (lexikalische Ambiguität) und auch Sätze mehrere Lesarten erlauben (syntaktische Ambiguität).

Definition 2.1.1 (Ambiguität)
Eigenschaft von Ausdrücken natürlicher Sprachen, denen mehrere Interpretationen [(auch: Lesarten)] zugeordnet werden können, bzw. die unter lexikalischem, semantischem, syntaktischem u. a. Aspekt in der linguistischen Beschreibung mehrfach zu spezifizieren sind. (Bußmann, 1983, S. 26)

Beispiel 2.1.1 (Ambiguität)
„Ein Rechtsklick mit der Maus$_1$ schließt das Fenster."
„Die Maus$_2$ knabbert an der Weihnachtsschokolade."

Beispiel 2.1.1 zeigt exemplarisch einen Fall von lexikalischer Ambiguität. Das Wort „Maus" kann je nach Lesart ein Peripheriegerät (Maus$_1$) oder ein Tier (Maus$_2$)

beschreiben. Aus dem Kontext „Rechtsklick“ wird für den menschlichen Leser die korrekte Bedeutung von „Maus“ schnell ersichtlich und auch die meisten automatischen Verfahren benötigen diese Kontext-*Trigger* zur Kompensation (Carstensen et al., 2010, S. 383). Diese Auflösung von Ambiguitäten wird als Disambiguierung bezeichnet und wird zumeist über den sprachlichen oder außersprachlichen Kontext realisiert (Bußmann, 1983, S. 26):

Definition 2.1.2 (Disambiguierung)
Disambiguierung ist der Vorgang und das Ergebnis der Auflösung lexikalischer oder struktureller Mehrdeutigkeit sprachlicher Ausdrücke durch den sprachlichen oder außersprachlichen Kontext.

Die Disambiguierung ist dabei abhängig von der Form der Ambiguität. Daher wird im Folgenden auf die Besonderheiten der lexikalischen, syntaktischen sowie der referentiellen Ambiguität eingegangen, auf die sich in dieser Arbeit konzentriert wird.

2.1.1 Lexikalische Ambiguität

Lexikalische Ambiguität (engl. *lexical ambiguity*) bezieht sich auf die Mehrdeutigkeit eines einzelnen Wortes (Lexem), also auf „Ausdrücke mit derselben Laut- und/oder Schriftform und mehr als einer lexikalischen Bedeutung“ (Löbner, 2003, S. 58). Dabei wird zwischen Homonymie (Gleichnamigkeit) und Polysemie (Vieldeutigkeit) unterschieden (z. B. Berry et al., 2003, S. 10), wobei diese Unterscheidung umstritten ist (Löbner 2003, S. 61; Lehmann 2013).

Von einem Homonym wird gesprochen, wenn ein Lexem den gleichen Wortkörper (Laut- und/oder Schriftform) wie ein weiteres Lexem hat, „aber in der Bedeutung und Herkunft verschieden ist“ (Dudenredaktion, 2016, S. 887). Dies wird in Beispiel 2.1.2 anhand des Wortes „Ton“ illustriert, das in der unterschiedlichen Verwendung des Wortkörpers keine gemeinsame Bedeutungsfacette hat.

Beispiel 2.1.2 (Homonymie)
„Ein Ton_1 erklingt bei falschen Eingaben.“ (Klang)
„Die Kinder formten Töpfe aus Ton_2.“ (Bodenart)

Während „[...] Fälle von Homonymie sehr selten und zufallsbedingt sind“ (Löbner, 2003, S. 60), kommt Polysemie im Alltag oft vor. Dabei handelt es sich um Lexeme, die gewollt „mehrere miteinander verbundene Bedeutungen“ (Löbner, 2003, S. 60) haben. „Polysemie entsteht durch Schaffung abgeleiteter Bedeutungen auf der Basis einer Grundbedeutung“ (Lehmann, 2013), was anhand des Beispiels 2.1.3 durch eine kontextbasierte Disambiguierung des Wortes „Schreibtisch“ aufgezeigt wird.

Beispiel 2.1.3 (Polysemie)
„Vor dem Start wird eine Verknüpfung auf dem Schreibtisch_1 erzeugt.“ (Desktop)
„Auf dem Schreibtisch_2 stapelt sich Papier.“ (Möbelstück)

„Schreibtisch“ kann sowohl für ein Möbelstück, als auch für eine graphische Benutzeroberfläche stehen – in beiden Fällen dient das Wort zur Beschreibung einer

Ablagemöglichkeit. Das Auftreten von Polysemie erklärt Löbner (2003) durch eine natürliche ökonomische Tendenz von Sprache – verfügbare Ausdrücke mit ähnlicher Bedeutung werden für neue Zwecke wiederverwendet. Dies bedeutet, dass jede dieser Bedeutungen auch gelernt werden muss (Löbner, 2003, S. 60).

Demnach ist lexikalische Ambiguität in der Erkennung und Disambiguierung noch insofern dankbar, als das die linguistische Reflexion einzelner Lexeme hinreichend ist, insbesondere dann, wenn die Anzahl möglicher Lesarten hoch ist oder die einzelnen Lesarten stark divergieren (Ceccato et al., 2004; Sennet, 2016). Hierbei wird auf bestehende linguistische Ressourcen zurückgegriffen (z. B. Rojas und Sliesarieva, 2010), um die potentielle Ambiguität zu erkennen (z. B. Kipper-Schuler, 2005). Darüber hinaus existieren Techniken wie das POS-*Tagging*, die Ambiguitäten erkennen können, die aufgrund der Wortart entstehen (z. B. Nomen „*Book*“ und Verb „*to book*“).

Eine Besonderheit bei der lexikalischen Ambiguität ist, dass die Ambiguität das Lexem als Ganzes betrifft. Vielfach wird daher in der Literatur auf Wortlisten (auch: Ambiguitätslisten) zurückgegriffen, die eine Auswahl ambiger Begriffe enthalten (z. B. Lami, 2005; Gleich et al., 2010; Nigam et al., 2012; Génova et al., 2013; Tjong und Berry, 2013), statt Merkmale zu identifizieren. Allerdings können nicht nur Lexeme ambig sein, sondern auch Satzgefüge (syntaktische Ambiguität).

2.1.2 Syntaktische Ambiguität

Syntaktische Ambiguität tritt nach Berry et al. (2003, S. 10 f.) auf, wenn eine Sequenz von Wörtern zu mehr als einer grammatikalischen Struktur führen kann, von der jede eine andere Bedeutung innehat (vgl. auch Ernst 2003, S. 87). Da es sich bei syntaktischer Ambiguität um ein hochkomplexes Problem handelt, ist auch die Effizienz möglicher Disambiguierungslösungen zu berücksichtigen (Carstensen et al., 2010, S. 312), die sich nach der Art der syntaktischen Ambiguität richten. In dieser Arbeit liegt der Fokus auf der Koordinations- und PP-Anbindungsambiguität.

Beispiel 2.1.4 (Koordinationsambiguität)
„[[Die Anwendung] [erstellt [[ausführliche Berichte]$_1$ und Dokus]]].“
„[[Die Anwendung] [erstellt [ausführliche [Berichte und Dokus]]$_2$]].“

Unter „Koordination“ werden in Anlehnung an Bußmann (1983, S. 276) „[...] syntaktische Struktur[en verstanden], die aus zwei oder mehr Konjunkten (= Wörter, Satzglieder oder Sätze)“ bestehen, wobei die Elemente durch „koordinierende Konjunktionen (und, aber, denn) verknüpft sind“.

So ist in Beispiel 2.1.4 unklar, ob sich das Adjektiv „*ausführliche*“ nur auf das erste Konjunkt „*Berichte*“ oder auf beide „*Berichte*“ und „*Dokumentation*“ bezieht. Laut Berry et al. (2003, S. 11) liegen diese Ambiguitäten vor, wenn mehr als eine Konjunktion in einem Satz genutzt oder eine Konjunktion zusammen mit einem Modifikator genutzt wird (vgl. Beispiel 2.1.4).

Ebenfalls relevant ist die Anbindungsambiguität bezogen auf Präpositionalphrasen (engl. *PP-attachment ambiguity*). Mehl et al. (1998) stellen diesbezüglich in einer „Untersuchung von 710 PP-Belegen [fest, dass][...] nicht weniger als 502 (=70,7%) von ihrer syntaktischen Position her nicht eindeutig zuzuordnen [sind]“ (Mehl et al., 1998, S. 1). Langer et al. (1997, S. 1) weisen darauf hin, dass auch fachsprachliche

Texte von Anbindungsambiguität geprägt sind. Dabei ist die „PP-Zuordnung [...] ein typisch computerlinguistisches Problem, weil zu seiner Lösung komplexes semantisches Wissen erforderlich ist, das in keinem sprachverarbeitenden System zur Verfügung steht“ (Mehl et al., 1998, S. 2).

Beispiel 2.1.5 (PP-Anbindungsambiguität)
„[[Die Software][verschickt[das Bild] [mit einem Knopfdruck]$_{PP}$]$_1$]“
„[[Die Software][verschickt[das Bild [mit einem Knopfdruck]$_{PP}$]$_2$]]“

Am Beispiel 2.1.5 zeigt sich, dass die Ambiguität darin besteht, dass die Präpositionalphrase „*mit einem Knopfdruck*“ das Instrument der Handlung (PP_1) innerhalb der Software sein kann (Verbalphrase) oder aber das Bild näher spezifiziert (PP_2), auf dem der Vorgang als solches illustriert wird. Hierbei ist PP_2 die Konstituente einer Nominalphrase. Insgesamt ergeben sich somit zwei Lesarten durch die unterschiedliche syntaktische Anbindung der PP.

Carstensen et al. (2010, S. 302) weisen darauf hin, dass die „Wahrscheinlichkeit einer PP-Anbindung in den allermeisten Fällen nicht nur strukturell determiniert ist, sondern auch stark vom lexikalischen Material abhängt“. Hindle und Rooth (1993) verbessern beispielsweise die syntaktische Disambiguierung durch die „Lexikalisierung einer Grammatik erheblich“ (Carstensen et al., 2010, S. 328). Dies bedeutet, dass die Disambiguierung unter Zuhilfenahme des lexikalischen Kontexts innerhalb eines automatisch geparsten Korpus geschieht (Hindle und Rooth, 1993).

Hingegen entsteht beispielsweise die elliptische Ambiguität nicht primär durch die Struktur, sondern durch Auslassungen (auch: Aussparungen). Auslassungen bezeichnen dabei ein fehlendes, lexikalisch oder syntaktisch notwendiges Element. Ellipsen sollten im *Natural Language Processing* (NLP) aufgelöst werden, da sie der automatischen Verarbeitung Informationen vorenthalten, die für den menschlichen Leser sichtbar sind (McShane und Babkin, 2016, S. 2; McShane und Babkin, 2015).

Normalerweise können diese fehlenden Elemente sehr wohl „aus dem sprachlichen Kontext oder der Redesituation [rekonstruiert werden]“ (Bußmann, 1983, S. 117). Ambig wird ein Satz erst dann, wenn dem Leser nicht eindeutig klar wird, ob ein Element fehlt (Berry et al., 2003, S. 11) oder wie es im Kontext zu kompensieren ist. Beispiel 2.1.6 (a) zeigt das Ergebnis einer Tilgungstransformation[13]. Es ist unklar, ob „kennt“ ausgelassen wurde und es sich somit um eine Ellipse handelt.

Beispiel 2.1.6 (Elliptische Ambiguität)
(a) „Ich kenne einen besseren Programmierer als Jens [kennt]“
(b) „Alle Chatpartner benehmen sich [gut].“

Im Gegensatz zur lexikalischen Ambiguität kann die syntaktische Ambiguität nur begrenzt auf Grundlage einzelner Wörter erkannt werden, da Beziehungen und Abhängigkeiten zwischen Wörtern zu berücksichtigen sind. Gleichwohl nutzen Gleich et al. (2010, S. 222) Signalwörter (z. B. „*only*“, „*also*“, „*even*“) als Merkmal, um Lexeme zu erkennen, die häufig für syntaktische Ambiguität verantwortlich sind sowie die Konjunktionen „*and*“ sowie „*or*“ in Kombination (Gleich et al., 2010, S. 222).

[13] Die eckigen Klammern stellen in diesem Beispiel fehlende Elemente dar.

Auch Chantree et al. (2007) nutzen Signalwörter als Merkmale zur Erkennung möglicher strukturell ambiger Sätze (Koordinationen).

Hingegen basiert die Arbeit von Agarwal und Boggess (1992) auf Strukturinformationen wie POS-*Tags* zur Erkennung von Konjunktionen. Diese können auch im Falle referentieller Ambiguität genutzt werden. Gleich et al. (2010, S. 222) nutzen zur Erkennung potentieller referentieller Ambiguität als Merkmal den Abgleich von Pronomina (z. B. „*she*“). Yang et al. (2010c) ergänzen die reinen POS-*Tags* durch sogenannte „*Construction patterns*“ wie „*adj* n_1 *c* n_2“ als Merkmale für Koordinationsambiguität (Yang et al., 2010c, S. 54).

Auch im Falle von PP-Anbindungsambiguität weist beispielsweise die Struktur V NP PP als Merkmal auf potentielle Ambiguität hin (Agirre et al., 2008, S. 318). Es ist zunächst nicht zu erkennen, ob die PP zur Nominalphrase oder zum Verb zugeordnet werden muss (s. Abschnitt 2.1.2).

Die Erkennung und das Auflösen von Ellipsen gilt als sehr herausfordernd (Bos und Spenader, 2011; Hardt, 1997) und es ist unrealistisch, eine vollständige Auflösung aller Ellipsen auf domänenübergreifenden Freitexten zu erwarten (McShane und Babkin, 2016, S. 2). Laut Pinkal (1985, S. 76 f.) lässt sich elliptische Ambiguität darüber hinaus nicht als isoliertes syntaktisches Problem begreifen. Als Beispiel können mehrstellige Prädikate dienen (Valenz), bei denen einzelne Argumentpositionen unbesetzt sind (vgl. „gut“ als Argument des Prädikats benehmen). Das Fehlen bzw. das Auslassen mindestens eines Arguments kann zu Ambiguität führen (s. Abschnitt 2.3).

2.1.3 Referentielle Ambiguität

Ein Grenzfall hinsichtlich der Charakterisierung von Ambiguität ist die referentielle Ambiguität. Sie kann entweder der syntaktischen oder der semantischen Ambiguität zugeordnet werden, je nachdem ob innerhalb eines Satzes referenziert wird oder über den Satz hinaus.

Als Referenz wird die Beziehung zwischen einem Wort oder einem Satz und einer Entität in der realen Welt, welche das Wort oder der Satz beschreibt, bezeichnet (Berry et al., 2003, S. 12). Bußmann (1983, S. 428) spricht hierbei von der „Bezugnahme auf innersprachlichen und außersprachlichen Kontext durch sprachliche Mittel“. In diesem Zusammenhang werden Anaphern als zurückverweisendes Element (Referenzausdruck) bezeichnet, die als Referenz in Abhängigkeit einer zuvor genannten Referenz eines anderen Elements stehen. Anaphern können daher nicht aufgelöst werden, ohne den Kontext oder einen weiteren Referenzausdruck (engl. *mention*) heranzuziehen (Stede, 2012, S. 41). Das Auflösen von Anaphern wird Anaphernresolution genannt (vgl. Definition 2.1.3).

Definition 2.1.3 (Anaphernresolution)
„The task of finding an antecedent for each anaphora in a text. An anaphor is characterized by the fact that its discourse referent can only be identified when its antecedent is interpreted. ‚Anaphora‘ is an irreflexive, non-symmetrical relation“ (Stede, 2012, S. 41).

Verweisen mindestens zwei Referenzausdrücke innerhalb eines Textes auf eine identische Entität, wird von Koreferenz gesprochen (Mitkov, 2014, S. 5). Eine Sequenz

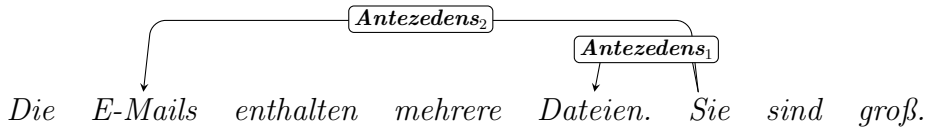
aller Referenzausdrücke, die einer spezifischen Entität zugeordnet werden können, wird Koreferenzkette (engl. *coreferential chain*) genannt und ist Ergebnis der Koreferenzresolution (vgl. Definition 2.1.4).

Definition 2.1.4 (Koreferenzresolution)
„The task of partitioning the set of mentions of discourse referents in a text into classes (or ‚chains') corresponding to those referents. Since referents are identical, ‚coreference' is an equivalence relation (reflexive, symmetrical, transitive)"
(Stede, 2012, S. 41).

Die Wiederaufnahme einer zurückliegenden Textstelle (anaphorische Verbindung) wird in Beispiel 2.1.7 anhand eines Personalpronomens aufgezeigt. Das Personalpronomen („*Sie*") referiert auf das zuvor genannte Antezedens („*Dateien*"). Referentielle Ambiguität entsteht nun, wenn eine Anapher sich auf mehr als ein Antezedens beziehen kann (Berry et al., 2003, S. 12).

Wie Beispiel 2.1.7 zeigt, könnte auch „*E-Mails*" mit „*Sie*" gemeint sein. Neben Personalpronomina können auch andere Pronomina und Proformen als Stellvertreter im Text fungieren.

Beispiel 2.1.7 (Anaphorische Ambiguität)



Hier wird ersichtlich, dass die unterschiedlichen Ausprägungen von Ambiguität nicht isoliert voneinander betrachtet werden können. Genauso verhält es sich mit dem Phänomen der Ungenauigkeit als Ganzes: Vagheit und Ambiguität sind verwandte Phänomene, die in Kombination auftreten können. Vagheit ist dabei weitaus mehr als eine Unterform der Ambiguität (Dönninghaus, 2005, S. 64 f.): Sie wird in vielen Anwendungen und Domänen als ein eigenständiges, komplexes, sprachliches Problem diskutiert (z. B. Petermann 2014; Dönninghaus 2005; Fries 1980).

Als grobes Unterscheidungsmerkmal stellt Pinkal (1991, S. 264) fest: „Vage Ausdrücke haben ein unbestimmtes Denotat; ambige Ausdrücke besitzen mehrere alternative Denotate". Es kann darüber hinaus nach Pinkal (1991) keine unbestimmte Lesart eines ambigen Ausdrucks geben: „Ambige Ausdrücke sind, im Gegensatz zu vagen, desambiguierungs- bzw. präzisierungsbedürftig" (Pinkal, 1991, S. 264). Um eine Unterscheidung der beiden Phänomene zu erleichtern, wird die Vagheit im Folgenden kurz beschrieben.

2.2 Vagheit

Die Komplexität des Vagheitsphänomens zeigt sich schon an den vielen Definitionsversuchen, die zum Teil miteinander unvereinbar sind (Tye 1998; Dönninghaus 2005, S. 159 ff.). Fries (1980, S. 4 ff.) sieht in der Vagheit eine Ausprägung von Mehrdeutigkeit, wobei er Mehrdeutigkeit als einen Oberbegriff für jegliche Möglichkeit der

mehrfachen Interpretation von Morphemen, Lexemen etc. wählt, unter den auch Ambiguität fällt. Bußmann (1983, S. 567) sieht in Vagheit ebenfalls einen „Teilaspekt von sprachlicher Mehrdeutigkeit“ und betrachtet sie somit als einen „komplementären Begriff zu Ambiguität“ (Bußmann, 1983, S. 567).

Definition 2.2.1 (Vagheit)
„Ein sprachlicher Ausdruck (Prädikat) ist vage, wenn für bestimmte Objekte durch die Bedeutung des Prädikats trotz Kenntnis der relevanten Tatsachen nicht eindeutig bestimmbar ist, ob sie unter den ausgedrückten Begriff fallen oder nicht, das heißt, wenn Grenzfälle existieren. Die Extension eines durch ein vages Prädikat ausgedrückten Begriffs hat also unscharfe Grenzen.“
(Kluck 2014, S. 14; Grice 1991, S. 177)

Berry et al. (2003, S. 14) diskutieren sprachliche Vagheit im Kontext von NFA, die als vage gelten, wenn nicht klar ist, wie sie zu messen sind. Sie nennen als Beispiel eine „*schnelle Antwortzeit*“ (engl. *fast response time*). Eine präzise Form der Beschreibung und Messung ist nicht gegeben und führt zu willkürlicher Quantifizierung, die wiederum offen lässt, ob der Kern der Anforderungen umgesetzt wurde (Berry et al., 2003, S. 14): Was für einen *Stakeholder* schnell sein kann, kann für einen anderen noch von moderater Geschwindigkeit sein.

Laut Löbner (2003, S. 62) ist Vagheit „bei allen Konzepten zu verzeichnen, die Merkmale beinhalten, deren Wert auf einer kontinuierlichen Skala variieren kann“. Ob etwas als groß, schnell oder schmackhaft bezeichnet wird, ist demnach „eine Frage des Grades auf einer offenen Skala“ (Löbner, 2003, S. 63). Laut Löbner (2003, S. 63) sind dabei „steigerbare Adjektive [...] generell vage“.

Beispiel 2.2.1 (Vagheit in einer Anforderung)
„Es müssen E-Mails mit großen Anhängen verschickt werden können“.

Das Phänomen der Vagheit kann, wie auch die lexikalische Ambiguität, auf Basis von Wortlisten erkannt werden. So nutzen beispielsweise Gleich et al. (2010) Vagheitslisten, die zum Teil auf der Arbeit von Berry et al. (2003) basieren. Auch sind, wie bereits angeführt, „steigerbare Adjektive [...] generell vage“ (Löbner, 2003, S. 63) – sie können daher als Merkmal potentieller Vagheit herangezogen werden. Darüber hinaus können nach Pinkal (1985, S. 223) nur relative Adjektive durch „sehr“ und „ziemlich“ modifiziert werden, was unter anderem in Geierhos und Bäumer (2017) zur maschinellen Erkennung von Vagheit herangezogen wird.

Löbner (2003, S. 63) weist darauf hin, dass Vagheit nicht isoliert von Ambiguität zu betrachten ist, sondern beides auch gemeinsam auftreten kann. So ist jede der Bedeutungsvarianten von „*schwer*“ („*gewichtig*“, „*schwierig*“, „*gravierend*“ usw.) für sich genommen vage. „Die zugrunde liegenden Skalen sind klar verschieden (Polysemie), aber wo genau im konkreten Fall die Grenze zwischen ‚*schwer*‘ und ‚*nicht schwer*‘ zu ziehen ist, ist eine Frage des Grades“ (Löbner, 2003, S. 63).

2.3 Unvollständigkeit

Unvollständigkeit ist in dieser Arbeit ebenfalls unter dem Aspekt der Softwareanforderungen zu definieren. Dabei ist, wie in Kapitel 2 bereits angeführt, nicht das

grundsätzliche Fehlen von Anforderungen unter Unvollständigkeit zu verstehen, sondern die lückenhafte Ausprägung einer benannten Anforderung (Auslassungen). Die Relevanz zeigt sich in der Beschreibung von Massey et al. (2014, S. 86), die hier als erster Definitionsversuch herangezogen wird (vgl. Definition 2.3.1).

Definition 2.3.1 (Unvollständigkeit)
Unvollständigkeit tritt auf, wenn eine Aussage nicht genügend Informationen beinhaltet, um eine einzige klare Auslegung zu ermöglichen.
(Massey et al., 2014, S. 86)

Diese Definition lässt allerdings offen, was unter „*enough information*“ zu verstehen ist. Diesbezüglich können Alshazly et al. (2014, S. 518) herangezogen werden, die den Fehlertyp der Auslassung (engl. *omission*) als „*necessary information related to the problem being solved by the software has been omitted from requirements document or are not complete*“ (Alshazly et al., 2014, S. 518) definieren.

Interessant an dieser Definition ist die Problemfokussiertheit, die das Spektrum fehlender Informationen eingrenzt. Eine Anforderung ist demnach unvollständig, wenn aufgrund von fehlender, zur Problemlösung notwendiger Informationen, keine klare Interpretation der Anforderung möglich ist. Firesmith (2005, S. 35) geht weiter und versteht unter „*necessary information*“ einen Informationsumfang, der eine Implementierung und Verifikation ohne zusätzliche Beschreibung ermöglicht. Ähnlich findet sich dies auch bei Wiegers (2005, S. 20): „Jede Anforderung muss die erwartete Funktionalität vollständig beschreiben. Sie muss alle Informationen enthalten, die der Entwickler benötigt, um diese Funktionalität zu entwerfen und zu implementieren“.

Fehlende Informationen, die aber keinen negativen Einfluss auf die beabsichtige Aussage einer Anforderungsbeschreibung haben, sind nach Lopes Margarido et al. (2011, S. 4) zu vernachlässigen.

Beispiel 2.3.1 (Prädikat-Argument-Struktur)
„Ich_{Arg_0} möchte $E\text{-}Mails_{Arg_1}$ mit großen Anhängen $\underline{senden}_{Präd.}$.“

Beispiel 2.3.1 zeigt eine Anforderungsbeschreibung, die das Senden von E-Mails als Gegenstand hat. Um diese Anforderung programmiertechnisch umsetzen zu können, müssen seitens der Softwareentwickler zwangsläufig Annahmen getroffen werden, da nicht angeführt wird, ob eine E-Mail beispielsweise formatiert oder an wen eine ausgehende E-Mail adressiert wird (z. B. an eine Person oder an eine Gruppe von Personen). Bei beiden Auslassungen handelt es sich demnach um notwendige Angaben, deren Fehlen erkannt und kompensiert werden muss. Hierzu werden in dieser Arbeit zwei Verfahren herangezogen: (1) Die Erkennung von Unvollständigkeit durch Hinzunahme von morpho-syntaktischem Wissen und (2) die Kompensation durch domänenspezifisches Wissen (s. Kapitel 5).

Die Erkennung von Unvollständigkeit durch Hinzunahme von morpho-syntaktischem Wissen (1) basiert in dieser Arbeit auf der Eigenschaft von Prädikaten, „Leerstellen um sich zu eröffnen, die in bestimmter Zahl und Art obligatorisch zu besetzen sind“ (Bußmann, 1983, S. 567). Dies wird auch als Valenz bezeichnet und entsprechende Leerstellen werden durch Argumente bestückt. „Je nachdem, wie viele [Argumente] ein Prädikat verlangt, bezeichnet man es als ein-, zwei- oder

dreistellig“ (Bußmann, 1983, S. 41). So hat beispielsweise das Prädikat „*senden*“ aus Beispiel 2.3.1 laut der Propbank (2010) drei Leerstellen zu besetzen (Arg_0, Arg_1, Arg_2), wobei nur zwei davon tatsächlich durch vorliegende Informationen aus dem Text instantiiert werden:

- (Arg_0) **sender** „*Ich*“
- (Arg_1) **sent** „*E-Mails mit großen Anhängen*“
- (Arg_2) **sent-to** –

Auf morpho-syntaktischer Ebene ist damit eine freie Leerstelle (engl. *non-instantiation*) identifiziert worden. Ungeklärt ist, ob es sich um eine obligatorische Information im Sinne der Problemlösung und damit um eine kompensationswürdige Argumentposition handelt.

Um dies zu klären, wird das domänenspezifische Wissen (2) herangezogen. Die Kompensation setzt voraus, dass Ressourcen existieren (z. B. Korpora, Ontologien), die für Anforderungen einer Domäne die obligatorischen Informationen (Problemfokussiertheit) enthalten. Wissensabfragen, welche Informationen zu einer spezifischen Anforderung erwartet werden und damit obligatorisch sind, können Unvollständigkeit aufdecken und beispielsweise durch hinterlegte Standardwerte kompensieren (s. Abschnitt 5.5.5).

Stand der Wissenschaft und Technik

3

Im Folgenden werden sowohl Arbeiten der maschinellen Anforderungsextraktion als auch der Erkennung und Kompensation von Ambiguität sowie Unvollständigkeit dargestellt. Zu Beginn wird im Sinne der weiteren Kapitelgliederung ein bestehendes NLP-Verarbeitungskonzept besprochen. Es folgen Arbeiten zur Anforderungsextraktion sowie zum Dokumententyp der Anforderungsbeschreibung (s. Abschnitt 3.2). Darauf aufbauend werden in Abschnitt 3.3.1 bestehende Disambiguierungsansätze dargelegt, bevor Ansätze zur Kompensation von Unvollständigkeit thematisiert werden (s. Abschnitt 3.3.2) und eine Betrachtung bestehender kombinierter Ansätze erfolgt (s. Abschnitt 3.3.3). Das Kapitel schließt mit Diskussion und Fazit.

3.1 Maschinelle Textanalyse im Kontext dieser Arbeit

Für Anforderungsbeschreibungen stellt Abbildung 3.1 einen in Geierhos und Bäumer (2017) beschriebenen sequenziellen Ablauf der maschinellen Textverarbeitung dar, der insbesondere auf die Extraktion semantischer Hauptkomponenten und die Kompensation von Ungenauigkeit sowie Unvollständigkeit abzielt.

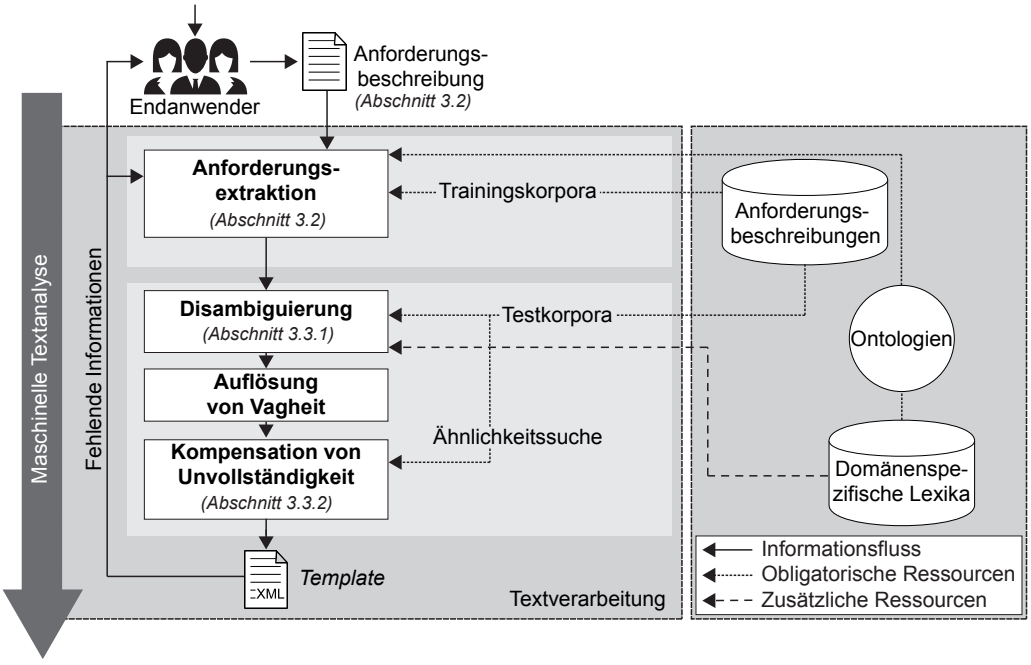


Abbildung 3.1: NLP-Verarbeitungsschritte im Arbeitskontext.
In Anlehnung an Geierhos und Bäumer (2017, S. 81)

Wenngleich auch nicht alle der in Abbildung 3.1 dargestellten Verarbeitungskomponenten Gegenstand dieser Arbeit sind (z. B. Auflösung von Vagheit), so eignet sich die Abbildung dennoch für die weitere Gliederung dieses Kapitels. Es werden zum einen Extraktions- und Kompensationskomponenten in einer beispielhaften Ausführungsreihenfolge dargestellt. Zum anderen wird der entsprechende Bedarf an natürlichsprachlichen Ressourcen ersichtlich.

Zwar wird in Abbildung 3.1 auf die Darstellung obligatorischer *Preprocessing*-Schritte[14] (z. B. Satzendeerkennung) verzichtet, nichtsdestotrotz ist deren Anwendung angesichts der in Kapitel 1.4 beschriebenen qualitativen Schwankungen innerhalb der Anforderungsbeschreibungen indiskutabel wichtig und wird auch in Geierhos und Bäumer (2017, S. 78 f.) diesbezüglich als elementar bezeichnet.

Ersichtlich wird, wie das konzipierte System Anforderungsbeschreibungen von Endanwendern entgegennimmt und daraufhin Anforderungen extrahiert, was unter anderem das Filtern von nebensächlichen Angaben beinhaltet. Der hier verarbeitete Dokumententyp der Anforderungsbeschreibung wird bereits in Kapitel 1.4 behandelt und im folgenden Szenario des OTF-*Computings* weiter vertieft (s. Abschnitt 3.2). Den identifizierten Herausforderungen der maschinellen Verarbeitung von Anforderungsbeschreibungen begegnend, werden anschließend geeignete Ansätze der Anforderungsextraktion und der einhergehende Ressourcenbedarf diskutiert.

In Abbildung 3.1 folgen auf die Anforderungsextraktion die Disambiguierung, die Auflösung von Vagheit und die Kompensation von Unvollständigkeit als isolierte Verarbeitungskomponenten. Diese Komponenten dienen der Anforderungsaufbereitung, bevor eine Ergebnisausgabe in Form eines *Templates* geschieht. Bestehende Ansätze der Disambiguierung werden in Abschnitt 3.3.1 behandelt und umfassen sowohl die lexikalische, syntaktische als auch die referentielle Disambiguierung.

Neben Ungenauigkeit ist Unvollständigkeit ein zentrales Thema der vorliegenden Arbeit, zu dem es bereits ebenfalls eine Vielzahl von Vorarbeiten gibt. Diese Arbeiten werden in Abschnitt 3.3.2 behandelt und decken die Erkennung und Kompensation von Unvollständigkeit ab.

Es existieren darüber hinaus Ansätze, die mehrere Erkennungs- sowie Kompensationsschritte kombinieren. Diese kombinierten Ansätze werden in Abschnitt 3.3.3 gesondert betrachtet, da sie mögliche Synergieeffekte sowie Limitationen aufzeigen und aufgrund ihres integrativen Systemcharakters besonderen Einfluss auf die abschließende Diskussion und das Fazit in diesem Kapitel haben (s. Abschnitt 3.4).

3.2 Anforderungsextraktion im OTF-Computing

Um die zentrale Rolle der Anforderungsextraktion in dieser Arbeit besser zu verstehen, ist ein Blick auf die zu verarbeitenden Dokumententypen notwendig. Wie in den Abschnitten 1.3 und 1.4 dargestellt, handelt es sich bei **Anforderungsbeschreibungen** um eine Unterform der Anforderungsdokumentation bzw. -spezifikation, die insbesondere durch ihren informalen Charakter und explizit durch zu erwartende Ungenauigkeit und Unvollständigkeit geprägt ist. Sie ist damit das Ergebnis individueller Rahmenbedingungen (z. B. Vorwissen der *Stakeholder*). Als Beispiel für diesen

[14] Diese notwendigen *Preprocessing*-Schritte werden gesondert in Kapitel C.1 dargestellt.

Dokumententyp können Anforderungsbeschreibungen der *Open Source*-Bewegung dienen, denen Eigenschaften wie ein hoher Anteil nebensächlicher, erläuternder Kommunikation und ein hoher Anteil an Ambiguitäten zugeschrieben werden. Diese Eigenschaften sind insbesondere auf die große Anzahl an (heterogenen) *Stakeholdern*, deren unterschiedlichen Erfahrungsständen und Fachwissen sowie ein fehlendes oder minimales Regelwerk zurückzuführen (Gill et al., 2014; Vlas und Robinson, 2011; Laurent und Cleland-Huang, 2009). Anforderungsbeschreibungen rücken damit näher an einen informalen Dokumententyp heran, der allgemein als *User Generated Content* (UGC) bezeichnet wird (s. insb. Moens et al., 2014, S. 7).

Wird die Thematik der Anforderungsbeschreibung bzw. -spezifikation im Kontext des OTF-*Computings* betrachtet, liegt der Fokus bestehender Arbeiten auf der Anwendung und Entwicklung semi-formaler bzw. formaler Spezifikationssprachen für *Softwareservices*[15]. In erster Linie umfasst das die *Service Specification Language* (SSL), die explizit zur umfassenden Spezifikation von *Services* entwickelt wurde und unter anderem FA sowie NFA abdeckt (Platenius et al., 2016, S. 5 ff.). Darüber hinaus entwickeln Huma et al. (2012) eine UML-basierte Beschreibungssprache, deren Vorteil es ist, dass *Stakeholder* sich der bereits etablierten und gut dokumentierten UML bedienen können und somit der Einarbeitungsaufwand geringer ausfallen kann.

Wie allerdings in Abschnitt 1.3.4 und darüber hinaus von Geierhos et al. (2015, S. 277) angemerkt wird, ist jede Form der semi-formalen bzw. formalen Spezifikation für Endanwender ungeeignet. Endanwender verfügen nicht über die notwendigen Fachkenntnisse, kennen darüber hinaus gewisse Angaben (z. B. Vor- und Nachbedingungen eines *Services*) nicht (vollständig) und können diese daher erst recht nicht formal spezifizieren (Ferrari et al., 2014). Im Vergleich dazu können Anforderungsbeschreibungen von Endanwendern verfasst werden (s. Abschnitt 1.4), sind aber auch im OTF-Kontext unstrukturiert, oftmals fehlerhaft (im Sinne von Grammatik und Rechtschreibung), unvollständig und mehrdeutig. Dies steht im Kontrast zu den bestehenden (semi-)formalen Spezifikationsmöglichkeiten, die bisher im OTF-*Computing* zur Verfügung stehen[16].

Eine weitere Herausforderung im Umgang mit Anforderungsbeschreibungen sind fehlende linguistische Ressourcen, die Qualität und Eigenschaften aufweisen, wie sie bei Anforderungsbeschreibungen im OTF-*Computing* zu erwarten sind. Tichy et al. (2015) beschreiben diese Situation zutreffend im Hinblick auf natürlichsprachliche Anforderungen: „*Textbooks contain few examples and they seem to be written by the authors or copied from other textbooks. Many examples about NLP requirements processing use an artificial, strongly restricted language*“ (Tichy et al., 2015, S. 161). Ein Problem dieser selbst kreierten Anforderungsbeschreibungen ist, dass sie oftmals idealtypisch sind oder bewusst auf einen bestimmten Problemfall hin gestaltet wurden – sie unterscheiden sich daher teils erheblich von „echten“ Anforderungen.

[15] „*A service is a software component that is deployed and running on a service provider's platform. One example for a service is Google Maps*“ (Platenius, 2016, S. 11).

[16] An dieser Stelle sei angemerkt, dass Unvollständigkeit und Ungenauigkeit auch bei (semi-)formalen Methoden nicht vollständig auszuschließen sind. So ist Unvollständigkeit im *OTF-Computing* sowohl auf der Anfrageseite (Benutzeranforderungen) als auch auf der Anbieterseite (Servicespezifikationen) vorzufinden. Die Gründe sind heterogen und reichen von Unwissenheit auf der Anfrageseite bis hin zu Auslassung von Informationen zur Wahrung von Geschäftsgeheimnissen auf Anbieterseite (Platenius et al., 2015, S. 7; Platenius, 2013, S. 716 f.).

Die **Anforderungsextraktion** im OTF-*Computing* sieht sich daher dem Problem gegenüber, dass Trainingsdaten fehlen, auf denen die Merkmale von Anforderungsbeschreibungen gelernt oder regelbasierte Ansätze entwickelt werden können. Es bedarf daher der Erstellung linguistischer Ressourcen (s. Kapitel 6), welche die Eigenschaften von Anforderungsbeschreibungen adäquat abbilden, um Ansätze der Anforderungsextraktion entwickeln und evaluieren zu können.

Derzeit existieren nur wenige Arbeiten, die sich der Anforderungsextraktion aus qualitativ stark schwankenden Texten widmen. So führen beispielsweise Vlas und Robinson (2011) eine Untersuchung von unstrukturierten und informalen Anforderungsbeschreibungen im Bereich der *Open Source*-Software durch, um mehr über frei formulierte Anforderungstexte zu lernen. Darüber hinaus stellt Dollmann (2016) bzw. Dollmann und Geierhos (2016) ein *Tool* namens REaCT zur Verfügung, welches Verfahren des maschinellen Lernens nutzt, um *On-Topic* Aussagen in Anforderungsbeschreibungen zu erkennen und die wesentlichen Bestandteile funktionaler Anforderungen in ein definiertes *Template* zu übertragen. Dollmann (2016) unterteilt dabei die Anforderungsbeschreibungen in einzelne Sätze und übergibt diese an die Klassifikationskomponente, die Sätze in *Off-Topic* und *On-Topic* unterteilen kann. Handelt es sich bei einem klassifizierten Satz um *On-Topic* und damit um funktionale Anforderungen, wird die Extraktion von Attribut-Wert-Paaren vorgenommen, mit dem Ziel, ein vorgegebenes *Template* iterativ zu befüllen: Die wichtigsten Elemente des *Templates* sind die Komponente (Subjekt), die Aktion (Prädikat) und das Objekt (Objekt). Aktionen beschreiben, was eine Komponente leisten soll und Objekte beschreiben, worauf sich die Aktionen beziehen. Sowohl Komponenten als auch Objekte können in den Anforderungsbeschreibungen weiter konkretisiert werden, wofür die Felder Verfeinerung der Komponente und Verfeinerung des Objektes vorgesehen sind. Darüber hinaus können Vor- und Nachbedingungen (z. B. Zeitrestriktionen) existieren, die für die Ausführung von Aktionen gelten müssen oder sollen und die im *Template* als Bedingungen angegeben sind.

Über genannte Beiträge hinaus existieren mehrheitlich Ansätze, die sich der Analyse und Kompensation qualitativ hochwertiger(er) Anforderungsbeschreibungen widmen oder weitreichende Annahmen zur Textqualität treffen. Sie sind somit für den Anwendungsfall in dieser Arbeit ungeeignet (z. B. Deeptimahanti und Sanyal, 2011).

3.3 Umgang mit Ambiguität und Unvollständigkeit

Dieser Abschnitt beinhaltet den Stand der Wissenschaft und Technik zur Erkennung und Kompensation von Ambiguität (s. Abschnitt 3.3.1) und Unvollständigkeit (s. Abschnitt 3.3.2). Die Themengebiete des *Requirements Engineerings* und des *Natural Language Processings* sind dabei durch gemeinsame Fragestellungen und Verfahren eng verbunden (Berzins et al., 2008).

3.3.1 Disambiguierung im Anforderungskontext

Eine Vielzahl an Veröffentlichungen thematisiert Ambiguität in Anforderungsbeschreibungen (Pekar et al., 2014; Umber und Bajwa, 2011; Kamsties, 2005; Osborne und

MacNish, 1996). Dabei zeigt sich, dass ambige Softwareanforderungen zu vielseitigen Problemen in der Softwareentwicklung führen können (Pekar et al., 2014, S. 242).

So wird häufig sowohl der Projekterfolg (Standish Group International, 1995) als auch die Kundenzufriedenheit (Sommerville, 2007, S. 121) durch Ambiguität als gefährdet angesehen. Allerdings reicht die Spannweite der Veröffentlichungen hierzu von „*[...] ambiguity is a more complex phenomenon than is often recognized in the literature*“ (Kamsties et al., 2001, S. 1) bis hin zu „*[...] requirement ambiguity did not cause many defects*“ (Philippo et al., 2013, S. 78). Ein möglicher Einfluss des Ambiguitätsphänomens auf die Anforderungsqualität und somit auf den Projekterfolg gilt daher in der Literatur durchaus als umstritten (s. insb. Philippo et al., 2013).

Die im Rahmen dieser Arbeit betrachteten Anforderungsbeschreibungen (s. Abschnitt 1.4) sind jedoch, bedingt durch die Verwendung von natürlicher Sprache (s. Abschnitt 1.3.1) und geringes bzw. fehlendes Vorwissen der Endanwender, prädestiniert für Ambiguität. So kommt es vor, dass eine Software gemäß vorgegebener Anforderungsbeschreibungen entwickelt wird, diese jedoch nicht den intendierten Anforderungen der *Stakeholder* entspricht (Kamsties und Paech, 2000, S. 2) oder sogar Fehler enthält (Firesmith, 2007, S. 19). Insbesondere im Fall des *OTF-Computings*, der eine automatisierte Komposition von *Services* vorsieht, kann Ambiguität zu mangelhafter Software führen (Geierhos et al., 2015, S. 279).

Die Ambiguitätserkennung kann dabei manuell – beispielsweise mit Hilfe von Wortlisten oder Checklisten (z. B. Kamsties et al., 2001), unterstützt durch Software oder gänzlich vollautomatisiert erfolgen. Dies deutet bereits an, dass es eine Vielzahl an Methoden gibt, die sich nur schwer vollumfänglich gegenüberstellen lassen (Bano, 2015; Shah und Jinwala, 2015). Zum einen gibt es Ansätze, die eine Vielzahl an Ambiguitätsformen erkennen können (generalisiert) und zum anderen existieren Verfahren, die sich auf die Erkennung einer Form (z. B. lexikalische Ambiguität, s. Abschnitt 2.1) beschränken (spezialisiert). In beiden Fällen werden linguistische Ressourcen benötigt, welche die Eigenschaften des jeweiligen Ambiguitätsphänomens abdecken. Sie werden im Folgenden zur besseren Verständlichkeit der Disambiguierungsansätze dargestellt.

3.3.1.1 Linguistische Ressourcen

Linguistische Ressourcen haben im NLP elementaren Charakter, da sie die notwendige Wissensbasis darstellen, auf der die jeweiligen Methoden arbeiten. Der Schwerpunkt liegt im Folgenden auf den Ressourcen, die für die lexikalische, syntaktische und referentielle Disambiguierung von Bedeutung sind.

Ressourcen lexikalischer Disambiguierung

Eine Ressource, die vielfach zur lexikalischen Disambiguierung aber auch in anderen NLP-Kontexten herangezogen wird, ist **WordNet**. Hierbei handelt es sich um ein frei verfügbares lexikalisch-semantisches Netz (Datenbank) für die englische Sprache (Miller, 1995). Es wird seit 1985 am *Cognitive Science Laboratory* der *Princeton University* entwickelt und enthält semantische sowie lexikalische Beziehungen zwischen einzelnen Wörtern (Nomina, Verben und Adjektive sowie Adverbien).

Dabei werden nicht die Wörter als solche untereinander in Beziehung gesetzt, sondern spezifische Lesarten (*Synsets*), was zu einer semantischen Disambiguation

führt. Als Beziehungen werden dabei unter anderem Synonymie, Antonymie und Hyponymie sowie Meronymie abgebildet, wobei die am häufigsten abgebildete Beziehung zwischen *Synsets* die Hyponymie ist. Einen Überblick über den Umfang von Wordnet gibt die in der WordNet-Dokumentation enthaltene Statistik (WordNet, 2010). WordNet enthält insgesamt 147.278 Einträge, wovon einige in mehrere syntaktische Kategorien fallen (vgl. Tabelle 3.1).

POS	*Unique Strings*	*Synsets*	*Word-Sense Pairs*
Nomina	117798	82115	146312
Verben	11529	13767	25047
Adjektive	21479	18156	30002
Adverben	4481	3621	5580

Tabelle 3.1: WordNet 3.0 Statistik (Verteilung der Einträge)

WordNet wird in dieser Arbeit primär zur *Word Sense Disambiguation* (WSD) eingesetzt und ist auf die englische Sprache spezialisiert. Für die deutsche Sprache steht GermaNet zur Verfügung, welches identisch aufgebaut ist (Henrich und Hinrichs, 2010; Hamp und Feldweg, 1997).

Eine umfangreiche und mehrsprachige Alternative zu WordNet ist **BabelNet**. Hierbei handelt es sich um ein mehrsprachiges Wörterbuch, das lexikalisches und enzyklopädisches Wissen zu Einträgen bereitstellt (Flati und Navigli, 2014, S. 11). Für die englische Sprache liegen beispielsweise 11,8 Millionen Wörter in ihrer Grundform vor, wobei davon 413.144 Wörter als polysem und 11,4 Millionen Wörter als eindeutig klassifiziert sind. Darüber hinaus fungiert BabelNet als semantisches Netz, welches Konzepte und *Named Entities* (NE) über semantische Beziehungen (Babel *Synsets*) verbindet (s. Abbildung 3.2).

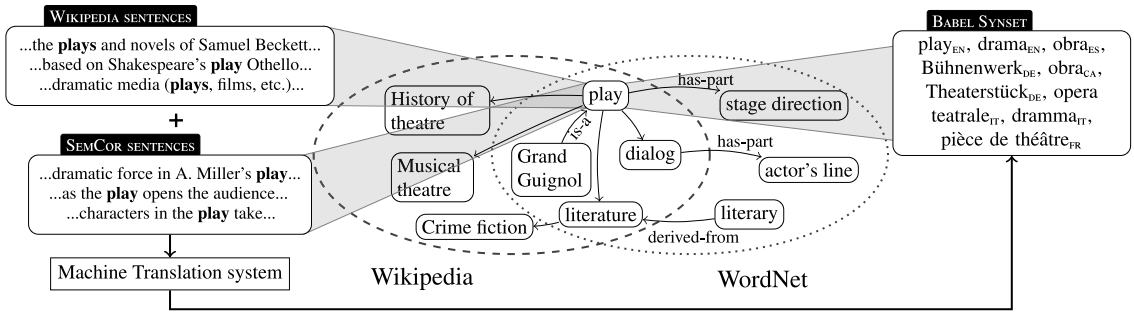


Abbildung 3.2: BabelNet als semantisches Netz.
Entnommen aus Navigli und Ponzetto (2012a, S. 221)

Jedes *Synset* enthält dabei eine bestimmte Lesart zusammen mit Synonymen in verschiedenen Sprachen. Über 1,5 Millionen *Synsets* sind in mindestens einer Domäne klassifiziert („*Sharepoint*“ → „*Computing*“), wobei die vorliegenden Informationen oftmals weit über die bloße Angabe einer Domäne hinausgehen (z. B. Kategorien: „*Windows software, Microsoft*“, Genre: „*Content Management Systems*“).

Tabelle 3.2 enthält eine allgemeine Statistik[17], die einen Überblick über den Umfang der Ressource gibt. In der Version 3.6 unterstützt BabelNet 271 Sprachen, darunter alle europäischen Sprachen. Für diese Arbeit ist die umfangreiche Sprachunterstützung gerade im Hinblick auf die Erweiterbarkeit und Adaptierbarkeit des Konzepts von Bedeutung (s. Abschnitt 7.4.2).

Mit 13,8 Millionen *Synsets* und insgesamt 745,9 Millionen Lesarten erreicht BabelNet darüber hinaus eine bemerkenswerte sprachübergreifende Abdeckung. Das semantische Netz beinhaltet 380,2 Millionen lexikalisch-semantische Beziehungen, wie sie auch in WordNet vorzufinden sind.

Aspekt	**Ausprägung**
Babel *Senses*	745,9
Babel *Synsets*	13,8
Glossareinträge	40,7
Komposita	0,7
Konzepte	6,1
lexikalisch-semantische Beziehungen	380,2
NE	7,7
RDF *Triples*	1.971,7

Tabelle 3.2: BabelNet 3.6 (Statistik, Angaben in Millionen)

Unter anderem wird BabelNet zur Disambiguierung (Navigli und Ponzetto, 2012b) und zum *Entity Linking* (EL) verwendet (Moro et al., 2014b). Letzteres zum Beispiel im verwandten Babelfy-System (s. Abschnitt 3.3.1.2).

Ressourcen syntaktischer Disambiguierung

Wie angeführt, ist die natürliche Sprache gleich mehrfach anfällig für Ambiguität. Im Hinblick auf die Erkennung und Kompensation syntaktischer Ambiguität, die beim *Parsing* eines Satzes auftreten kann, wird auch von einem Suchprozess gesprochen: Ein statistischer NLP-Algorithmus sucht auf Grundlage definierter grammatikalischer Regeln verschiedene Kombinationswege für eine Satzstruktur und disambiguiert somit die Eingabe über die wahrscheinlichste Kombination (Allen, 1995, S. 47). Die dafür benötigten Wahrscheinlichkeiten und Regeln müssen zuvor gelernt werden, wobei zum Beispiel sogenannte *Treebanks* genutzt werden können (Theda, 2017, S. 19 ff.).

Treebanks[18] (auch: Baumbanken) „als spezielle Form von Korpora sind ein fester Bestandteil der Computerlinguistik, da sie detaillierte linguistische Informationen kodieren [(vgl. Abbildung 3.3)]. Sie werden dabei als eine Sammlung von Einheiten (meist Sätzen) verstanden, deren syntaktische Satzstruktur annotiert ist“ (Carstensen et al., 2010, S. 492). Grundsätzlich besteht ein Korpus dabei aus Text, einem Annotationsschema und beschreibenden Metadaten, wobei das Annotationsschema entweder konstituenten- oder dependenzbasiert gewählt wird (Carstensen et al., 2010, S. 492; Hajičová et al., 2010, S. 171; Theda, 2017, S. 19). Die Baumbanken werden

[17] Siehe weiterführend: `http://babelnet.org/stats` (Stand: 12.01.17).
[18] Die syntaktische Struktur wird traditionell als Baumstruktur kodiert.

dabei überwiegend „manuell oder semi-automatisch erstellt und [haben] folglich einen kleineren Umfang als ein automatisch geparstes Korpus“ (Carstensen et al., 2010, S. 493). Ihre Erstellung gilt daher gemeinhin als zeit- und kostenintensiv, weshalb Qualitätsmerkmalen wie Wiederverwendbarkeit und Konsistenz bei der Bankenerstellung ein besonderer Stellenwert zugesprochen wird (Carstensen et al., 2010, S. 493, 495 f.). Eine Übersicht über bestehende, teils verwandte Baumbanken gibt Theda (2017, S. 17 ff.), deren Arbeit im Folgenden herangezogen wird[19]:

„Bis Mitte der 90er Jahre wurden vor allem [...][konstituentenbasierte] Annotationsschemata verwendet. Hier werden die hauptsächlichen syntaktischen Kategorien wie NP oder VP in einer Baumstruktur annotiert“ (Carstensen et al., 2010, S. 493). Als etablierte und verbreitete Ressource gilt dabei die Penn Treebank (Marcus et al., 1993; Marcus et al., 1994), die mit mehr als 50.000 annotierten Sätzen sehr umfangreich ist und von einer Vielzahl an statistischen *Parsern* als Ressource herangezogen wird[20]. Bei den Sätzen handelt es sich um englischsprachige Sätze, die hinsichtlich POS-*Tags* und syntaktischer Struktur annotiert sind. Neben dieser englischsprachigen Version stehen auch adaptierte Sprachversionen zur Verfügung, wobei auf die Penn Arabic Treebank (Maamouri et al., 2004) und die Penn Chinese Treebank (Xue et al., 2005) als Beispiele zu verweisen ist (Theda, 2017, S. 20).

Weitere konstituentenbasierte *Treebanks* sind die BulTreeBank (Simov, 2004) und die LinGO Redwoods (Oepen et al., 2004). Für die deutsche Sprache existiert beispielsweise das TIGER Korpus (Brants et al., 2002) sowie die, exemplarisch in Abbildung 3.3 als Auszug abgebildete, Tübinger Baumbank des Deutschen / Zeitungskorpus (TüBa-D/Z) (Telljohann et al., 2015).

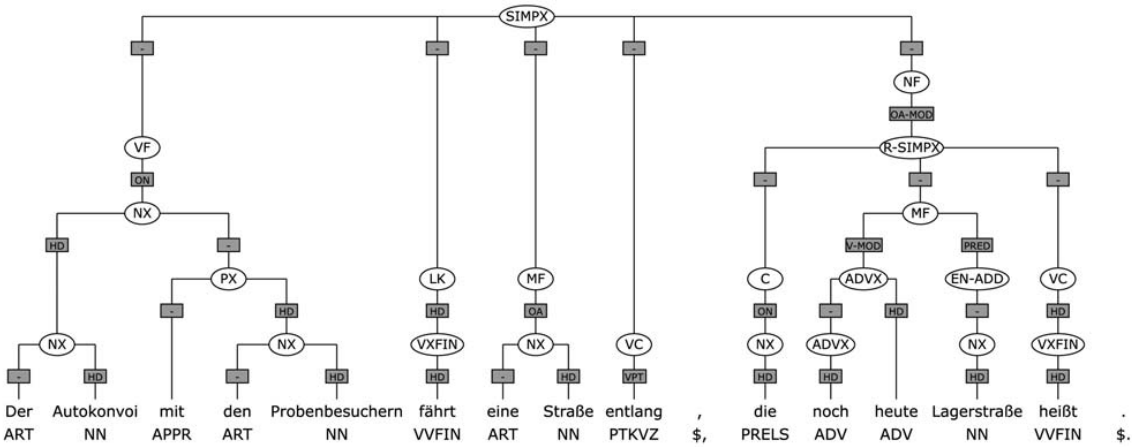


Abbildung 3.3: Ein beispielhafter Satz aus der TüBa-D/Z.
Entnommen aus Carstensen et al. (2010, S. 501)

Darüber hinaus stehen dependenzbasierte *Treebanks* zur Verfügung, die zunehmend Verwendung finden. Ziel ist es hier, die gerichteten Abhängigkeiten zwischen zwei

[19] Eine weitere umfangreiche Übersicht findet sich in der englischsprachigen Wikipedia. Siehe: `https://en.wikipedia.org/wiki/Treebank` (Stand: 08.02.17).

[20] Theda (2017) nennt u. a. Charniak (1997), Collins (1996) sowie den Standford Parser (Klein und Manning, 2003) und Parsey McParseface (Andor et al., 2016) als beispielhafte *Parser*.

Wörtern abzubilden (Carstensen et al., 2010, S. 494). Als prominenter Vertreter ist dabei die *Prague Dependency Treebank* (PDC) zu nennen (Hajič et al., 2001).

Ressourcen referentieller Disambiguierung

„Im Gegensatz zu syntaktisch annotierten Korpora [...] stehen für die Anaphernresolution nur sehr wenige annotierte Korpora zur Verfügung“ (Carstensen et al., 2010, S. 407). Dabei ist zu beachten, dass die Anaphern- und Koreferenzauflösung auf unterschiedliche Weise durchgeführt werden kann, wobei zwischen linguistischen Ansätzen, Heuristiken und Methoden des maschinellen Lernens unterschieden wird (Carstensen et al., 2010, S. 404). Nicht alle dieser Herangehensweisen sind dabei auf Ressourcen angewiesen (s. Abschnitt 3.3.1.4), weshalb im Folgenden der Fokus vermehrt auf dem Ressourcenbedarf der Methoden des maschinellen Lernens liegt.

Zum Training und zur Evaluation dieser Methoden werden im Rahmen der Koreferenzauflösung spezielle Korpora verwendet (s. insb. Recasens Potau, 2010, S. 10). Beispiele sind die Korpora ACE, MUC sowie die TüBa-D/Z, die entsprechende Annotationsebenen[21] beinhalten. Eine Auswahl[22] findet sich in Tabelle 3.3.

Korpus	**Quelle**	**Domäne**	**Token**
ACE2004	Doddington et al. (2004)	Zeitungstexte	189.620
MUC6	Grishman und Sundheim (1996)	Zeitungstexte	30.000
OntoNotes1	Weischedel et al. (2007)	Gemischt	300.000
OntoNotes5	Weischedel et al. (2012)	Gemischt	1.745.000
3S12	Schäfer et al. (2012)	Wissenschaft	1.326.147
TüBa-D/Z	Telljohann et al. (2015)	Zeitungstexte	1.787.801
WikiCoref	Ghaddar und Langlais (2016)	Enzyklopädieartikel	60.000

Tabelle 3.3: Annotierte Korpora zur Koreferenzauflösung (Auswahl). In Anlehnung an Ghaddar und Langlais (2016, S. 140)

Als grundlegende und etablierte Ressourcen sind die ACE- sowie die MUC-Korpora zu nennen. Sie sind mit Koreferenzrelationen annotiert und bestehen sowohl aus Trainings- als auch aus Testdatensätzen. Ihnen wird jedoch vorgehalten (z. B. Guha et al., 2015, S. 1108; Chaimongkol et al., 2014, S. 3187), nur begrenzt adaptierbar zu sein, da sie überwiegend auf Zeitungstexten (und Ähnlichem) basieren und damit zum Beispiel nicht als Trainingsdaten für wissenschaftliche Texte geeignet sind. Aus diesem Grund existieren weitere Korpora, die hinsichtlich verschiedener Fragestellungen und Domänen konzipiert sind. Als aktuelleres Beispiel ist das Korpus von Guha et al. 2015 zu nennen, welches wissenschaftliche Texte mehrerer Forschungsgebiete umfasst.

Zum Vergleich der Ressourcen wird oftmals die Gesamtanzahl an *Token* herangezogen, wobei fraglich ist, ob sich nicht problemspezifische Annotationen und deren Anzahl eher zum Vergleich eignen. Als Beispiel kann die TüBa-D/Z herangezogen werden, welche 54.382 Koreferenz-Relationen, 50.721 anaphorische Relationen und 1.582 kataphorische Relationen aufweist[23]. Da aber nur wenige Arbeiten solche de-

[21] Beispielsweise enthält die TüBa-D/Z Annotationsebenen zu Anaphern und Koreferenz-Relationen.
[22] Die Auswahl wurde hinsichtlich unterschiedlicher Domänen und Datenumfang getroffen.
[23] Siehe: `http://sfs.uni-tuebingen.de/ascl/resources/corpora/tueba-dz.html` (Stand: 15.02.17).

taillierten Angaben beinhalten und ein Vergleich damit nicht durchzuführen ist, wird in dieser Arbeit ebenfalls auf *Token* als Vergleichseinheit zurückgegriffen, um die unterschiedlichen Ressourcenumfänge darzustellen (vgl. Tabelle 3.3). Dabei zeigt sich, dass bei der Auswahl geeigneter Ressourcen nicht nur auf die Domäne, sondern auch auf den Ressourcenumfang zu achten ist. Einen großen Umfang und eine nennenswerte thematische Abdeckung verspricht OntoNotes5.

Wie Carstensen et al. (2010) weiterhin anmerken, benötigen Methoden des maschinellen Lernens über diese Ressourcen hinaus ein „gewisses Maß an Domänen- oder Weltwissen“ (Carstensen et al., 2010, S. 404), welches beispielsweise der Auflösung definiter NPs dienlich ist. Dieses kann ergänzend aus Ressourcen wie WordNet, YAGO oder Wikipedia abgerufen werden (s. z. B. Rahman und Ng, 2011). Auch existieren Ressourcen, die mehrere Annotationsebenen umfassen, Zwischenverbindungen abbilden (vgl. Abbildung 3.4) und somit die Anwendung von Domänen- oder Weltwissen ermöglichen bzw. erleichtern.

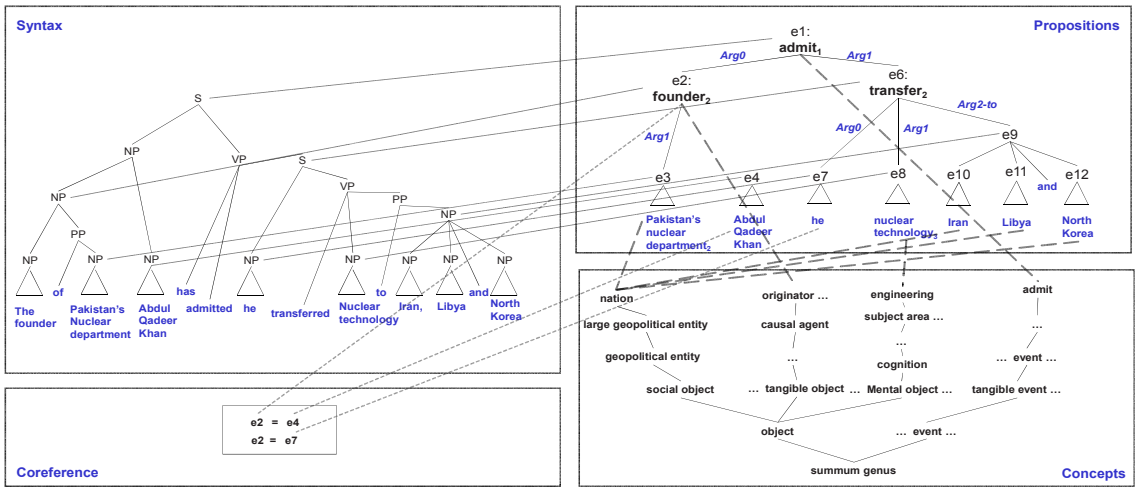


Abbildung 3.4: Zwischenverbindungen einzelner Annotationsebenen (OntoNotes). In Anlehnung an Weischedel et al. (2011, S. 61)

3.3.1.2 Lexikalische Ambiguität

Lexikalische Ambiguität und deren Disambiguierung (auch: *Word Sense Disambiguation*, s. Abschnitt 2.1.1) wird im Anforderungskontext vielseitig diskutiert. Nach Bano (2015, S. 23) widmet sich dabei die Mehrheit der Publikationen der reinen Ambiguitätserkennung, während sich eine deutlich geringere Anzahl an Veröffentlichungen mit der lexikalischen Disambiguierung in diesem Anwendungsgebiet befasst. Dabei ist der Diskussionsgegenstand, nämlich die Ambiguität einzelner Lexeme, nicht so eindeutig in der Literatur definiert, wie es auf den ersten Blick erscheint.

Eine oftmals herangezogene Definition von lexikalischer Ambiguität ist die von Berry et al. (2003, S. 10): „*Lexical ambiguity occurs when a word has several meanings*“. Dennoch wird der Begriff an vielen Stellen inkonsistent verwendet. So definieren zum Beispiel Nigam et al. (2012) den Begriff unter Bezug auf Berry et al. (2003), fassen in der Analyse aber auch Fälle von Vagheit unter dem Begriff zusammen (Nigam et al., 2012, S. 354). Auch Huertas und Juárez-Ramírez (2012, S. 374) bezeichnen

vage Begriffe (z. B. Steigerung von Adjektiven und Adverbien) als lexikalisch ambig. Im Folgenden werden daher auch Verfahren der lexikalischen Disambiguierung im Anforderungskontext herangezogen, die von der ursprünglichen Definition abweichen.

Die **Erkennung von lexikalischer Ambiguität** ist (z. B. neben der syntaktischen und referentiellen Disambiguierung) ein Teilgebiet der Disambiguierung und geschieht vielfach über einen Abgleich einzelner Lexeme mit (für z. B. eine Domäne oder Sprache) geeigneten Ressourcen (z. B. Korpora, Wörterbücher)[24]. Hierbei wird für ein erkanntes Wort (z. B. Computer) in einer gegebenen Ressource nachgeschlagen, ob es zugehörige Lesarten gibt (z. B. „*a machine for performing calculations automatically*“) und wenn ja, ob mehr als eine existiert (in diesem Fall sind es zwei Lesarten) und das Wort damit als potentiell ambig gilt. Eine vereinfachte Variante davon ist, Wörter mit einer Liste von bereits als ambig bekannter Wörter abzugleichen.

Ein solcher Abgleich zwischen Lexemen und einer Liste von ambigen Wörtern erfolgt zum Beispiel bei Lami (2005), Gleich et al. (2010), Nigam et al. (2012), Génova et al. (2013) sowie Tjong und Berry (2013). Kiyavitskaya et al. (2008), Matsuoka und Lepage (2011) sowie Rojas und Sliesarieva (2010) greifen auf WordNet als linguistische Ressource zurück. Ergänzend wird bei Rojas und Sliesarieva (2010) noch VerbNet (Kipper-Schuler, 2005) als Ressource hinzugezogen.

Ein Vorteil einer eigenen oder modifizierbaren Ressource ist, dass sowohl spezifisches Vokabular (z. B. Fachtermini) als auch Eigennamen (z. B. „*Sharepoint Server*“) abgebildet werden können, die gegebenenfalls einer besonderen Disambiguierung bedürfen. WordNet ist zwar nicht auf technisches Fachvokabular spezialisiert, hat aber einen sehr umfangreichen Wortschatz und deckt damit bereits viele technische Fachbegriffe ab (z. B. „*mailing list*“). Bei Eigennamen ist WordNet allerdings als Ressource aufgrund seiner geringen Abdeckung nicht heranzuziehen – wohl wissend, dass es Verfahren zur Erweiterung von WordNet gibt (z. B. Toral et al., 2008; Magnini et al., 2002). Da in der Regel keine Beschränkung der Domäne vorgenommen wird, ist eine Erweiterung jedoch in den meisten Fällen kaum zufriedenstellend durchzuführen.

Exemplarisch für Verfahren zur reinen Erkennung lexikalischer Ambiguität wird im Folgenden das vielfach diskutierte *Systemized Requirements Engineering Environment* (SREE) von Tjong und Berry (2013) vorgestellt. Es handelt sich dabei um eine Applikation, die potentielle Ambiguität (die Autoren verstehen hierunter sowohl Ambiguität als auch Ungenauigkeit, Unbestimmtheit und Vagheit) in natürlichsprachlichen Anforderungen durch Abgleich mit entsprechenden Indikatorlisten aufdecken kann. SREE besteht dabei aus zwei Hauptkomponenten: (1) Dem *Ambiguity Indicator Corpus* (AIC) nach Tjong (2008) und dem (2) *Lexical Analyzer*. Während (1) ein Korpus darstellt, der „Indikatoren“[25] für ambige Lexeme enthält, handelt es sich bei (2) um die Komponente, die Anforderungen tokenisiert, mit dem AIC abgleicht und den Endanwender über potentielle Ambiguitäten informiert.

Wie unter anderem das Beispiel SREE zeigt, ist bisher stets eine hohe Benutzerinteraktion bei der Kompensation potentieller Ambiguitäten erforderlich, wenngleich die Erkennung von potentiell ambigen Wörtern eine große Hilfe darstellen kann

[24] Für weitere Angaben zur fundamentalen Rolle der Ressourcen im WSD siehe Navigli (2009, S. 6).

[25] Es handelt sich beim AIC um eine Sammlung von zehn Subkorpora: *Continuance*, *Coordinator*, *Directive*, *Incomplete*, *Optional*, *Plural*, *Pronoun*, *Quantifier*, *Vague* und *Weak*. Das Korpus *Quantifier* enthält z. B. die Wörter: *all*, *any*, *few*, *little*, *many*, *much*, *several* und *andsome*.

(Tjong und Berry, 2013, S. 82). Allerdings weisen Shah und Jinwala (2015, S. 1) darauf hin, dass das manuelle Auflösen von Ambiguität ein ermüdender, zeitraubender, fehleranfälliger und schlussendlich teurer Vorgang ist (Popescu et al., 2008; Berry et al., 2003). Ein (semi-)automatisiertes Vorgehen ist demnach zur Wahrung der Qualität und Motivation sowie zur Begrenzung der Kosten erforderlich. An dieser Stelle ist auf die Arbeit von Mihalcea (2003) hinzuweisen, in der lexikalisch ambige Lexeme erkannt werden, aber nur dann als lexikalisch ambig gelten, wenn sie nicht durch den unmittelbaren Kontext (im Sinne von eindeutigen Lexemen in der Nachbarschaft) disambiguiert werden können. Dies reduziert die Anzahl potentiell ambiger Lexeme für Disambiguationsverfahren und kann somit die Performanz verbessern.

Neben dem Abgleich mit (umfangreichen) linguistischen Ressourcen existieren somit auch Möglichkeiten, die ohne externe Ressourcen anzuwenden sind. Diesbezüglich widmet sich beispielsweise der Forschungsbereich *Word Sense Induction* (WSI) der Idee, identische Lesarten über den gemeinsamen Kontext („*similar neighboring words*“) zu identifizieren und zu gruppieren (Navigli, 2009, S. 26). Hierbei ist zwischen „*Context clustering*“, „*Word clustering*“ und „*Cooccurrence graphs*“ zu unterscheiden, die insbesondere in Manning und Schütze (1999) genauer ausgeführt werden (Navigli, 2009, S. 26). Dieses Thema leitet bereits den logischen Folgeschritt ein, nämlich die tatsächliche lexikalische Disambiguierung.

Die **lexikalische Disambiguierung** (auch: *Word Sense Disambiguation, WSD*) hat zum Ziel, die Ambiguität eines einzelnen Lexems aufzulösen (s. Abschnitt 2.1.1)[26]. Hierbei gilt es herauszufinden, welche Lesart eines Lexems in einem gegebenen Kontext gemeint ist (Agirre und Edmonds, 2007, S. 1).

Dabei kann die WSD unterschiedlich aufgebaut sein und durchgeführt werden (Navigli, 2009, S. 14 ff.; Agirre und Edmonds, 2007, S. 13), wobei es grundsätzlich ein Klassifikationsproblem ist: „*Word senses are the classes, the context provides the evidence, and each occurrence of a word is assigned to one or more of its possible classes based on the evidence*“ (Agirre und Edmonds, 2007, S. 2). Die Einteilung der bestehenden Verfahren geschieht unter Hinzunahme der Arbeit von Agirre und Edmonds (2007, S. 13). Diese unterteilen bestehende WSD-Verfahren in die Kategorien: Wissensbasiert (engl. *knowledge-based*), unüberwacht korpusbasiert (engl. *unsupervised corpus-based*), überwacht korpusbasiert (engl. *supervised corpus-based*) und Kombinationen derer (engl. *combinations*)[27]. Unter wissensbasiert fallen sowohl Disambiguierungsregeln, Restriktionen und Präferenzen[28] als auch semantische Ähnlichkeitsmaße, die den Kontext eines Lexems im Abgleich mit einer Ressource berücksichtigen. Auch fallen Heuristiken in diese Kategorie (Agirre und Edmonds, 2007, S. 13). Ein Beispiel hierfür ist das Vorgehen von Gale et al. (1992), welches sich am besten mit „*one-sense-per-discourse*“ beschreiben lässt. Es basiert auf der Beobachtung, dass es sehr wahrscheinlich ist[29], dass ein Wort in einem Diskurs nur eine einzige Lesart einnimmt und wird entsprechend als zusätzliches Merkmal in WSD-Systemen berücksichtigt.

[26] Eine Übersicht über Verfahren zur WSD geben Navigli (2009) sowie Agirre und Edmonds (2007).
[27] Navigli (2009, S. 14) führen für wissensbasierte Verfahren die Begriffe „*knowledge-rich*“ und „*dictionary-based*“ an; für korpusbasierte Verfahren hingegen den Begriff „*knowledge-poor*“.
[28] Restriktionen und Präferenzen grenzen („*rule out*“) das Spektrum möglicher Lesarten durch (semantische) Beziehungen wie *EAT-FOOD* ein. Siehe dazu Agirre und Edmonds (2007, S. 119).
[29] Gale et al. (1992, S. 1) geben für hochwertige Diskurse die Tendenz zu einer Lesart mit 98% an.

Häufig finden auch **semantische Ähnlichkeitsmaße** im Rahmen der WSD Anwendung, wobei gleich mehrere Maße zur Verfügung stehen (vgl. Tabelle 3.4). Die Auswahl eines geeigneten Maßes ist dabei nicht trivial und kann zum Beispiel unter Berücksichtigung von bestehenden Gegenüberstellungen wie McCarthy et al. (2004) und Patwardhan et al. (2003) sowie insbesondere Budanitsky und Hirst (2006) erfolgen, wobei schnell deutlich wird, dass die Eignung eines Maßes als Ähnlichkeitsindikator jeweils von der angedachten Anwendung abhängt (Vöhringer und Fliedl, 2011, S. 783). In Tabelle 3.4 sind diejenigen Ansätze markiert (•), die in den Arbeiten zu semantischen Ähnlichkeitsmaßen von (BH01) Budanitsky und Hirst (2001), (MC04) McCarthy et al. (2004), (PT03) Patwardhan et al. (2003) sowie (CM05) Corley und Mihalcea (2005) die besten Evaluationsergebnisse erzielt haben.

Maß	**Originalquelle**	**BH01**	**MC04**	**PT03**	**CM05**
WUP	Wu und Palmer (1994)	–	–	–	○
HSO	Hirst und St-Onge (1995)	○	–	○	–
RES	Resnik (1995)	○	–	○	○
JCN	Jiang und Conrath (1997)	•	○	•	•
LCH	Leacock und Chodorow (1998)	○	–	○	○
LIN	Lin (1998)	○	–	○	○
aLESK	Banerjee und Pedersen (2002)	–	•	•	–

Tabelle 3.4: Maße semantischer Nähe und deren Nutzung bei der Disambiguierung
• = Gewinner; ○ = Verlierer; – = Nicht evaluiert

Im Kontext von Softwareanforderungen nutzen beispielsweise Matsuoka und Lepage (2011) das semantische Ähnlichkeitsmaß WUP (Wu und Palmer, 1994), um zwischen verschiedenen möglichen Lesarten zu unterscheiden. Endanwender können daraufhin die erkannten vermeintlich ambigen Lexeme, wenn notwendig, überarbeiten. Allerdings obliegt die Disambiguierung schlussendlich noch immer den Endanwendern. Eine Lösung kann die automatische Entscheidung für oder gegen eine Lesart über einen definierten Grenzwert sein, wie es beispielsweise Vöhringer und Fliedl (2011) im Rahmen der Erkennung und Auflösung von Terminologiekonflikten realisieren.

Weiterhin sind wissensbasierte Verfahren auf Ressourcen wie Disambiguierungsregeln[30] und Heuristiken angewiesen, die nicht für alle Sprachen und Domänen vorliegen und damit Schwachstellen darstellen. Auch überwachte korpusbasierte Verfahren weisen diese Schwachstellen auf, da sie auf manuell annotierten Korpora trainiert werden oder ein *Bootstrapping*-Verfahren nutzen (Semi-überwachtes Verfahren).

Dieser Problematik sehen sich unüberwachte korpusbasierte Verfahren nicht gegenüber (Agirre und Edmonds, 2007, S. 12), da diese vor allem Methoden, die Wörter in ihrem spezifischen Kontext *clustern* und darüber die Lesart erschließen, umfassen (Agirre und Edmonds, 2007, S. 14). „*These line of work is often referred to as word sense discrimination, as the word meanings are not disambiguated against a sense inventory, but are discriminated against each other*“ (Mihalcea, 2010, S. 1028). Einen solchen Ansatz, der sowohl automatisiert im Training als auch in der Anwendung vorgeht, stellt beispielsweise Schütze (1998) vor.

[30] Disambiguierungsregeln sind aufwändig in der Erstellung und Wartung, da sie untereinander komplexe Abhängigkeiten besitzen und umfangreiches Expertenwissen voraussetzen.

Diese bisher recht klare Kategorisierung der Verfahren liegt in der Realität nicht vor, da ebenfalls Ansätze existieren, die wissensbasierte und korpusbasierte Verfahren kombinieren (z. B. Montoyo et al., 2005). Auch ergänzen zusätzlich hybride Verfahren die bisherigen WSD-Ansätze, beispielsweise durch die Hinzunahme von EL[31]. Beide Vorgehensweisen (Kombination von WSD-Ansätzen und die Hinzunahme von EL) zielen auf mögliche Synergieeffekte ab. So können beispielsweise semantische Ähnlichkeitsmaße dazu genutzt werden, um bevorzugte Lesarten in unüberwachten korpusbasierten Verfahren zu trainieren (Agirre und Edmonds, 2007, S. 13).

Ein aktuelles und mehrsprachiges WSD- und EL-System ist **Babelfy**[32]. Dieses ist ein kombiniertes System zur Disambiguierung und zum EL in kurzen und langen Fließtexten (Moro et al., 2014b, S. 241). Der Ansatz ist wissensbasiert und nutzt das semantische Netz von BabelNet, was insbesondere die semantischen Beziehungen sowie die benannten Entitäten umfasst (Moro et al., 2014a). Nach Moro et al. (2014a, S. 25) ist es der erste Ansatz, der die Disambiguierung parallel zum EL durchführt.

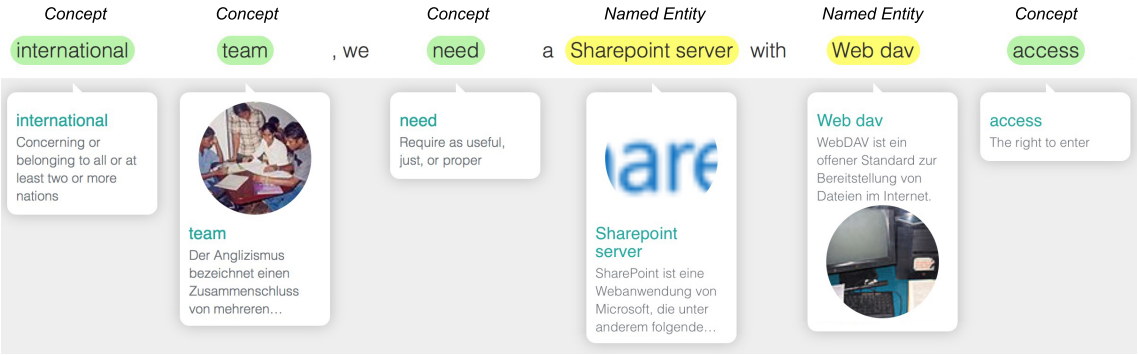


Abbildung 3.5: Disambiguierung und *Entity Linking* mittels Babelfy

Abbildung 3.5 stellt auszugsweise die Funktionalität von Babelfy am Satz „*Because we do lot of teamwork in our international team, we need a Sharepoint server with Web dav access*“ unter Verwendung der Weboberfläche dar.

Zum einen werden Konzepte wie „*team*“ und „*international*“ erkannt und annotiert, zum anderen werden benannte Entitäten erkannt und verknüpft. Dies ist insbesondere im Rahmen dieser Arbeit von Interesse, da auch Fachtermini wie „*Web dav*“ korrekt erkannt und annotiert werden. Hilfreich ist auch die Funktion, dass Babelfy Komposita wie „*Sharepoint server*“ sowohl als Komposition als auch in der jeweiligen isolierten Bedeutung erkannt und annotiert (nicht abgebildet).

Im Rahmen dieser Arbeit eignet sich Babelfy insbesondere aufgrund der sehr guten WSD-Ergebnisse (Raganato et al., 2017). Zum Beispiel bei experimenteller Anwendung auf sechs verschiedenen Goldstandards (Moro et al., 2014b). Hervorzuheben sind hierbei die guten sprachübergreifenden Ergebnisse bei der Anwendung auf kurzen sowie hochgradig ambigen Texten. Letzteres wurde durch Anwendung auf dem KORE50-Korpus[33] mit guten Ergebnissen erprobt (Moro et al., 2014b, S. 240).

[31] EL verfolgt dabei das Ziel, benannte Entitäten in Fließtexten zu erkennen und eindeutig auf eine gegebene Wissensbasis (engl. *knowledge base*) zu referenzieren (Flati und Navigli, 2014, S. 12).

[32] Siehe weiterführend: `http://babelfy.org` (Stand: 12.01.17).

[33] *Keyphrase Overlap Relatedness for Entity Disambiguation*. Siehe: `http://mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/aida` (Stand: 12.01.17).

Wie dargestellt werden konnte, wird dem Problem der lexikalischen Ambiguität im NLP vermehrt durch POS-*Tagging* und WSD-Verfahren begegnet. Liegt der Verarbeitungsfokus jedoch nicht nur auf dem einzelnen Wort, greifen diese Vorgehensweisen zu kurz. So ist die natürliche Sprache zusätzlich geprägt von umfangreicher syntaktischer Ambiguität (Jurafsky und Martin, 2009, S. 38), deren Auflösung die Betrachtung auf syntaktischer Ebene bedarf (s. Abschnitt 2.1.2).

3.3.1.3 Syntaktische Disambiguierung

Von syntaktischer Ambiguität wird gesprochen, wenn, aufgrund des Zusammenspiels umfangreicher Grammatikregeln einer Sprache, „[...] einem Ausdruck mehr als eine syntaktische Beschreibung zugeordnet werden kann“ (Ernst, 2003, S. 87). Dabei kann bereits die „Anzahl der syntaktischen Lesarten von ganz gewöhnlichen Sätzen, die von größeren *Parsing*-Systemen geliefert wird, [...] erheblich höher [sein] als der Ambiguitätsgrad, den selbst geschulte Syntaktiker auf den ersten Blick erkennen“ (Carstensen et al., 2010, S. 308). Carstensen et al. (2010, S. 308) weist in diesem Zusammenhang darauf hin, dass gültige Beispielsätze existieren, deren Satzstrukturanalyse zu Ambiguitätsgraden von einer Million und mehr führen. Es handelt sich bei syntaktischer Ambiguität demnach nicht um ein domänenspezifisches Problem der maschinellen Anforderungsverarbeitung. Vielmehr ist es eine elementare Herausforderung natürlicher Sprachen, welcher überwiegend mit Methoden der probabilistischen Satzstrukturanalyse[34] begegnet wird (Jurafsky und Martin, 2009, S. 38).

Zur automatischen Satzstrukturanalyse werden *Parser* (auch: *Natural language parser*) herangezogen (Lobin und Heringer, 2010, S. 41), die in der Lage sind, gegebenen Sätzen syntaktische Strukturen zuzuordnen. Dies umfasst die „Beschreibung des syntaktischen Baus von Sätzen durch Ermittlung elementarer Grundeinheiten wie Morphem, Wort, Satzglied und ihre Beziehung untereinander“ (Bußmann, 1983, S. 445). Diesbezüglich kann eine Dreiteilung der Disambiguierung in Eingabe, Verarbeitung und Ausgabe erfolgen:

- **Eingabe:** Natürlichsprachliche Anforderungsbeschreibung
- **Verarbeitung:** Überprüfung der Eingabe hinsichtlich gültiger Grammatik und Zuordnung passender syntaktischer Struktur(en)
- **Ausgabe:** Repräsentationen syntaktischer Strukturen (z. B. als Bäume)

Wie im Schritt der Verarbeitung deutlich wird, kann die Zuordnung mehrerer möglicher syntaktischer Strukturen erfolgen, was eher der Regelfall als der Sonderfall ist. Nach Carstensen et al. (2010, S. 303) besteht ein solcher „syntaktischer Analyseprozess zu einem nicht unwesentlichen Anteil aus Suchprozessen. Ein solcher Suchprozess lässt sich graphentheoretisch als Durchlaufen eines Suchraums (Suchgraphen) charakterisieren, der einen Startzustand Z_0 [...] und einen oder mehrere Endzustände E_1, E_2, ... , E_n [besitzt]“ (Carstensen et al., 2010, S. 303). Hierbei ist „die

[34] Der engl. Begriff des *Parsings* wird in verschiedenen wissenschaftlichen Disziplinen unterschiedlich (aber oftmals ähnlich) verwendet. Die Computerlinguistik nutzt den Begriff z. B. im Sinne der automatischen Satzstrukturanalyse, während er in der Psycholinguistik beschreibt, wie Menschen Satzstrukturen kognitiv verarbeiten (Theda, 2017, S. 12 f.; Carstensen et al., 2010, S. 303).

Verwaltung alternativer Lösungsmöglichkeiten ein zentrales Problem“ (Lobin und Heringer, 2010, S. 41). So können alle Möglichkeiten ausgegeben werden, wie ein Satz syntaktisch analysiert werden kann oder aber es wird die wahrscheinlichste Variante ausgegeben. Letzteres wird durch die probabilistischen *Parser* erreicht, die ihr Wissen aus einer Menge annotierter Sätze ableiten (s. Abschnitt 3.3.1.1) und versuchen, die wahrscheinlichste Satzstruktur in bisher unbekannten Sätzen zu analysieren[35].

Grundsätzlich ist darüber hinaus zwischen Dependenz- und Konstituentenparsern zu unterscheiden, die jeweils von unterschiedlichen syntaktischen Strukturen ausgehen. Dependenzparser analysieren die grammatikalische Struktur eines Satzes mit dem Fokus auf der Beziehung zwischen regierenden Wörtern und abhängigen Elementen. Dabei wird überwiegend davon ausgegangen, dass das Verb die Satzstruktur in Form von Leerstellen vorgibt und somit alle anderen Wörter in einem Satz vom Verb (direkt oder indirekt) über definierte Beziehungen abhängig sind (Carstensen et al., 2010, S. 282 ff.). Demgegenüber basieren Konstituentenparser auf der Idee, dass sich die natürliche Sprachsyntax mithilfe von kontextfreien Grammatiken beschreiben lässt, wobei „neben Wörtern auch komplexere Einheiten, die sogenannten Konstituenten oder Phrasen“ (Carstensen et al., 2010, S. 281), sowie Beziehungen zwischen Konstituenten, angenommen werden. Die Ergebnisse lassen sich, unabhängig von der gewählten Grammatiktheorie, als Strukturbäume darstellen (vgl. Abbildung 3.6).

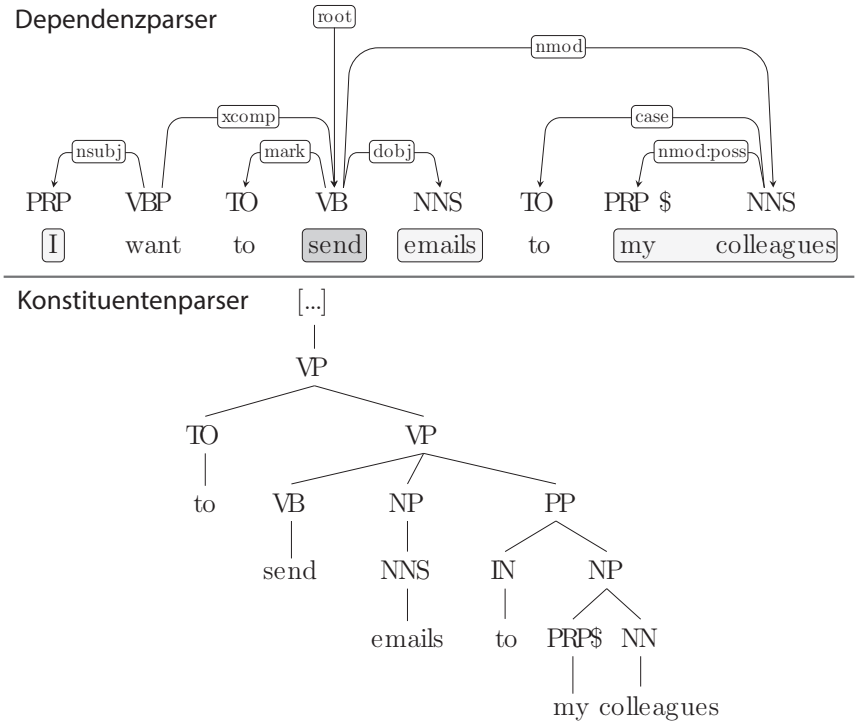


Abbildung 3.6: Gegenüberstellung verschiedenartiger Strukturbäume

Dass *Parsing* im Kontext von Softwareanforderungen einen hohen Stellenwert hat, zeigen Arbeiten wie die von Roth et al. (2014), die in ihrer Veröffentlichung „*Software Requirements: A new Domain for Semantic Parsers*“ die Verbindung zwischen seman-

[35] Siehe weiterführend: `http://nlp.stanford.edu/software/lex-parser.shtml` (Stand: 11.01.17).

tischen *Parsern* und der Domäne der Softwareanforderungen aufzeigen. Dependenzparser im Speziellen nutzen beispielsweise Drechsler et al. (2014) zur automatischen Verarbeitung natürlichsprachlicher Softwareanforderungen. Der *Stanford Parser* wird zur syntaktischen Analyse natürlichsprachlicher Anforderungen unter anderem bei Deeptimahanti und Sanyal (2009), Umber und Bajwa (2011), Friedrich et al. (2011), Bajwa et al. (2012) sowie Landhäußer et al. (2015) genutzt.

Um einen weiteren Überblick über aktuelle *Parsing*-Ansätze zu erhalten, werden im Folgenden die Ergebnisse aus Choi et al. (2015, S. 389) herangezogen, die zehn statistische Dependenzparser (Stand der Technik) in Hinblick auf Präzision und Performanz testen (vgl. Tabelle 3.5).

Parser	**Quelle**	**Präzision**	**Performanz**
ClearNLP	Choi und McCallum (2013)		2.
GN13	Goldberg und Nivre (2013)		
LTDP	Huang et al. (2012)		
Mate	Bohnet (2010)	1.	
RBG	Lei et al. (2014)	2.	
Redshift	Honnibal et al. (2013)		
spaCy[36]	–		1.
SNN	Chen und Manning (2014)		
Turbo	Martins et al. (2013)	3.	
Yara	Rasooli und Tetreault (2015)		3.

Tabelle 3.5: Überblick über aktuelle Dependenzparser.
In Anlehnung an Choi et al. (2015)

Es zeigt sich, dass keines der *Top-Parsing*-Verfahren präzise und zugleich performant ist. Choi et al. (2015) weisen in diesem Zusammenhang darauf hin, dass alle Ansätze eine Vielzahl an Konfigurationsmöglichkeiten aufweisen und hinsichtlich eines Kompromisses zwischen Performanz und Präzision optimiert werden können. Dennoch empfehlen Choi et al. (2015) die Verfahren Mate, RBG, Turbo, ClearNLP und Yara im Hinblick auf Präzision, während spaCy und ClearNLP[37] unter dem Aspekt der Performanz zu empfehlen sind. Darüber hinaus findet sich eine umfangreiche Übersicht bestehender Dependenz- und Konstituentenparser in Theda (2017).

Der Schwerpunkt in dieser Arbeit liegt hinsichtlich syntaktischer Ambiguitäten auf der Koordinationsambiguität und der PP-Anbindungsambiguität (s. Abschnitt 2.1.2). Letztere ist ein etablierter Forschungsgegenstand (Bailey et al., 2015; Agirre et al., 2008, S. 318), über den unter anderem Lapata und Keller (2005, S. 21 ff.) eine komprimierte Übersicht geben. Oftmals wird die **PP-Anbindungsambiguität** (s. Abschnitt 2.1.2) dabei als Klassifikationsproblem begriffen, in welchem es zu klären gilt, ob eine Anbindung an einer NP oder VP, bei gegebenen Kontextinformationen, erfolgen muss (Lapata und Keller, 2005, S. 21). Dabei erfordert die Auflösung von PP-Anbindungsambiguitäten die Hinzunahme von Zusatzwissen, da die resultierenden syntaktischen Strukturen jeweils gültig sind und eine Entscheidung ohne

[36] Siehe weiterführend: `https://spacy.io/` (Stand: 11.01.17).
[37] Unter Verwendung von „*greedy parsing*“.

weiteren Kontext nicht erfolgen kann: Diesbezüglich nutzt beispielsweise McLauchlan (2004) verschiedene Thesauri, um die Entscheidung zu unterstützen[38] und auch Ressourcen wie VerbNet und WordNet werden herangezogen (Bailey et al., 2015; Agirre et al., 2008), um semantische Informationen in die Entscheidungsfindung mit einfließen zu lassen und so eine syntaktische Präferenz zu generieren. Auch nutzen Nakov und Hearst (2005) online verfügbare Inhalte und speziell aus diesen abgeleitete Charakteristika (z. B. die optionale Klammerung von PP als Hinweis auf eine VP-Anbindung). Es existieren darüber hinaus mehrere Verfahren des maschinellen Lernens (z. B. Ratnaparkhi et al., 1994; Collins und Brooks, 1995; Zavrel et al., 1997; Ratnaparkhi, 1998; Pantel und Lin, 2000), die sich dieser Problemstellung annehmen und die eine Genauigkeit zwischen 81,60% und 88,10% erreichen (Bailey et al., 2015, S. 13). Zum Vergleich: Die durchschnittliche Genauigkeit menschlicher Entscheider liegt bei 88,20% (Bailey et al., 2015) bzw. 93,20% (Lapata und Keller, 2005)[39].

Der PP-Anbindungsdisambiguierung im Anforderungskontext widmen sich Bajwa et al. (2012), wobei das Ziel die automatische Überführung von natürlichsprachlichen zu formal spezifizierten Randbedingungen in der *Object Constraint Language* (OCL) ist. Zur Satzstrukturanalyse der natürlichen Sprache (Englisch) nutzen sie dabei den *Stanford Parser* sowie den *Stanford* POS-*Tagger*, wobei sie auftretende syntaktische Ambiguität (*attached ambiguity*) und Homonymie als hauptsächlich qualitätsmindernde Probleme (hinsichtlich konsistenten und validen Spezifikationen) in ihrem Fall identifizieren (Bajwa et al., 2012, S. 179). Zur Disambiguierung führen sie eine syntaktische Analyse durch, indem sie erzeugte Dependenzbäume mit UML-Klassenmodellen abgleichen[40], die zusammen mit dem Ursprungstext bereitgestellt werden. Durch dieses Vorgehen können sie die Genauigkeit des *Stanford Parsers* von 85% auf 93% (*attachment ambiguity*) und von 97% auf 99% (Homonymie) erhöhen (Bajwa et al. 2012; Shah und Jinwala 2015). Wie Shah und Jinwala (2015, S. 3) allerdings anmerken, besteht hierbei die Gefahr, dass Ambiguitäten, die nur im UML-Klassenmodell bestehen, erst durch dessen Hinzunahme als weiterer Kontext, mit in die Spezifikation aufgenommen werden.

Die **Koordinationsambiguität** ist im Vergleich zur zuvor genannten Ambiguität ein weniger populäres Themengebiet im Bereich der maschinellen Verarbeitung von Softwareanforderungen und wird vor allem durch die vielzitierte Arbeit von Berry et al. (2003) als Problem in den Fokus gerückt. Auch sonst existieren zu diesem Thema vergleichsweise wenige Arbeiten (Nakov und Hearst, 2005, S. 840; Chantree et al., 2007, S. 287) – einen Überblick geben sowohl Chantree et al. (2007, S. 288 f.) als auch Yang et al. (2010c, S. 61). Dabei kann Koordinationsambiguität überall dort auftreten, wo koordinierende Konjunktionen genutzt werden und ist insbesondere im Englischen problematisch, da Konjunktionen dort hochfrequent auftreten (Chantree et al., 2007, S. 288). Nach Chantree et al. (2007, S. 288) machen „*and*“ und „*or*“ zusammen 3% der Wörter im *British National Corpus* (BNC)[41] aus, was eine beachtliche Anzahl darstellt. Wie auch bei Yang et al. (2010c) und Chantree et al. (2007) liegt der

[38] Z. B. wird die Kookkurrenz lex. Einheiten in Form von Quadrupel (v, n_1, p, n_2) abgespeichert.

[39] In Bailey et al. (2015, S. 13) stehen den menschlichen Entscheidern nur Quadrupel zur Verfügung, während in Lapata und Keller (2005, S. 23) bei Kenntnis des gesamten Satzes getestet wird.

[40] Das Verfahren sieht vor, dass zusätzliche Kontextinformationen (ähnlich zu genannten Thesauri) aus den UML-Diagrammen extrahiert werden und eine syntaktische Präferenz ermöglichen.

[41] Siehe weiterführend: `http://www.natcorp.ox.ac.uk` (Stand: 13.01.17).

Fokus in dieser Arbeit auf den Konjunktionen „*and*“ und „*or*“. Die Erkennung von potentiell ambigen Wortkonstellationen ist über syntaktische Muster möglich. So präsentieren beispielsweise Yang et al. (2010c, S. 54) diesbezüglich eine Auswahl syntaktischer Muster, die sie „*Construction patterns used in coordination ambiguity*“ nennen. Agarwal und Boggess (1992) erkennen Koordinationen unter Hinzunahme von POS-*Tags*, ein simples aber effektives Vorgehen, das von Chantree et al. (2007, S. 290) als sinnvoller initialer Schritt in einem Gesamtsystem zur Disambiguierung genannt wird. Chantree et al. (2005) testen darüber hinaus in ihrer Arbeit die Hypothese, dass die wahrscheinlichste Lesart einer Koordination über die Wortverteilung[42] in einem generischen Korpus gefunden werden kann (Chantree et al., 2005; Chantree et al., 2007). Dieses Vorgehen erinnert an die Arbeit von Resnik (1999), der die semantische Ähnlichkeit zwischen Wörtern zur Disambiguierung hinzuzieht und welche Chantree et al. (2005) auch als Vergleichswert heranziehen. Darüber hinaus existieren Ansätze, die Koordinationsambiguität aufzulösen versuchen, indem sie linguistische Merkmale (z. B. Groß- und Kleinschreibung, Kommata) der Konjunktion heranziehen (Okumura und Muraki, 1994) oder indem Sie die Kookkurrenz, also das gemeinsame Auftreten von lexikalischen Einheiten (Modifikatoren und verbundene Wörter), berücksichtigen und Regeln und Muster auf Grundlage verschiedener linguistischer Ressourcen ableiten (Yang et al., 2010c, S. 61). Beispielsweise werden online verfügbare Inhalte und speziell aus diesen abgeleitete Charakteristika herangezogen (z. B. Nakov und Hearst, 2005, S. 839 ff.). Goldberg (1999) wiederum nutzt das WSJ, um die Disambiguierung in den Fällen der syntaktischen Struktur „N_1 P N_2 CC N_3“ wie zum Beispiel in „*collection of files and documents*“ zu ermöglichen.

Speziell auf Anforderungsbeschreibungen gehen auch die Arbeiten von Chantree et al. (2006) sowie Yang et al. ein (Yang et al., 2010a; Yang et al., 2010b; Yang et al., 2010c), wobei nicht die Disambiguierung im Mittelpunkt steht, sondern die Gefahr der Fehlinterpretation, die von einer spezifischen Koordinationsambiguität ausgeht (Yang et al., 2010b, S. 1218). Diesem Ansatz liegt die Annahme zu Grunde, dass die meisten Ambiguitäten (zumindest von Menschen) nicht falsch interpretiert werden und daher nicht schädlich sind (Yang et al., 2010b, S. 1218). Dies ist im Rahmen der vorliegenden Arbeit interessant, da eine Beachtung nur schädlicher Ambiguitäten zu einer Minimierung der Laufzeit führen würde – allerdings müsste sichergestellt werden, dass die Ambiguitäten auch in maschineller Verarbeitung unschädlich sind.

3.3.1.4 Automatische Koreferenzauflösung

Koreferenzauflösung ist auch in der Verarbeitung natürlichsprachlicher Anforderungen von erheblicher Bedeutung. Körner (2014) merkt allerdings an, dass viele Verfahren sich „[...] hauptsächlich mit der Auflösung von Personalpronomina und den dazugehörigen Bezeichnern/Namen“ (Körner, 2014, S. 174) beschäftigen und im Softwareanforderungskontext unerprobt sind.

Um referentielle Ambiguität (s. Abschnitt 2.1.3) erkennen und kompensieren zu können, müssen zuerst Referenten und zugehörige (sowie möglicherweise zugehörige) Referenzausdrücke erkannt werden. Es existiert eine Reihe von Verfahren zur Ko-

[42] Betrachtet wird, ob lexikalische Köpfe vermehrt mit ihrem Modifikator oder mit dem, durch Koordination verknüpften, zweiten Kopf als syntaktische Einheit auftreten.

referenzauflösung, die bereits zur Erfüllung dieser Aufgabe mit Komponenten zur Auflösung von referentieller Ambiguität ausgestattet sind.

Ziel dieser Verfahren ist es, Koreferenzketten zu bilden, was bedeutet, koreferente Referenzausdrücke zusammen darzustellen (Stede, 2012, S. 50). Eine Sonderform sind Verfahren, die auch *Singletons* erkennen – also Referenten, die nur einmalig im Text vorkommen (z. B. Recasens et al., 2013). Einen Überblick über die Thematik der Koreferenzresolution geben unter anderem Stoyanov et al. (2009) sowie Mitkov (1999). Ambiguität nimmt dabei bei Mitkov (1999) einen hohen Stellenwert ein.

Ansätze der Anaphernresolution können in „Linguistische Ansätze“, „Heuristiken“ und „Maschinelles Lernen“ unterteilt werden (Carstensen et al., 2010, S. 400 ff.). An anderer Stelle findet sich eine Aufteilung von Verfahren der Koreferenzanalyse in „Wissensbasierte Ansätze“ und „Maschinelle Lernverfahren“ (Geierhos, 2010, S. 94). Unter anderem Prokofyev et al. (2015, S. 463) wiederrum unterteilen Verfahren hinsichtlich der eingebundenen Ressourcen in „wissensreich“ (engl. *knowledge-rich*) und „wissensarm“ (engl. *knowledge-lean*).

Wissensarme Verfahren, die in vielen Fällen nur auf wenigen Regeln, ausgewählten *Features* sowie morphologischen als auch syntaktischen Informationen beruhen (z. B. Lee et al., 2011; Bengtson und Roth, 2008; Mitkov, 1998), erreichen teils sehr gute Evaluationswerte (Harabagiu et al., 2001, S. 1). An dieser Stelle ist beispielsweise das *Centering*-Modell (Grosz et al., 1995) als früher linguistischer Ansatz der Anaphernresolution zu nennen. Carstensen et al. (2010, S. 408) bezeichnen den Ansatz von Lappin und Leass (1994) als wichtig im Bereich der Heuristiken.

Bei der Anwendung **wissensreicher Verfahren** stehen die Ressourcen im Mittelpunkt, wobei die herangezogenen Wissensquellen dabei vielfältiger Natur sind. Vielfache Verwendung finden WordNet (Huang et al., 2009; Ponzetto und Strube, 2006; Markert und Nissim, 2005; Harabagiu et al., 2001) und umfangreiche Korpora (Haghighi und Klein, 2009; Yang und Su, 2007; Markert und Nissim, 2005). Die Akquise von Wissen ist aber nach wie vor zeitintensiv, aufwändig sowie fehleranfällig und damit der sprichwörtliche Flaschenhals vieler Verfahren (Uryupina et al., 2012, S. 185; Harabagiu et al., 2001, S. 1).

Webressourcen wie die Wikipedia sind aufgrund ihres Umfangs (Haghighi und Klein, 2009; Yang und Su, 2007) und auch wegen ihrer internen Verknüpfung beliebt (Kobdani et al., 2011; Bryl et al., 2010). So nutzen beispielsweise Strube und Ponzetto (2006) Wikipedia zur Berechnung der semantischen Ähnlichkeit von Referenzausdrücken als Erweiterung (zusätzliches *Feature*) ihres ML-Verfahrens zur Koreferenzauflösung (Ponzetto und Strube, 2006).

Über einzelne Ressourcen hinaus bietet das *Semantic Web* eine bisher nicht dagewesene Menge an semantisch angereicherten Datenbeständen und ist aufgrund dieses Umfangs sowie der Struktur prädestiniert für die Anwendung in NLP-Applikationen, so auch für die Koreferenzauflösung. Bryl et al. (2010, S. 759) weisen jedoch darauf hin, dass die Erweiterung bestehender Verfahren der Koreferenzauflösung um diese Ressourcen bei weitem kein trivialer Akt ist. Insbesondere benennen sie die Heterogenität und die Ambiguität der verschiedenen Ressourcen im *Semantic Web* als problematisch. Darüber hinaus besteht ein Problem in der Wissensverteilung – zu manchen Themen existiert viel, zu anderen Themen wenig Wissen (Bryl et al., 2010, S. 759). Somit kann eine ausreichende Abdeckung nicht garantiert werden.

Diesbezüglich untersuchen Uryupina et al. (2012) die zwei etablierten Webressourcen Wikipedia und YAGO hinsichtlich einer möglichen Verbesserung in der Leistung von Systemen zur Koreferenzauflösung. Hierzu extrahieren sie semantische Informationen und integrieren diese in das *Beautiful Anaphora Resolution Toolkit* (BART). Dabei stellen sie fest, dass eine Verbesserung nur erreicht wird, wenn Maßnahmen zur erweiterten Disambiguierung und Filterung der Ressourcen herangezogen werden. Die daraufhin ausgearbeiteten Lösungen zur Reduzierung des Rauschens in den Daten basieren auf der Anwendung von Disambiguationswerkzeugen und *Pruning*, um stark generische Informationen zu entfernen. Auch Rahman und Ng (2011) können durch Hinzunahme von Weltwissen (YAGO) die Auflösung von Koreferenzen verbessern. Darüber hinaus beziehen sie auch FrameNet mit ein. Ressourcen wie FrameNet und VerbNet werden oftmals aufgrund ihrer Strukturiertheit und semantischen Annotationen herangezogen. Darüber hinaus nutzen Ng (2007) zur Akquise von Wissen beispielsweise die *Penn Treebank*, die semantisch annotierte Nominalphrasen enthält. Das Problem des begrenzten Ressourcenumfangs bleibt aber bestehen.

In Tabelle 3.6 sind Verfahren der automatischen Koreferenzresolution, zusammen mit einer Aufteilung in „regelbasiert“ (Reg.) und „datenbasiert“ (Dat.), aufgelistet, wobei bereits eine Vorauswahl getroffen wurde: Für diese Arbeit sind nur solche Verfahren relevant, für die eine Implementierung vorliegt bzw. Zugriff auf alle notwendigen Ressourcen gegeben ist und bei denen es sich nicht nur um primäre Experimentierumgebungen handelt. Weiterhin enthält Tabelle 3.6 Angaben darüber, ob sich die Verfahren in aktiver Weiterentwicklung befinden (•).

Als Beispiel ist BART anzuführen, dass ein modulares Koreferenzresolutionssystem darstellt (Carstensen et al., 2010; Versley et al., 2008) und dabei verschiedene statistische Ansätze sowie ein einfaches Anpassen der *Features* unterstützt. Allerdings handelt es sich dabei primär um eine Entwicklungsumgebung und wurde daher unter einen anderem Schwerpunkt entwickelt (Versley et al., 2008, S. 9, 11), als es für diese Arbeit erforderlich ist (z. B. Geschwindigkeit).

Nr.	Ansatz	Quelle	Aktiv	Reg.	Dat.
1	Illinois	Bengtson und Roth (2008)	•	○	•
2	CherryPicker	Rahman und Ng (2009)	○	○	•
3	Reconcile	Stoyanov et al. (2010)	–	○	•
4	ARKref	O'Connor und Heilman (2013)	○	•	○
5	dcoref	Lee et al. (2013)	•	•	•
6	Berkeley	Durrett und Klein (2013)	–	○	•
7	HOTCoref	Björkelund und Kuhn (2014)	○	○	•

Tabelle 3.6: Ansätze zur automatischen Koreferenzauflösung

Im Folgenden werden zwei aktive Verfahren detaillierter vorgestellt. Das ***dcoref*** (5) ist Bestandteil des *Stanford CoreNLP Natural Language Processing Toolkits* (Manning et al., 2014). Es ermöglicht die Erkennung von Referenzausdrücken und die Auflösung von pronominaler sowie nominaler Koreferenz (Lee et al., 2013) und befindet sich in aktiver Entwicklung[43]. Dabei werden bewährte Vorgehensweisen deterministischer,

[43]Siehe weiterführend: `http://www-nlp.stanford.edu/software/dcoref.shtml` (Stand: 11.01.17).

regelbasierter Systeme (Transparenz und Modularität) sowie des ML (weitreichende Informationen und präzise *Features*) kombiniert und sequenziell angewendet (Lee et al., 2013, S. 887 f.). Als eine Ressource wird dabei initial Wikipedia herangezogen (Lee et al., 2013, S. 895). Die sogenannte Siebarchitektur (engl. *sieve architecture*) basiert darauf, verschiedene Koreferenzmodelle sequenziell, geordnet nach Präzision, anzuwenden, wobei die Anwendung stets auf der Ausgabe der vorherigen Komponente beruht. Vielfache Anwendung erfährt *dcoref* aufgrund des modularen Aufbaus und der herausragenden Performanz. Dabei ist *dcoref* nicht gänzlich ohne Anpassungen an die jeweilige Sprache (z. B. Zhang et al., 2012) und Domäne anzuwenden, was Passonneau et al. (2015, S. 243, 245 ff.) exemplarisch anhand der Domäne „Finanznachrichten" aufzeigen. Allerdings kommt das *Toolkit* den Entwicklern hier durch eine hohe Konfigurierbarkeit entgegen[44].

Als datenbasiertes Verfahren steht das ***Illinois Coreference Package*** (1) zur Verfügung, welches von der *University of Illinois at Urbana-Champaign* entwickelt wird. Es umfasst ein *Tool* zur Resolution von Koreferenzen sowie eine Reihe von dazugehörigen NLP-*Features* (z. B. WordNet Relationen, semantische Klassen), angelehnt an Culotta et al. (2007). Nach Bengtson und Roth (2008, S. 6) liegt der Schwerpunkt auf der englischen Sprache – weitere Sprachimplementierungen sind nicht bekannt. Der modulare Aufbau des Verfahrens erlaubt eine Evaluation der einzelnen *Features* hinsichtlich des Beitrags zur Koreferenzauflösung (Bengtson und Roth, 2008, S. 4, 8).

Zur **Evaluation** werden spezielle Korpora verwendet (s. Abschnitt 3.3.1.1 sowie Recasens Potau, 2010, S. 10). Tabelle 3.7 zeigt quantitative Evaluationsmaße der angeführten Ansätze – soweit vorhanden. Gezeigt wird das harmonisierte F_1-Maß[45]. Die Ansätze basieren in der Regel auf englischer Sprache, jedoch sind weitere Faktoren von Relevanz, sodass ein direkter Vergleich schwer fällt bzw. nur einen Eindruck der Erkennungsqualität geben kann. Von Bedeutung ist unter anderem auch die jeweilige Konfiguration, wodurch die folgenden Angaben nur eine Übersicht geben können. Weiterhin weist Recasens Potau (2010) auf die generelle Problematik der Vergleichbarkeit hin: „*The lack of a reliable metric, the use of different corpora (and of different portions of the same corpus) and the reliance on true or system mention boundaries [...] make any comparison between different systems meaningless*" (Recasens Potau, 2010, S. 19).

Zwar existiert augenscheinlich eine Vielzahl an Verfahren zur automatischen Koreferenzresolution, allesamt stehen aber dem Problem der Ambiguität gegenüber (z. B. Raghunathan et al., 2010, S. 500), wie Poesio und Artstein (2005, S. 76) am Beispiel der Anaphernresolution aufzeigen. Dabei zeigt sich Baldwin (1997, S. 39) wenig überrascht davon, dass kurze Texte tendenziös eher ambig sind als Texte, die voll umfänglich verfasst wurden.

[44] Siehe weiterführend: `http://nlp.stanford.edu/software/dcoref.shtml` (Stand: 11.01.17).
[45] CEAF wird unterteilt in *Mention* (M) und *Entity*-basiert (E).
[46] ACE04, vgl. Bengtson und Roth (2008, S. 300)
[47] ACE05, vgl. Rahman und Ng (2009, S. 976)
[48] ACE05, vgl. Reconcile Development Team (2011)
[49] ACE04-Roth-Dev, vgl. O'Connor und Heilman (2013, S. 7)
[50] CoNLL11-Dev, vgl. Stanford NLP Group (2016)
[51] CoNLL12-Dev, vgl. Stanford NLP Group (2016)
[52] CoNLL12, vgl. Berkeley NLP Group (2016)

Nr.	Ansatz	Evaluation		
		MUC	B^3	CEAF
1	Illinois[46]	75,8	80,8	–
2	CherryPicker[47]	69,3	61,4	(M) 59,5
3	Reconcile[48]	59,9	69,1	–
4	ARKref[49]	–	80,5	–
5	dcoref[50]	60,7	52,1	(E) 50,1
	dcoref[51]	65,0	54,5	(E) 56,1
6	Berkeley[52]	70,6	58,2	(E) 54,8
7	HOTCoref[53]	70,7	58,6	(E) 55,6

Tabelle 3.7: Ansätze zur Koreferenzauflösung (F_1-Maß)

Verfahren der Koreferenzresolution begegnen dieser Herausforderung durch ausgewählte disambiguierende Faktoren (Regeln, *Features*). Lee et al. (2011, S. 28) weisen dabei auf die Notwendigkeit hochgradig präziser lexikalischer und syntaktischer *Features* hin. Allerdings können auch mit präzisen *Features* nicht alle Ambiguitäten ohne semantisches Zusatzwissen aufgelöst werden. Herausgestellt werden sollen an dieser Stelle Verfahren, die als Nachbearbeitung auf Koreferenzketten angewendet werden, um Ambiguität aufzulösen. Prokofyev et al. (2015) entwickeln mit SANAPHOR ein System, dass auf das semantische Web zurückgreift und annotierte Referenzausdrücke durch zusätzliches Wissen (z. B. Wikipedia) zu disambiguieren versucht. Bestehende Koreferenzketten werden wenn notwendig modifiziert[54]. Auch Bansal und Klein (2012) nutzen zur Disambiguierung *Web features*, die auf *Reconcile* angewendet werden (Stoyanov et al., 2010).

Der Vollständigkeit halber ist noch auf Verfahren zu verweisen, die Ambiguität erkennen aber nicht auflösen. Yang et al. (2011) präsentieren beispielsweise ein Klassifikationsverfahren, dass potentiell schädliche anaphorische Ambiguitäten (engl. *potentially nocuous ambiguities*), das bedeutet Ambiguitäten, die sehr wahrscheinlich fehlinterpretiert werden, erkennt und Endanwender über deren Existenz informiert (Yang et al., 2010a; Yang et al., 2010b; Yang et al., 2010c). Es wird demnach nicht versucht, die Ambiguität durch die Wahl der am wahrscheinlichsten Disambiguation zwangsläufig aufzulösen (Yang et al., 2011, S. 186). Dies ist aber im Rahmen der Zielsetzung einer weitestgehenden Automatisierung der Anforderungskompensation erforderlich.

3.3.2 Reduktion von Unvollständigkeit

Vollständigkeit wird in der Literatur oftmals als eine elementare Qualitätseigenschaft von Anforderungen und in einem Atemzug mit Konsistenz, Eindeutigkeit und Verifizierbarkeit genannt (Grande, 2011, S. 83 f.; Fabbrini et al., 2000, S. 3 f.; IEEE, 1998, S. 4). Dabei bleibt unklar, was unter dem Begriff der Vollständigkeit (engl. *completeness*) verstanden wird, welcher Bezugspunkt gewählt wird (z. B. eine Anforderung oder eine Dokumentation von Anforderungen) und wann der Zustand

[53] CoNLL12, vgl. Björkelund und Kuhn (2014, S. 53 ff.)
[54] Demonstriert wird das Verfahren an *Stanford dcoref*.

der Vollständigkeit erreicht ist (Firesmith, 2005, S. 27; Ferrari et al., 2014, S. 25 f.). Firesmith (2005, S. 41) und Davis et al. (1993, S. 145) stellen diesbezüglich in Frage, ob dieser Zustand generell erreicht werden kann. Nach Pekar et al. (2014, S. 243) ist Unvollständigkeit nach Ambiguität das in der Literatur am meisten identifizierte Problem im Kontext von Anforderungstexten.

Oftmals beziehen sich vollständige Anforderungen (engl. *complete requirements*) oder die Vollständigkeit von Anforderungen (engl. *completeness of requirements*) auf das gänzliche Fehlen von Anforderungen innerhalb einer Anforderungsdokumentation. In dieser Arbeit liegt der Fokus auf vorhandenen, aber unvollständigen Anforderungen, die beispielsweise von Firesmith (2005, S. 36 ff.) als „*complete individual requirements*“ bezeichnet werden. Allerdings ist dieser Begriff noch weiter zu erläutern, da Vollständigkeit in Abhängigkeit der Art von Anforderung (s. Abschnitt 1.2) unterschiedlich ausgeprägt sein kann. Eine NFA kann vollständig sein, obwohl Informationen fehlen, die eine FA voraussetzen würde (Firesmith, 2005, S. 36 ff.). So kann der Satz „Die Sicherheit wird durch Verschlüsselung aller ausgehender E-Mails gewährleistet“ für eine NFA hinreichend sein, während er für eine FA nicht geeignet ist, da Details zur funktionalen Umsetzung (z. B. Verschlüsselung durch PGP) fehlen.

Weitgehende Einigkeit besteht in den Auswirkungen, die unvollständige Anforderungen für ein Softwareprodukt (Ghazarian, 2009), die Produktsicherheit (HSE, 2003) oder ein gesamtes Projekt (Kamata und Tamai 2007; Standish Group International 1995; Bell und Thayer 1976) haben können. Im Kontext dieser Arbeit ist auch auf die negativen Auswirkungen auf die Kundenakzeptanz und -zufriedenheit hinzuweisen (Firesmith, 2005, S. 28; Davis et al., 1993, S. 142). Darüber hinaus kann Unvollständigkeit weitere Formen der Ungenauigkeit, beispielsweise Ambiguität, begünstigen oder sogar verursachen (s. Abschnitt 2.1.2). Das Phänomen der Unvollständigkeit ist demnach nicht als isoliertes Phänomen zu betrachten.

3.3.2.1 Identifikation unvollständiger Anforderungen

Der Identifikation unvollständiger Anforderungen widmen sich sowohl Praxis als auch Wissenschaft seit mehreren Jahrzehnten (z. B. Fagan, 1976). Bereits Boehm (1984, S. 86) führt eine Vielzahl an Verifikations- und Validierungstechniken (z. B. Lesetechniken, Checklisten und Interviews aber auch mathematische Beweise und Modelle) auf, die unter anderem das Unvollständigkeitsphänomen adressieren. Im Fokus der Literatur stehen dabei generelle Überlegungen zur Identifikation sowie Möglichkeiten der (softwaretechnischen) Unterstützung (z. B. Decker et al., 2007). Im Folgenden werden manuelle sowie softwareunterstützte Identifikationverfahren angeführt. Manuelle Verfahren dienen in dieser Arbeit allerdings primär der thematischen Abdeckung. Einen umfassenden Überblick geben beispielsweise Aurum et al. (2002) sowie Laitenberger und DeBaud (2000).

Manuelle Verfahren

Eine naheliegende Möglichkeit, Unvollständigkeit zu erkennen, ist ein Abgleich mit den eigenen, individuellen Erfahrungswerten während des Lesens. Dieses Vorgehen wird ***ad hoc review*** genannt und ist für Anforderungsbeschreibungen in natürlicher Sprache geeignet (Shull et al., 2000, S. 75). Es ist aber weder systematisch im Aufbau,

noch fokussiert und nur schwer nachzuvollziehen, geschweige denn an neue Projektumstände anzupassen (Shull et al., 2000, S. 74). Weitere Ansätze zur Identifikation (und Kompensation) von Unvollständigkeit wie von Yadav et al. (1988) basieren ebenfalls oftmals auf der Begutachtung durch Dritte und sind daher geprägt von Subjektivität, eingeschränkten Perspektiven bzw. individuellen Schwerpunkten sowie einer inkonsistenten Bewertung (Menzel et al. 2010, S. 15; España et al. 2009, S. 1). Auch weil die Wahrnehmung von Vollständigkeit aufgrund von expliziten und impliziten Annahmen stark variieren kann (Albayrak et al., 2009). Eine standardisierte Vollständigkeitsprüfung, wie von Firesmith (2005, S. 39) vorgeschlagen, kann diese Probleme minimieren aber nicht gänzlich ausschließen.

Firesmith (2005, S. 39) empfiehlt daher im Umgang mit „*individual requirements*“ unter anderem die Erstellung von **Checklisten** für Anforderungsarten und deren Komponenten sowie die Nutzung von projektspezifischen „*requirement completeness guidelines and/or standards*“ (Firesmith, 2005, S. 39). Checklisten sind dabei als Vorgehen unter anderem aufgrund einer ausgeprägteren Struktur und besseren Anpassungsmöglichkeiten an neue Projektumstände den *ad hoc reviews* vorzuziehen (Shull et al., 2000, S. 74). Darüber hinaus existieren systematische **Lesetechniken** wie das „*Perspective-Based Reading*“ (Shull et al., 2003) oder das „*Defect-Based Reading*“, die in Form von Schritt-für-Schritt-Verfahren auftreten (Shull et al., 2001). Dabei werden mehreren Lesern unterschiedliche *Stakeholder*-Perspektiven bzw. mögliche Softwaredefekte mit konkreten Fragestellungen (z. B. zur Vollständigkeit) zugeteilt, um eine möglichst große Spezifiziertheit und Vollständigkeit zu erreichen.

Aurum et al. (2002, S. 146 ff.) zeigen dabei auch softwaretechnische Unterstützungsmöglichkeiten auf, die den Softwareinspektionsprozess effizienter gestalten sollen. Daran anknüpfend werden im Folgenden softwareunterstützte Ansätze zur Identifikation von Unvollständigkeit herangezogen.

Softwareunterstützte Verfahren

Softwareunterstützte Verfahren identifizieren Unvollständigkeit zum einen durch den Abgleich einer Anforderungsbeschreibung mit weiteren Ressourcen, beispielsweise Qualitätsmodellen oder Ontologien. Zum anderen gibt es Verfahren, die unter anderem domänenspezifische Wörterbücher als Unvollständigkeitsindikatoren heranziehen.

Huertas und Juárez-Ramírez (2012) erkennen unvollständige Anforderungen beispielsweise über einen Abgleich mit vorgegebenen „W-Fragen“[55]. So gilt eine Anforderung (in diesem Fall ein Satz) als vollständig, wenn *Actor* („*Who*“), *Function* („*What*“) und *Detail* („*Where / When*“) angegeben sind.

Systematischer gehen Fabbrini et al. (2001) vor, die mit QuARS (*Quality Analyzer of Requirement Specification*) eine Anwendung bereitstellen, die auf Grundlage eines Qualitätsmodells die Überarbeitung von Anforderungsbeschreibungen durch einen *Stakeholder* ermöglicht. Im Rahmen der *Specification completion* wird auf Satzbasis nach unvollständig spezifizierten Subjekten gesucht (z. B. „*flow*“) die einer weiteren Spezifizierung bedürfen (z. B. „*data flow*“). Ähnlich gehen Fantechi und Spinicci (2005) vor, die mit dem *Java Requirement Analyzer* einen Ansatz vorstellen, der

[55] Offene „W-Fragen“ sind Fragen, die mit einem W-Wort beginnen (z. B. „Wann“).

die Satzstruktur analysiert und mit speziellen Wörterbücher abgleicht, um fehlende Bestandteile eines *Subject-Action-Object*-Triples zu identifizieren.

Unvollständige Spezifizierung greifen auch Körner (2014) sowie Körner und Brumm (2010) zur Verbesserung natürlichsprachlicher Anforderungen auf: Unvollständig spezifizierte Prozesswörter (engl. *incompletely specified process words*) und unvollständig spezifizierte Bedingungen (engl. *incompletely specified conditions*). Körner und Brumm (2010) stellen dabei eine Anwendung namens *Requirements Engineering Specification Improver* (RESI) bereit, die sprachliche Mängel in Anforderungstexten aufzeigen und im Dialog mit einem *Stakeholder* kompensieren kann. RESI bezieht die hierbei notwendigen Informationen (z. B. Prädikate und Leerstellen) aus Ressourcen wie ResearchCyc[56] und semantischen Wortdatenbanken wie WordNet (Miller, 1995). Vergleichbar ist der Ansatz von Landhäußer et al. (2015), der unvollständige Nominalisierungen erkennt und Nutzern zur Korrektur anzeigt.

Geierhos et al. (2015) diskutieren eine Vorgehensweise, die über den Abgleich mit einer Ontologie hinausgeht und auf domänenspezifischer Ähnlichkeitssuche (engl. *similarity retrieval*) basiert. Ziel ist es, unbeschränkte Anforderungsbeschreibungen zu ermöglichen und Endanwender mittels Textvorschlägen zu unterstützen (Geierhos et al., 2015, S. 277). Dabei werden natürlichsprachliche Anforderungsbeschreibungen, die in Form und Inhalt einmalig sind (UGC), auf ihre semantischen Hauptkomponenten reduziert, indiziert und iterativ als Vorlagen verwendet. Hierbei werden domänenspezifischen Ontologien herangezogen, um fehlende Informationen im Eingabetext zuverlässig durch Informationen aus den Vorlagen kompensieren zu können (Geierhos und Bäumer, 2016).

Ontologien werden auch andernorts zur Erkennung von Unvollständigkeit eingesetzt (Bhat et al. 2014; Kaiya und Saeki 2006; Kaiya und Saeki 2005). So präsentieren Verma und Kass (2008) mit RAT (*Requirements Analysis Tool*) eine Anwendung, die automatisch eine Vielzahl von syntaktischen und semantischen Analysen auf natürlichsprachlichen Anforderungsdokumenten anwendet und Texte basierend auf „*Best Practices* der Branche“ prüft. Dabei werden sowohl unvollständige Anforderungen (z. B. offene Leerstellen eines Prädikats) als auch fehlende Anforderungen identifiziert (Verma und Kass, 2008, S. 753). Als Ressourcen werden benutzerspezifische Glossare, kontrollierte Syntax und domänenspezifische Ontologien einbezogen.

Es müssen allerdings nicht zwangsläufig klassische Lexika oder Ontologien als Ressourcen dienen. Ferrari et al. (2014) ermöglichen mit ihrem *Completeness Assistant for Requirements* (CAR) die Messung von Vollständigkeit durch Hinzunahme von Dokumenten, die während der Aufnahme von Anforderungen anfallen. Vollständigkeit zeichnet sich dabei dadurch aus, dass alle in den Dokumenten genannten Konzepte und Abhängigkeiten auch in den formulierten Anforderungen wiederzufinden sind. Interessant an diesem Ansatz ist das Vorgehen, Ausdrücke und Verbindungen zwischen Ausdrücken aus den Dokumenten zu extrahieren und als Abgleich zu nutzen.

3.3.2.2 Kompensation unvollständiger Anforderungen

Es existieren nur wenige Arbeiten zur Kompensation unvollständiger natürlichsprachlicher Anforderungen. Verfahren, die natürlichsprachliche Anforde-

[56] Siehe weiterführend: `http://www.cyc.com/platform/researchcyc/` (Stand: 11.01.17).

rungen unterstützen, beschränken die Anwendung dabei oftmals auf kontrollierte Sprachen (z. B. Holtmann et al., 2011). Dennoch existieren, wie bereits zuvor im Rahmen der Identifikationsverfahren dargestellt, Verfahren wie RESI von Körner und Brumm (2010), die Vollständigkeit im Sinne vollständig spezifizierter Prozesswörter behandeln. Einen solchen prädikatbasierten Ansatz präsentieren Bäumer und Geierhos (2016). Die Identifikation unvollständiger Prädikate basiert dabei auf bestehenden Techniken im Kontext von *Semantic Role Labeling* (SRL), die sowohl Prädikate als auch Argumente erkennen können. Informationen über die spezifische Prädikat-Argument-Struktur (PAS) sind dabei Ressourcen wie *Propbank* (Palmer et al., 2005) und *FrameNet* (Baker et al., 1998) zu entnehmen, wobei in diesem Fall *Propbank* genutzt wird. Die Kompensation erfolgt über ein speziell angepasstes IR-Modul, dessen Index mit bereits vorverarbeiteten Anforderungsbeschreibungen bzw. *User Stories* gespeist wurde. Dies ermöglicht kontextspezifische Suchanfragen (auch: Kompensationsanfragen), die eine geeignete Instanz für eine fehlende Instantiierung im jeweiligen Kontext zurückgeben können.

Eine Weiterentwicklung stellt der Ansatz von Geierhos und Bäumer (2016) dar, der ebenfalls auf eine prädikatbasierte Kompensation zurückgreift. Zusätzlich wird die Kompensation durch domänenspezifisches Wissen verfeinert (mithilfe entsprechender Ontologien) sowie erweitert („*Concept Expansion*“). Wird beispielsweise eine geeignete Instanz für ein fehlendes Argument gefunden (z. B. „*Friends*“), prüft das erweiterte Verfahren, ob ähnliche Instanzen der gleichen semantischen Kategorie vorliegen und schlägt diese ebenfalls vor (z. B. „*Colleagues*“ und „*Family*“).

Diese Verfahren unterliegen dabei zwei nennenswerten Limitationen: (1) der nur begrenzt verfügbaren domänenspezifischen Ressourcen sowie der Performanz existierender *Semantic Role Labeler*. Erstgenanntes ist bereits in Abschnitt 3.2 als Problem diskutiert worden. Letzteres ist Gegenstand aktueller Forschung, so zum Beispiel bei Schenk und Chiarcos (2016) sowie Laparra und Rigau (2013).

Ferrari et al. (2014) schlagen, ähnlich zu Geierhos und Bäumer (2016), ein auf Textvorschlägen basiertes Verfahren vor, was Unvollständigkeit feststellt und interaktiv behebt. Den Aspekt fehlender Ressourcen kompensieren sie, indem bestehende Dokumente herangezogen werden, die während der Aufnahme von Anforderungen anfallen, beispielsweise bei Planungstreffen. CAR extrahiert relevante Terme und Termbeziehungen aus den Dokumenten und untersucht eine gegebene Anforderungsbeschreibung hinsichtlich vorkommender Terme und der Übereinstimmung. Abbildung 3.7 zeigt diesbezüglich die CAR-Programmoberfläche, welche sowohl die Anforderungen (erste Textbox) als auch die zusätzlichen Dokumente als Freitext (zweite Textbox) beinhaltet (Ferrari et al., 2014, S. 31 ff.).

Stetig wird die Vollständigkeit der Anforderungen berechnet (vgl. Ferrari et al., 2014, S. 31) und es werden dem Benutzer Terme vorgeschlagen, auf dessen Grundlage eine weitere Anforderung zu schreiben und der Anforderungssammlung hinzuzufügen ist. Wird eine neue Anforderung hinzugefügt, wird die Vollständigkeit erneut berechnet. Wie in Abbildung 3.7 ersichtlich ist, werden kontrollierte natürlichsprachliche Anforderungsbeschreibungen vorausgesetzt.

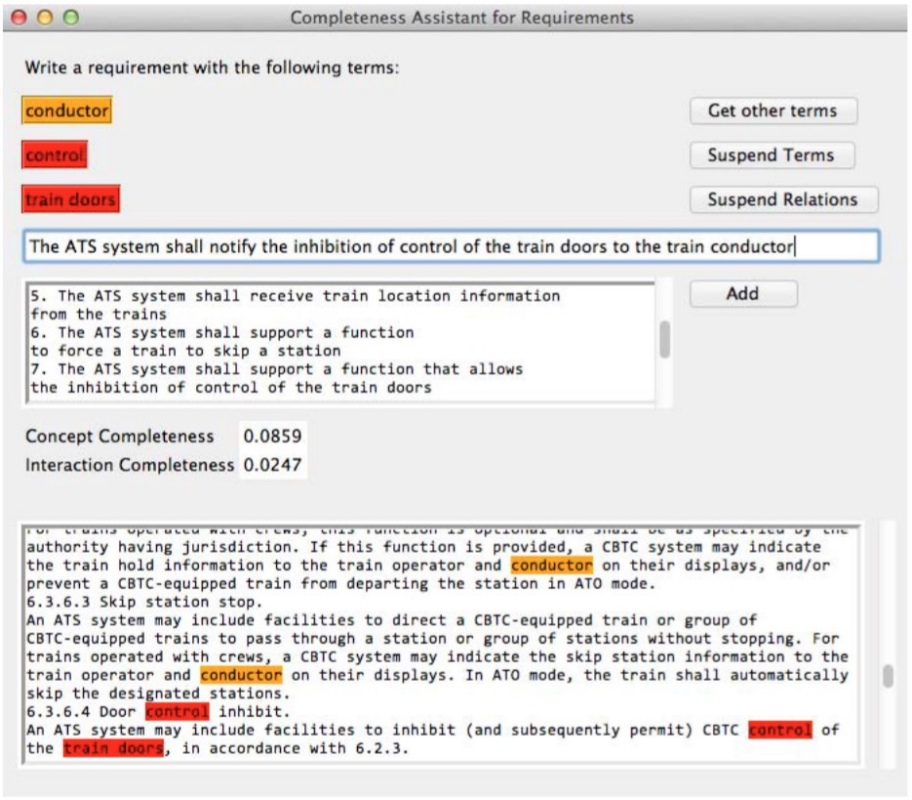


Abbildung 3.7: *Completeness Assistant for Requirements.*
Entnommen aus Ferrari et al. (2014, S. 32)

3.3.3 Kombinierte Ansätze

Die Idee, natürlichsprachliche Anforderungen nicht isoliert auf eine Form der Ungenauigkeit oder Unvollständigkeit zu prüfen, sondern mehrere Verfahren der Erkennung und/oder Kompensation zu kombinieren, findet sich mehrfach in der Literatur. So existieren sowohl kombinierte Ansätze, die unterschiedliche Formen der Ambiguität erkennen können (z. B. Tjong und Berry, 2013; Bajwa et al., 2012) als auch Ansätze, die beispielsweise Ambiguität sowie Unvollständigkeit erkennen (z. B. Körner, 2014; Huertas und Juárez-Ramírez, 2012; Fabbrini et al., 2001).

Einen Überblick über bestehende Forschungsansätze zur Disambiguierung im Kontext von natürlichsprachlichen Anforderungen geben Husain und Beg (2015) sowie Shah und Jinwala (2015). Shah und Jinwala (2015) unterscheiden dabei Verfahren im Wesentlichen hinsichtlich des Grades an Automatisierung, gewählten Ansatzes (Regelbasiert, Ontologie-basiert etc.) und verwendeten Technologien (z. B. *Stanford Parser*). Eine weitere umfangreiche Darstellung existierender Ansätze zur Disambiguierung im Anforderungskontext gibt Bano (2015), die den Fokus allerdings auf empirische Arbeiten legt. Im Folgenden werden kombinierte Ansätze der Erkennung und Kompensation von Ambiguitäten sowie Unvollständigkeit im Sinne einer bestmöglichen Gesamtübersicht aufgeführt (vgl. Tabelle 3.8).

Tabelle 3.8 listet kombinierte Ansätze hinsichtlich der Dimensionen „Defekte“, „Ziele“, „Eingabe / Ausgabe“ (I/O) und „Interaktion“ auf. „Defekte“ gibt die Abdeckung der Verfahren wieder. So handelt es sich bei NL2OCL und SR-Elicitor um Verfahren,

die eine geringe Abdeckung haben und bei QuARS und $\text{QuARS}_{express}$ um Ansätze, die eine Vielzahl von Defiziten abdecken. Darüber hinaus unterscheiden sich die Verfahren auch in der Zielsetzung, während QuARS das Ziel verfolgt, möglichst viele Defizite in Anforderungstexten zu erkennen, zielen NL2OCL und SR-Elicitor auf die Erkennung sowie Kompensation ab. Beide Verfahren verzichten dabei auf Benutzerinteraktion. Demgegenüber steht der RESI, welcher eine hohe Benutzerinteraktion bei der Kompensation ausgewählter Defizite vorsieht. RESI und der *Natural Language Automatic Requirement Evaluator* (NLARE) werden detaillierter vorgestellt, um die Unterschiede und Besonderheiten aufzuzeigen.

Huertas und Juárez-Ramírez (2012) stellen mit NLARE einen kombinierten Ansatz vor, der auf FA und die Erkennung von Ambiguität, Unvollständigkeit und Atomarität spezialisiert ist. Unter Ambiguität verstehen die Autoren steigerbare Adjektive und Adverbien. Unvollständigkeit bezieht sich auf den Abgleich mit vorgegebenen W-Fragen („*Who*“, „*What*“, „*Where*“, „*When*“) und Atomarität bezeichnet das Qualitätsmerkmal, dass ein einzelner Satz auch nur einen einzigen Anforderungsgegenstand beschreiben soll.

		NLARE[57]	NLARE2[58]	SREE[59]	RESI[60]	QuARS[61]	NL2OCL[62]	SR-Elicitor[63]	SRR-Director[64]	$\text{QuARS}_{express}$[65]	Smella[66]	AQUSA[67]
Defekte	Lexikalische Ambiguität	●	●	●	●	●	●	●	●	●	●	●
	Syntaktische Ambiguität	○	○	●	○	●	●	●	●	●	○	○
	Referentielle Ambiguität	○	○	●	○	●	○	○	○	●	○	○
	Vagheit	○	○	●	●	●	○	○	●	●	●	○
	Unvollständigkeit	●	●	●	●	●	○	○	○	●	●	●
	Lesbarkeit	○	○	○	○	●	○	○	○	●	○	●
	Konsistenz	○	○	○	○	●	○	○	○	●	●	●
	Atomarität	●	●	○	○	●	○	○	○	●	○	●
Ziele	Erkennung	●	●	●	●	●	●	●	●	●	●	●
	Kompensation	○	○	○	●	○	●	●	○	○	○	○
I/O	Strukturierte Eingabe	○	○	○	●	○	●	○	○	○	●	●
	Strukturierte Ausgabe	○	○	○	●	○	●	●	○	○	○	○
Interakt.	Hoch	○	○	○	●	○	○	○	○	○	○	○
	Mittel	○	○	○	○	●	○	○	○	●	○	○
	Niedrig	●	●	●	○	○	○	○	●	○	○	○
	Keine	○	○	○	○	○	●	●	○	○	●	●

Tabelle 3.8: Kombinierte Kompensationsverfahren

[57] vgl. Huertas und Juárez-Ramírez (2012).
[58] vgl. Huertas und Juárez-Ramírez (2013).
[59] vgl. Tjong (2008) sowie Tjong und Berry (2013).
[60] vgl. Körner und Brumm (2010) sowie Körner (2014).
[61] vgl. Lami (2005).
[62] vgl. Bajwa et al. (2012).
[63] vgl. Umber und Bajwa (2011).

NLARE nutzt zum Verarbeiten der natürlichen Sprache das *Natural Language Toolkit* (NLTK) und reguläre Ausdrücke (engl. *regular expressions, RegEx*). Die sequenzielle Verarbeitung umfasst neben einer Satzgrenzenerkennung und Tokenisierung eine Rechtschreibkorrektur als *Preprocessing* (vgl. Abbildung 3.8). Als Ausgabe erhalten Anwender Hinweise wie „*The requirement is ambiguous because it contains the word 'earlier' and 'later'*“ (Huertas und Juárez-Ramírez, 2012, S. 375). Eine Kompensation oder weitere Hilfestellung findet nicht statt.

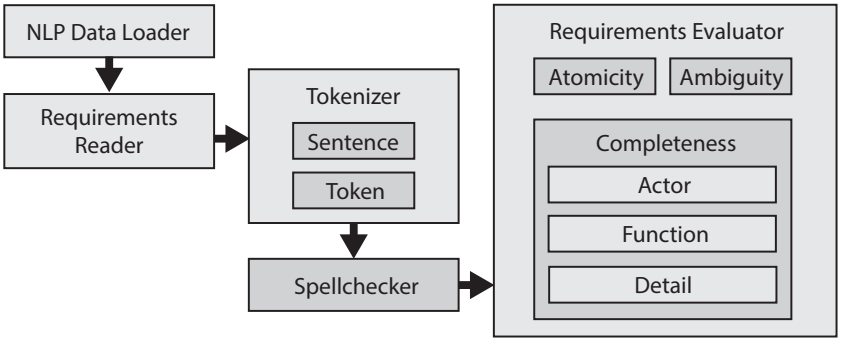


Abbildung 3.8: *Natural Language Automatic Requirement Evaluator.*
In Anlehnung an Huertas und Juárez-Ramírez (2012, S. 373)

Der RESI von Körner und Brumm (2010) bzw. Körner (2014) unterscheidet sich deutlich von NLARE im Hinblick auf (vorgesehene) Benutzerinteraktion, Flexibilität und Abdeckung linguistischer Defekte.

RESI ist in der Lage, Anforderungsspezifikationen als Graph einzulesen und automatisiert auf linguistische Defekte zu untersuchen. Werden Defekte gefunden, initiiert RESI einen Benutzerdialog (Körner und Brumm, 2010, S. 456). Hierbei wird nicht nur auf die problematischen Textstellen verwiesen, sondern explizite Kompensationshinweise für jede Art von Defizit (z. B. unvollständiges Prozesswort „*Returning-Something*“) sowie Ausprägung (z. B. „*SUBJECT: giver*“) gegeben (Körner und Brumm, 2010, S. 456 f.). Dies erfordert das Einbinden von Ressourcen, die zum einen das Erkennen der Defizite ermöglichen (Regeln) und zum anderen die zusätzlichen Informationen zur Kompensation bereitstellen (unterschiedliche Ontologien).

Abbildung 3.9 zeigt den Programmablauf von RESI. Nach dem Einlesen der strukturierten Anforderungsspezifikationen wird ein *Preprocessing* durchgeführt, welches POS-*Tagging* und Lemmatisierung umfasst. Das daraus resultierende Spezifikationsobjekt wird mittels angewandter Regeln auf linguistische Defizite untersucht, die vom Benutzer zuvor ausgewählt wurden. Wie Abbildung 3.9 zeigt, werden die Regeln iterativ auf das Spezifikationsobjekt angewandt und der Benutzer zur Kompensation erkannter Defizite aufgefordert. RESI exportiert nach Durchlauf aller gewählter Regeln das Spezifikationsobjekt als Graph.

[64] vgl. Rojas und Sliesarieva (2010).
[65] vgl. Bucchiarone et al. (2010).
[66] vgl. Femmer et al. (2016a).
[67] vgl. Lucassen et al. (2016).

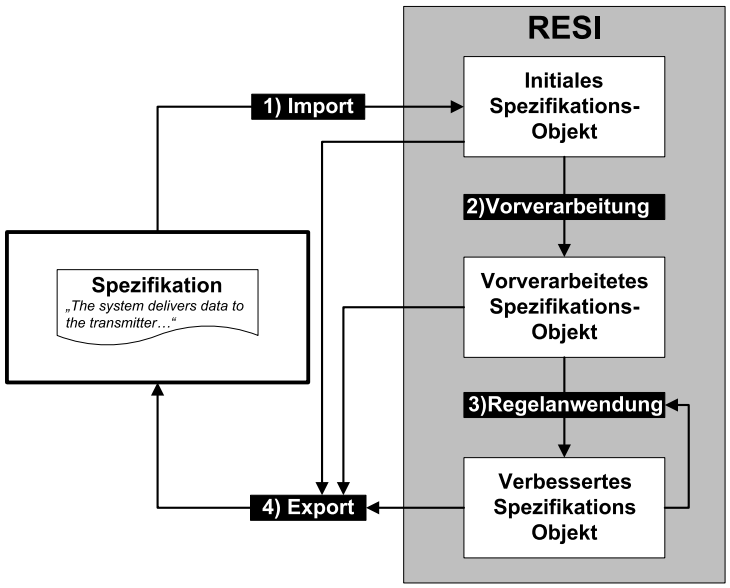


Abbildung 3.9: *Requirements Engineering Specification Improver.* Entnommen aus Körner (2014, S. 60)

3.4 Diskussion und Zwischenfazit

Natürlichsprachliche Anforderungsbeschreibungen ermöglichen es Endanwendern, an der Idee des OTF-*Computings* zu partizipieren und bedarfsgerechte Servicekompositionen zu nutzen. Wie dargestellt, sind dabei natürlichsprachliche Anforderungsbeschreibungen als Ausgangspunkt zu erwarten, die unvollständig und hochgradig ambig sind sowie stark im Umfang und Detailgrad variieren. Dies steht im Kontrast zu den bisher im OTF-*Computing* genutzten Spezifikationsansätzen für *Services*, die auf semi-formalen bzw. formalen Sprachen beruhen (Huma et al., 2012; Platenius et al., 2016). Diese sind ungeeignet, da Endanwender nicht über die notwendigen Fachkenntnisse verfügen und somit eine unüberwindbare Einstiegsbarriere vorfinden.

Die natürliche Sprache als Bestandteil der OTF-Vision bedeutet dabei, dass sie als alleinige Schnittstelle zum Endanwender fungiert. Alle notwendigen Informationen müssen aus den Anforderungsbeschreibungen, die Endanwender zur Verfügung stellen, extrahiert werden – unter Berücksichtigung genannter Defizite wie Unvollständigkeit und Ambiguität. Gleichzeitig muss die Interaktion mit dem Endanwender auf ein Minimum reduziert werden, um eine performante Bereitstellung der gewünschten Servicekomposition zu ermöglichen. Aus diesem Grund sind Anforderungsextraktions- und Kompensationverfahren erforderlich, die a) performant sind und b) keine bis minimale Benutzerinteraktion erfordern.

Die Extraktion von Anforderungen aus natürlichsprachlichen Beschreibungen stellt ein weitestgehend unbearbeitetes Forschungsfeld dar. Nur wenige Ansätze existieren, die in der Lage sind, Anforderungen aus Fließtexten zu extrahieren und auf die Kernelemente zu reduzieren. Nennenswert ist vor allem das von Dollmann und Geierhos (2016) entwickelte REaCT, dass zum einen eine Klassifikation von *On-* und *Off-Topic*-Inhalten vornehmen kann und zum anderen die Extraktion von semantischen Kernelementen vollzieht.

Demgegenüber handelt es sich bei Ungenauigkeit und Unvollständigkeit in Anforderungsbeschreibungen um Themen des REs, die mit großem Forschungsinteresse seitens der Wissenschaftsgemeinschaft einhergehen. Dies mag der Tatsache geschuldet sein, dass es sich dabei nicht um reine RE-Themen handelt, sondern um Themen, die viele wissenschaftliche Fachbereiche tangieren, darunter insbesondere die Computerlinguistik mit dem Ziel der maschinellen Textverarbeitung.

Nicht zu unterschätzen ist allerdings die Praxisrelevanz dieser Thematik, die sich in auffällig vielen Kooperationen zwischen Praxis und Wissenschaft in diesem Bereich abzeichnet. Fehlerhafte Softwareanforderungen stellen eine Gefahr für den Projekt- und Unternehmenserfolg dar und betreffen sowohl kleine Softwaremanufakturen als auch große Softwarehäuser. Schon längst sehen sich neben diesen klassischen Softwareherstellern zum Beispiel auch Autohersteller (z. B. Mercedes Benz) und Raumfahrtbehörden (z. B. NASA[68]) mit der Notwendigkeit, natürlichsprachliche Softwareanforderungen einer Qualitätskontrolle und Kompensation zu unterziehen, konfrontiert. Die Notwendigkeit, diese Gefahr zu minimieren, geht mit der Erkenntnis einher, dass natürlichsprachliche Anforderungen nach wie vor notwendig sind und Ungenauigkeit und Unvollständigkeit gleichzeitig so vielfältig auftreten können, dass softwareseitige Unterstützung und automatische Kompensation erforderlich ist.

Als Resultat dieser Bemühungen existiert eine Vielzahl an Ansätzen und Verfahren, die spezifische oder mehrere Formen von Ungenauigkeit und Unvollständigkeit erkennen und in manchen Fällen kompensieren können. Diese Ansätze und Verfahren unterscheiden sich im Vorgehen, Grad der Automatisierung, Aus- und Eingabeformaten, Performanz sowie zahlreichen Annahmen und Umweltfaktoren (vgl. Tabelle 3.8). Gemein haben sie, dass der Fokus der Entwicklung auf der Erkennung und/oder Kompensation liegt. Dies ist ein wesentlicher Unterschied zu den Anforderungen, die in dieser Arbeit an die Verfahren gestellt werden (z. B. weitestgehende Automatisierung).

Doch wie passen diese bestehenden Arbeiten zu der Vision des OTF-*Computings*? Wie können sie dabei helfen, Anforderungsbeschreibungen, die hochgradig individuell sind und auf mehreren Ebenen fehleranfällig sein können, soweit zu verbessern, dass sie die bedarfsgerechte Komposition von Softwareservices ermöglichen? Und lassen sich die Ansätze und Verfahren kombinieren, sodass sowohl die Notwendigkeit ihrer Anwendung erkannt werden kann als auch die Anwendung einzelner Komponenten im Einklang einer synergetischen Kompensationsstrategie steht?

Am Beispiel der Ambiguität von Lexemen wird die facettenreiche Problematik im Kontext des OTF-*Computings* greifbar. So ist es beispielsweise schnell ersichtlich, dass eine maschinelle Verarbeitung des Wortes „senden“ mit seinen acht Lesarten zu Verständnisproblemen führen kann, auch wenn für Endanwender im Moment der Anforderungsbeschreibung für eine E-Mail-Anwendung nur die Lesart „*transmitted to another place*“ im Vordergrund steht. Für die lexikalische Disambiguierung existiert dabei eine Vielzahl an Verfahren, die die wahrscheinlichste Lesart im Kontext einer Anforderungsbeschreibung ermitteln können. Derzeit eine der vielversprechendsten Softwarelösungen außerhalb des RE-Kontextes ist Babelfy, die unstrukturierten Fließtext entgegennehmen kann und disambiguierte Lesarten pro Lexem wiedergibt, ohne dabei Benutzerinteraktion zu erfordern. Auf Grund einer heterogenen, umfangreichen Datenbasis ist Babelfy domänenübergreifend einsetzbar.

[68] *National Aeronautics and Space Administration.*

Liegt der Fokus nicht mehr auf dem einzelnen Lexem sondern auf der Zusammenfügung von Wörtern, steigt die Komplexität der Erkennung und Kompensation. In dieser Arbeit wird unter dem Begriff der syntaktischen Ambiguität sowohl das Phänomen der Anbindungsambiguität im Falle von Präpositionalphrasen als auch die Koordinationsambiguität zusammengefasst. Wie aufgezeigt wurde, bestehen auch hier bereits Ansätze und Verfahren, allerdings bei weitem nicht so viele und etablierte, wie bei der lexikalischen Ambiguität. Wird der RE-Kontext bei der Auswahl eines Verfahrens hinzugenommen, verringert sich die Anzahl erneut erheblich.

Die Erkennung von potentiell ambigen Strukturen ist sowohl bei der PP-Anbindungsambiguität als auch bei der Koordinationsambiguität über syntaktische Muster möglich. Dies ist im Rahmen dieser Arbeit von besonderem Vorteil, da der Abgleich mit Mustern sehr performant durchgeführt werden kann und die Entscheidung, ob eine Kompensationsmethode aufgerufen werden muss oder nicht, in diesen Fällen sehr zuverlässig und ohne großen Aufwand erfolgt.

Die Disambiguierung gestaltet sich dann allerdings komplizierter, da nur wenige Arbeiten zur Auflösung von Koordinationsambiguität und PP-Anbindungsambiguität existieren, die im Rahmen dieser Arbeit Anwendung finden können. Im Falle der Koordinationen existieren zum Beispiel Ansätze, die linguistische Merkmale der Konjunktion heranziehen oder das gemeinsame Auftreten von lexikalischen Einheiten (Modifikatoren und verbundene Wörter) berücksichtigen. Das Ableiten von Regeln und Mustern auf Grundlage verschiedener linguistischer Ressourcen erscheint dabei zielführend und performant zugleich. Auch für die Disambiguierung von PP-Anbindungen existieren Verfahren, die auf Zusatzwissen zur Auflösung zurückgreifen (z. B. WordNet, VerbNet). Darüber hinaus werden Verfahren des maschinellen Lernens genutzt, die sehr gute Ergebnisse erzeugen.

Wie dargestellt werden konnte, existieren darüber hinaus mehrere Ansätze und Verfahren der Anaphernresolution und der automatischen Koreferenzresolution. Die Ansätze können dabei grob in „Linguistische Ansätze“, „Heuristiken“ und „Maschinelles Lernen“ unterteilt werden. Hervorgehoben werden kann dabei die *dcoref*-Komponente, die Bestandteil des *Stanford CoreNLP Natural Language Processing Toolkits* ist. Sie ermöglicht die Erkennung von Referenzausdrücken und die Auflösung von pronominaler sowie nominaler Koreferenz, ist als externe Programmkomponente konzipiert und befindet sich in aktiver Entwicklung.

Unvollständige Softwareanforderungen zu erkennen ist Gegenstand einer Vielzahl an Publikationen. So werden beispielsweise *Reviews*, Lesetechniken und Checklisten als manuelles Vorgehen zum einen und softwareunterstützte Anforderungsabgleiche mit Qualitätsmodellen, Ontologien oder Anforderungsdokumenten zum anderen vorgeschlagen. Vielfach liegt der Fokus dabei auf der Erkennung gänzlich fehlender Anforderungen und nicht auf Unvollständigkeit im Sinne fehlender Teilinformationen. Die Kompensation unvollständiger Softwareanforderungen ist ein weniger mit Publikationen bedachtes Forschungsfeld, indem vor allem die Ansätze von Körner und Brumm (2010) sowie Geierhos und Bäumer (2016) nennenswert sind. Beide Arbeiten legen den Fokus auf unvollständige Prädikate („Prozesswörter“) als semantisches Zentrum einer FA. Während beide Arbeiten unvollständige Prädikate erkennen und eine Kompensation initiieren, kann nur das Verfahren von Bäumer und Geierhos (2016) eine automatische Kompensation auf Basis ähnlicher Anforderungsbeschrei-

bungen vornehmen und die Ergebnisse strukturiert ausgeben. Damit ist es geeignet, in ein automatisiertes Kompensationssystem aufgenommen zu werden und kann dabei helfen, fehlende Angaben von Endanwendern zu kompensieren, noch bevor die Komposition geeigneter *Services* erfolgt.

Eine wirkliche Zusammenführung der genannten Verfahren im Sinne einer weitestgehend automatisierten Gesamtstrategie zur Verbesserung von Anforderungsbeschreibungen existiert nicht, wenn auch einzelne Verfahren mehrere Defizite abdecken: So deckt beispielsweise das populäre *Tool* QuARS eine ganze Reihe von Ambiguitäten und anderen Qualitätsmerkmalen (z. B. Lesbarkeit) ab. Eine Kompensation findet aber nicht statt, sodass eine hohe Benutzerinteraktion zumindest bei der Kompensation notwendig ist. Benutzerinteraktion ist dabei auch ein wesentlicher Einflussfaktor der Performanz und Akzeptanz des gesamten Verarbeitungsvorgangs und als kritisch zu bezeichnen. Endanwender erwarten, dass ihre Anforderungen *on-the-fly* verarbeitet werden und das sie schnellstmöglich passende Servicekompositionen präsentiert bekommen. Eine wiederholte Nachfrage bezüglich der Auflösung von Ambiguitäten oder der Kompensation von Unvollständigkeit wäre ein ermüdender, langsamer Prozess. Darüber hinaus kann nicht sichergestellt werden, dass die Endanwender überhaupt die Ambiguität oder die Unvollständigkeit erkennen, was sehr wahrscheinlich zu einem Abbruch der Anforderungsbeschreibung führen würde. Diesbezüglich wäre zum Beispiel NL2OCL eine Alternative, das den Fokus auf die Kompensation lexikalischer und syntaktischer Ambiguität legt und keine Benutzerinteraktion vorsieht. Allerdings sind für die Auflösung strukturierte Zusatzinformationen erforderlich, die im betrachteten OTF-Szenario mit Endanwendern nicht vorliegen.

Es bestehen also durchaus Arbeiten, die mehrere Formen von Ungenauigkeit und Unvollständigkeit erkennen und/oder kompensieren können. Allerdings gehen diese Arbeiten oftmals in der Erkennung und Kompensation strikt iterativ vor, ohne die Auswirkungen der Kompensation auf die Anforderungsbeschreibung und auf Folgekomponenten zu berücksichtigen. Wie wirkt sich beispielsweise die Kompensation von Unvollständigkeit auf mögliche Ambiguitäten aus? Darüber hinaus wird in den wenigsten Fällen hinterfragt, ob die Anwendung eines Kompensationsschrittes zum Verständnis einer betroffenen FA wirklich notwendig ist – betrifft die Ambiguität zum Beispiel wirklich wesentliche Elemente einer FA oder kann die Kompensation im Sinne der Performanz gegebenenfalls übersprungen werden?

Ein weiterer Punkt, der durch bestehende Verfahren nicht abgedeckt wird, ist die Synergie zwischen einzelnen Verfahren. So wird in keinem bekannten Fall unterstützt, dass die Kompensation eines Defizits bereits wertvolle Informationen zur Lösung eines weiteren Defizits erzeugen kann.

Zusammenfassend kann festgestellt werden, dass mit REaCT mindestens ein geeignetes Verfahren zur Extraktion von Anforderungen aus Fließtexten besteht und eine Reihe an Ansätzen und Verfahren zur Erkennung und Kompensation von Ungenauigkeit und Unvollständigkeit in natürlichsprachlichen Anforderungsbeschreibungen existiert. Diese Verfahren sind zum größten Teil spezialisiert auf ein spezifisches Defizit und nicht primär für die Integration in ein Kompensationssystem vorgesehen. Entsprechend selten anzutreffen sind Überlegungen zu Benutzerinteraktion, Performanz, Synergien, Interoperabilität und Kompatibilität der Verfahren.

Teil II

Methodische Vorgehensweise

4 Zu leistende Arbeit

Das Ziel dieser Arbeit ist die strategiebasierte Erkennung und Kompensation von Ambiguität und Unvollständigkeit in natürlichsprachlichen Anforderungsbeschreibungen, dargestellt am Anwendungsfall des OTF-*Computings*. Aufbauend auf dem Stand der Wissenschaft und Technik (s. Kapitel 3) sowie anschließender Diskussion werden die Anforderungen an diese Arbeit im Folgenden weiter konkretisiert.

4.1 Konzeption eines strategiebasierten Anforderungskompensationssystems

Zur Kompensation von Ambiguität und Unvollständigkeit in Anforderungsbeschreibungen existiert eine Reihe von Verfahren, von denen aber nur wenige für eine Automatisierung und den Anwendungsfall des OTF-*Computings* geeignet sind – sei es aus Gründen mangelnder Performanz und Verfügbarkeit, hoher Benutzerinteraktion, fehlender Ressourcen und Weiterentwicklung oder Inkompatibilitäten (s. Abschnitt 3.4). Darüber hinaus handelt es sich mehrheitlich um Insellösungen, die zwar die Erkennung und/oder Kompensation von Ambiguitäten und Unvollständigkeit unterstützen, jedoch nicht für die Integration in ein automatisiertes Softwaresystem wie CORDULA vorgesehen sind. Ihnen mangelt es beispielsweise an Schnittstellen und standardisierten Ein- und Ausgabeformaten. Diese Umstände erfordern zum einen die Auswahl geeigneter Kompensationsverfahren und zum anderen Überlegungen zur Initialisierung und Steuerung der jeweiligen Verarbeitungskomponenten (s. Abschnitt 5.5) im resultierenden Softwaresystem (s. Kapitel 7).

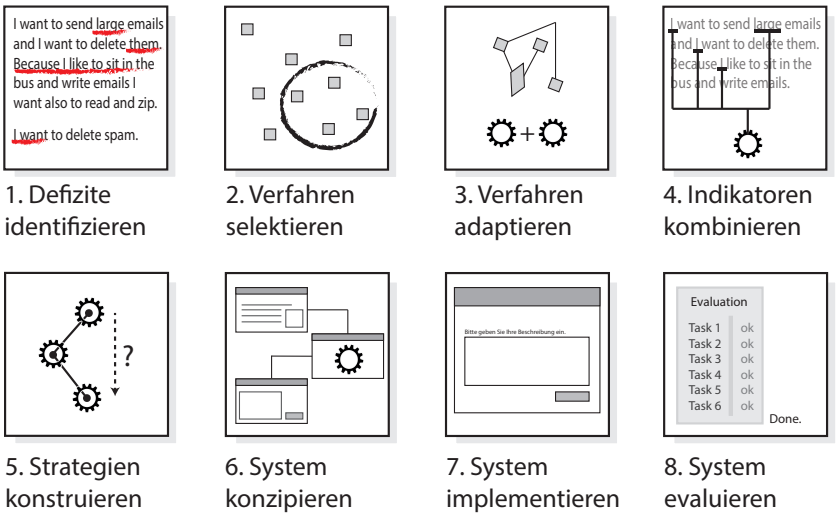


Abbildung 4.1: Methodische Vorgehensweise in der Dissertation

Abbildung 4.1 zeigt die stark abstrahierte methodische Vorgehensweise, die dieser Arbeit zugrunde liegt. Wie ersichtlich wird, sind aufbauend auf der Problemstellung Verfahren zur Erkennung und Kompensation lexikalischer, syntaktischer und referentieller Ambiguität sowie Unvollständigkeit zu selektieren und zu adaptieren, sodass sie kombiniert und zweckgebunden angewendet werden können.

Zusätzlich sind Steuerungsmechanismen notwendig (Indikatoren und Strategien), die die Notwendigkeit der Kompensation festlegen und die einzelnen Kompensationsverfahren steuern. Diese strategiebasierte Kompensation ist Teil eines Softwaresystems, das es im Rahmen dieser Dissertation zu konzipieren, zu implementieren und zu evaluieren gilt. Hierzu sind unter anderem lexikalische Ressourcen notwendig, die in Teilen noch nicht existent sind und daher entwickelt werden müssen.

Die Zusammenführung von Indikatoren, Strategien sowie Erkennungs- und Kompensationsverfahren wird innerhalb eines maschinellen Textanalysesystems vorgenommen. Hierzu muss zuerst ein Softwarekonzept entwickelt werden (s. Abschnitt 5.5), das unter anderem die Benutzerschnittstellen, einzelne Verarbeitungsschritte sowie Ein- und Ausgabeformate beinhaltet. Genauer gesagt umfasst die Konzeptionstätigkeit:

- Ermittlung notwendiger *Preprocessing*-Schritte für Anforderungsbeschreibungen und Konzeption einer entsprechenden *Preprocessing pipeline*
- Ermittlung der, zur Problemlösung und Implementierung geeigneten, Erkennungs- und Kompensationsverfahren
- Bedarfsgerechte Erweiterung ermittelter Verfahren im Sinne einer performanten, zielführenden Kompensation
- Definition der Ausgabeparameter zur maschinellen Weiterverarbeitung kompensierter sowie strukturierter FA im Anwendungsfall des OTF-*Computings*
- Definition notwendiger Ausgabeparameter der Benutzerauskunft (Ergebnisse, Erläuterungen, Kompensationsprotokolle)
- Abgleich des Konzepts mit Qualitätsmerkmalen der Softwareentwicklung (insb. Leistungsfähigkeit, Adaptierbarkeit, Wartbarkeit)

Auf die Lösung dieser Teilziele folgt die Implementierung des Textanalysesystems (CORDULA), das als funktionaler Prototyp umgesetzt wird. Dies bedeutet, dass die grundsätzliche Funktionsfähigkeit des Konzepts aufgezeigt und evaluiert werden kann. Hierzu sind als Implementierungsgegenstände insbesondere zu benennen:

- Entwicklung plattformübergreifender Benutzerschnittstellen zur Eingabe unstrukturierter Anforderungsbeschreibungen sowie zur Ausgabe kompensierter, strukturierter FA und Erläuterungen zum Kompensationsprozess
- Implementierung ausgewählter Verarbeitungskomponenten
- Implementierung der entwickelten Indikatoren zur Erkennung potentieller Ambiguität und Unvollständigkeit

- Implementierung der entwickelten Strategien zur Steuerung ausgewählter Verarbeitungskomponenten in CORDULA
- Bereitstellung einer standardisierten Schnittstelle zur maschinellen Weiterverarbeitung kompensierter FA im Anwendungsfall des OTF-*Computings*

Die Konzeption und Implementierung geht dabei einher mit folgenden Fragen:

- Wie kann die Kompensation performant durchgeführt werden?
- Können Synergien zwischen den Verarbeitungskomponenten die Kompensationsergebnisse verbessern?
- Welche Softwarearchitektur ist für das Konzept, insbesondere im Anwendungsfall des OTF-*Computings*, geeignet?

4.1.1 Auswahl geeigneter Kompensationsverfahren

Der Arbeitsschwerpunkt liegt auf der Kombination ausgewählter Verfahren zwecks automatischer Ausführung und somit weniger auf der Erstellung oder Optimierung von Kompensationsverfahren. Diesbezüglich hat sich bereits in Abschnitt 3.3 ein sehr heterogenes Bild bestehender Verfahren ergeben. Aus diesem Grund gilt es als ein Teilziel, die für diese Arbeit geeigneten Kompensationsverfahren zu identifizieren. Als grundsätzlich geeignet wird ein Verfahren angenommen, wenn es ...

- ... mindestens eine konkrete Erkennung und/oder Kompensation vollzieht (z. B. Disambiguierung von Koordinationsambiguität),
- ... über Schnittstellen zur Integration verfügt (z. B. als Programmbibliothek existiert),
- ... standardisierte Ein- und Ausgabeformate unterstützt (z. B. etablierte NLP-Formate wie CoNLL-U[69]),
- ... keine Benutzerinteraktion zwingend voraussetzt,
- ... vollständig zugänglich und frei verfügbar ist,
- (... sich in aktiver Entwicklung befindet).

Darüber hinaus können Anforderungen an ein Verfahren bestehen, die sich aus der spezifischen Kompensation oder Implementierung heraus ergeben (z. B. notwendige Konfigurationsparameter um Synergien nutzen zu können) und die hier nicht aufgeführt sind. Die Frage, die es in den folgenden Abschnitten und insbesondere im Abschnitt der Systemkonzeption (s. Abschnitt 5.5) zu beantworten gilt, ist:

[69] Siehe weiterführend: `http://universaldependencies.org/format.html` (Stand: 05.03.2017).

- Welche Verfahren eignen sich für die Erkennung und Kompensation lexikalischer und referentieller Ambiguität, Koordinationsambiguität, PP-Anbindungsambiguität sowie Unvollständigkeit in UGC unter den Gesichtspunkten der Automatisierung und hoher Performanz?

Allein die Auswahl der Verfahren reicht nicht aus, um zum einen ihre Wechselwirkungen bei gemeinsamer Ausführung abschätzen zu können und zum anderen sie vollautomatisiert zu implementieren. Aus diesem Grund ist die Entwicklung weiter fortgeschrittener Kompensationsstrategien erforderlich.

4.1.2 Entwicklung fortgeschrittener Kompensationsstrategien

Wie zuvor dargestellt, werden ausgewählte Verfahren im Sinne der Automatisierung und Performanz kombiniert. Dabei stellt sich die Frage, wie die Verfahrenskombinationen gestaltet werden können, damit zum einen nur die Verfahren ausgeführt werden, die wirklich im Falle einer spezifischen Anforderungsbeschreibung benötigt werden und zum anderen diese so miteinander interagieren, dass sie sich gegenseitig unterstützen. Das bedeutet allerdings auch, dass mit Widersprüchen und Konflikten zu rechnen und seitens des Systems umzugehen ist. Dies wird in dieser Arbeit über kontextsensitive Indikatoren (Erkennung von Kompensationsbedarf) und Kompensationsstrategien (Steuerung der Verarbeitungskomponenten) realisiert, deren Entwicklung elementares Teilziel dieser Arbeit ist. Indikatoren können hier als Qualitätsmerkmale in den Anforderungsbeschreibungen verstanden werden, die Aufschluss über potentielles Vorkommen von Ambiguität und Unvollständigkeit geben und die Ausführung von Kompensationsstrategien begründen. Kompensationsstrategien greifen daraufhin auf gefundene Indikatoren zurück und kombinieren entsprechend der Beschreibungsqualität bedarfsgerecht die Kompensationsverfahren. Sie definieren auch den Informationsumfang im Kommunikationsprozess zwischen einzelnen Verfahren. Ein solches Vorgehen der optimalen Kombination bestehender Verfahren zur Kompensation von ambigen und unvollständigen Anforderungsbeschreibungen existiert derzeit noch nicht. Die daher notwendige Entwicklung fortgeschrittener Kompensationsstrategien wird von folgenden Fragestellungen begleitet:

- Welche Indikatoren können identifiziert werden, um Formen der Ambiguität und Unvollständigkeit in Anforderungsbeschreibungen zu erkennen?
- Welche Strategien sind notwendig, um die Kompensation von Ungenauigkeit und Unvollständigkeit flexibel und performant durchzuführen?
- Können Kompensationsverfahren sich gegenseitig im Sinne einer korrekten Entscheidungsfindung unterstützen?

4.1.3 Erstellung linguistischer Ressourcen

Wie in Abschnitt 3.2 und insbesondere von Tichy et al. (2015) dargestellt, gibt es nicht genügend linguistische Ressourcen für natürlichsprachliche Anforderungsbeschreibungen. Diese annotierten Korpora werden benötigt, um die hier angesprochenen

Indikatoren, Strategien und Verarbeitungsschritte der Anforderungskompensation für das Softwaresystem zu entwickeln und zu evaluieren. Es existieren beispielhafte Anforderungsbeschreibungen aus Lehrbüchern. Darüber hinaus gibt es vereinzelnd Beschreibungen, die im Internet frei zugänglich über Suchmaschinen gefunden werden können. Auch Tichy et al. (2015) stellen eine Sammlung von Anforderungsbeschreibungen zur Verfügung, allerdings sind auch dort längst nicht alle der insgesamt (derzeit) 46 englischsprachigen Beschreibungstexte im Rahmen dieser Arbeit nutzbar: So befinden sich Texte in natürlicher aber auch kontrollierter Sprache unter diesen Beschreibungen, wovon aber viele Lehrbeispiele sind. Darüber hinaus sind Beschreibungen zwar aufgeführt aber nicht mehr abrufbar. Aus diesem Grund ist ein Teilziel dieser Arbeit, eigene linguistische Ressourcen zu erstellen, die insbesondere für die verschiedenen Anwendungsfälle in dieser Arbeit geeignet sind. Geeignet heißt in diesem Fall, dass sie hinsichtlich domänenspezifischer Merkmale ähnlich zu Anforderungsbeschreibungen sind. Im Folgenden werden die erforderlichen Ressourcen in den Kontext dieser Arbeit eingebettet. Die Erstellung fehlender Ressourcen wird in Kapitel 6 beschrieben und begleitet von folgenden Fragestellungen:

- Aus welchen frei zugänglichen Quellen können Anforderungsbeschreibungen akquiriert werden?
- Welche Eigenschaften müssen diese Beschreibungen aufweisen, um als ähnlich zu denen zu gelten, die im OTF-*Computing* zu erwarten sind?
- In welchem Format und hinsichtlich welcher Struktur sind die akquirierten Beschreibungen abzuspeichern?

4.1.3.1 Anforderungsbeschreibungskorpus

Um mehr über die Charakteristika von Anforderungsbeschreibungen zu erfahren, die Gegenstand dieser Arbeit sind, ist eine Sammlung eben dieser Beschreibungen erforderlich. Wie allerdings bereits dargestellt, existiert so ein Korpus derzeit nicht. Diesem Problem kann auf zwei Arten begegnet werden:

Zum einen können Anforderungsbeschreibungen durch eine Personengruppe unter Anleitung verfasst werden, wobei die Anleitung in diesem Fall nur die grobe Vorgabe eines Anwendungsszenarios im OTF-*Computings* umfasst. Dies hätte den Vorteil, dass die Beschreibungen in den OTF-Kontext passen. Allerdings geht dies mit den gleichen Nachteilen einher, die auch die Anforderungsbeschreibungen haben, die aus Lehrbüchern extrahiert werden: Sie sind, trotz größtmöglicher Vermeidung von Beeinflussung, konstruiert. Zum anderen kann auf ähnliche Beschreibungen zurückgegriffen werden, die bereits existieren. Ähnlich bedeutet dabei, dass diese Beschreibungen bestimmte Merkmale mit den Softwarebeschreibungen gemein haben, die im OTF-*Computing* zu erwarten sind. Positiv hervorzuheben ist, dass diese Anforderungsbeschreibungen nicht aufgrund eines Arbeitsauftrags oder sonstiger Einflussnahme verfasst wurden und somit Charakteristika realer Anforderungsbeschreibungen aufzeigen. Allerdings wären diese Anforderungen nicht OTF-spezifisch und bezögen sich nicht zwangsläufig auf Softwareservices. Wie allerdings in Abschnitt 3.2 dargestellt, existieren auch keine Korpora, die ähnliche Beschreibungen umfassen. Dies bedeutet, dass in beiden Fällen das Erstellen entsprechender Korpora

unabdingbar ist. In dieser Arbeit wird deshalb die Erstellung einer solchen Sammlung von Beschreibungen als Teilziel umgesetzt. Daher gilt es, frei zugängliche (insb. online verfügbare) Anforderungsbeschreibungen zu akquirieren und in einem Korpus zusammenzuführen (s. Abschnitt 6.1). Diese Anforderungsbeschreibungen müssen von Endanwendern in natürlicher Sprache (Englisch) verfasst sein und sind nicht Teil einer professionellen Anforderungsdokumentation.

4.1.3.2 Softwarespezifisches PAS-Korpus

Wie in Abschnitt 2.3 dargestellt, wird Unvollständigkeit über unbesetzte Stellen (d. h. fehlende Instantiierung) in der Prädikat-Argument-Struktur (PAS) einer FA definiert. Um dabei automatisch zu ermitteln, welche Prädikate mit welchen Argumenten einhergehen, um eine Anforderung als vollständig zu bezeichnen, ist das Nachschlagen in einer entsprechenden Ressource erforderlich, die diese domänenspezifischen Angaben enthält (s. Abschnitt 2.3). Eine solche Ressource existiert speziell für Softwareanforderungsbeschreibungen derzeit noch nicht. Wohl aber existieren allgemeine sprachliche Ressourcen, die die Valenz eines Prädikats abspeichern (z. B. *Propbank*). Deshalb ist ein weiteres Teilziel dieser Arbeit, generelle Valenzinformationen (über die Stelligkeit eines Prädikats) mit seiner domänenspezifischen Verwendung zu verknüpfen. Dafür ist eine Ressourcenerweiterung notwendig, die im Rahmen dieser Arbeit zu leisten ist: So kann eine binäre Angabe für jede Lesart eines Prädikats Aufschluss darüber geben, ob ein Argument erforderlich ist oder nicht. Um die Prädikate dabei einer bestimmten Domäne zuordnen zu können, sind Korpora notwendig, die hinsichtlich ihrer Prädikat-Argument-Struktur analysiert wurden. In dieser Arbeit wird diese Analyse für die Domäne der Softwareanforderungen und im speziellen für Anforderungen aus dem Bereich der E-Mail-Kommunikation geleistet[70].

Die bisher geplante Ressourcenerweiterung erlaubt zwar die Erkennung von Unvollständigkeit, jedoch noch nicht deren Kompensation. Für die Kompensation sind Informationen erforderlich, die Aufschluss über die Möglichkeiten zur Vervollständigung einer leeren Argumentposition im spezifischen Kontext einer Anforderungsbeschreibung geben. Um eine größtmögliche Abdeckung verschiedener Kontexte zu erreichen, ist ein Ressourcenumfang erforderlich, der das Korpus aus Abschnitt 4.1.3.1 überschreitet und einen hohen Variantenreichtum (mit Bezug auf Kontexte) aufweist. Die Vorstellung des PAS-Korpus findet in Abschnitt 6.2 statt.

4.2 Evaluation des Textanalysesystems

Die Funktionsweise des Konzepts bzw. des funktionalen Prototyps von CORDULA gilt es im Rahmen einer Evaluation zu prüfen (s. Kapitel 8). Hierbei liegt zum einen der Evaluationsschwerpunkt auf den entwickelten Indikatoren und Strategien (s. Abschnitt 5.2). Es gilt zu evaluieren, ob die Strategiewahl und -anwendung erwartungsgemäß funktioniert (s. Abschnitt 4.2.1). Dabei ist die Zuverlässigkeit der zugrundeliegenden Indikatoren von besonderer Bedeutung. In diesem Zusammenhang gilt es Fehlertypen zu identifizieren, die bei der weiteren Entwicklung berücksichtigt

[70] E-Mail-Kommunikation wird hier als gemeinsames Beispielszenario der Arbeitsbereiche im Teilprojekt B1 „*Parameterized Service Specifications*“ des SFB901 OTF-*Computing* aufgegriffen.

werden müssen. Zum anderen ist sowohl die Performanz des Gesamtsystems als auch der Indikatoren, Strategien und Komponenten zu evaluieren (s. Abschnitt 4.2.2), da somit erstens die Identifikation von Leistungsengpässen im Softwaresystem ermöglicht wird und zweitens eine Performanzverbesserung die Steigerung der Nutzerzufriedenheit und -akzeptanz erwarten lässt (s. Abschnitt 7.4).

4.2.1 Evaluation der Strategieanwendung

Die Evaluation der Strategieanwendung (s. Abschnitt 8.2) betrifft sowohl die indikatorbasierte Strategiewahl (Korrektheit) als auch die Anwendungshäufigkeit der einzelnen Strategien (Aufteilung). Darüber hinaus ist von Interesse, wie sich das Softwaresystem verhält, wenn keine geeignete Strategie für eine vorgefundene Indikatorkombination vorhanden ist und wie oft diese Situation mit den, in dieser Arbeit vordefinierten, Kompensationsstrategien auftritt (Abdeckung). Hieraus ergeben sich folgende Evaluationsaufträge:

- Evaluierung der jeweiligen Strategiehäufigkeit bei der Strategiewahl
- Evaluierung der Indikatorzuverlässigkeit
- Evaluierung der Abdeckung auftretender Indikatorkombinationen

Neben diesen Evaluationsaufträgen mit direktem Bezug zur Strategieanwendung ist auch die Identifikation von Fehlertypen Gegenstand dieses Evaluationsteils. Hierbei ist das Ziel, Fehler in der indikatorbasierten Strategieauswahl zu identifizieren und zu systematisieren, um sie in der Folgeentwicklung berücksichtigen zu können.

- Identifikation von Fehlern, die die Indikatorzuverlässigkeit negativ beeinflussen und somit die Strategieauswahl erschweren
- Identifikation der Auswirkung von Indikatorabhängigkeiten[71] auf die Qualität der Strategieauswahl

Folgende Fragen sind bei der Evaluation der Strategieanwendung zu berücksichtigen:

- Welche Datenbasis (Evaluationskorpus) eignet sich für die Evaluation?
- Nach welchen Kriterien kann die Fehlertypisierung erfolgen?

4.2.2 Evaluation der Systemperformanz

Die Evaluation der Performanz (s. Abschnitt 8.2) kann auf Systemebene aber auch auf Ebene der Verarbeitungskomponenten, Strategien und Indikatoren erfolgen. Performanz (im Sinne von Laufzeit) wird in dieser Arbeit dabei in Initialisierungs-, Ausführungs- und Gesamtlaufzeit unterteilt, um einen detaillierteren Einblick in die Laufzeiten zu erhalten. Da die Laufzeit des System unweigerlich von den gewählten

[71] Teilinformationen (z. B. semantische Kategorien) werden in mehreren Indikatoren herangezogen.

Kompensationsstrategien abhängt und die Laufzeit der Strategien wiederum maßgeblich (aber nicht ausschließlich) durch die gewählten Kompensationsmethoden bestimmt wird, empfiehlt sich eine Evaluation der Performanz auf allen Ebenen. Hierbei ist von besonderem Interesse, wie sich das Softwaresystem unter steigender Last verhält (bspw. die Verarbeitungszeit linear zur Last steigt) oder wie einzelne Komponenten zur Gesamtlaufzeit beitragen. Diesbezüglich sind ebenfalls die implementierten Maßnahmen zur Performanzsteigerung (insb. *Caching*-Ansätze) hinsichtlich Nutzen und Entwicklung (z. B. Speicherumfang) zu evaluieren.

Mit Bezug auf CORDULA als Gesamtsystem sind sowohl die Ausführungszeiten (insb. unter steigender Last bzw. Anforderungsumfang) als auch die Initialisierungszeiten von Interesse. Darüber hinaus ist ein hoher Anteil nebensächlicher Angaben in den Anforderungsbeschreibungen zu erwarten (s. Abschnitt 1.4), weshalb die Frage nach dem Einfluss dieser nebensächlichen Angaben auf die Systemlaufzeit aufkommt. Folgende Evaluationsaufträge lassen sich für die weitere Vorgehensweise ableiten:

- Messung der Ausführungszeit bei zunehmendem Beschreibungsumfang
- Messung Initialisierungszeit auf Systemebene
- Evaluierung möglicher Laufzeitbeeinflussung nebensächlicher Angaben
- Evaluierung der Strategielaufzeiten unter zunehmendem Beschreibungsumfang

Wie bereits angeführt, sind diese Laufzeiten unmittelbar beeinflusst von den Verarbeitungskomponenten, die daher ebenfalls zu evaluieren sind. Hierbei ist von Interesse, welche Verarbeitungskomponenten gewählter Methoden die Systemlaufzeit am meisten beeinflussen. Wobei sich hierbei auch die Frage stellt, ob nicht sogar einzelne Komponentenbestandteile für hohe Laufzeiten verantwortlich sind und diese, einmal identifiziert, nicht in der weiteren Entwicklung ausgetauscht werden können.

- Identifikation der Laufzeitanteile hinzugezogener Verarbeitungskomponenten sowohl im *Preprocessing* als auch in der Kompensation
- Identifikation von Wertebereichen, in denen die jeweilige Komponenteninitialisierungszeit schwankt
- Messung der Laufzeit einzelner Komponentenbestandteile

Als eine Möglichkeit der Performanzsteigerung wird das WSD-*Caching* implementiert. Um dabei den Effekt auf die Performanz nachzuweisen und die Entwicklung des Zwischenspeichers besser zu verstehen und beispielsweise abschätzen zu können, ob ein Sättigungseffekt[72] eintreten kann, sind folgende Evaluationsschritte notwendig:

- Messung der Laufzeit der lexikalischen Disambiguierung mit und ohne WSD-*Caching* und Bestimmung der Performanzsteigerung
- Evaluierung der Entwicklung und insb. des Zuwachses von Lesarten im Zwischenspeicher zur Identifikation eines Sättigungszustandes, bei dessen Erreichung keine weiteren Lesarten in den WSD-*Cache* aufgenommen werden

[72] Zustand, in dem keine weiteren Lesarten in den Zwischenspeicher aufgenommen werden.

- Evaluierung der Anfragenverteilung im WSD-*Cache* sowohl innerhalb einer Domäne als auch auf domänenübergreifenden Anforderungsbeschreibungen

Auf Grundlage dieser Arbeitsaufträge werden in Abschnitt 8.2 Evaluationsfragen erstellt, die es mittels geeigneter Evaluationsmethoden zu beantworten gilt. Die Evaluation der Systemperformanz wird dabei begleitet von folgenden Fragestellungen:

- Welche Datenbasis (Evaluationskorpus) eignet sich für die Evaluation?
- Wie lässt sich die Laufzeitmessung weitestgehend automatisieren?
- Welche Anforderungen sind an die Datenauswahl und -kombination zu stellen?

5 Konzeptentwicklung

Im Folgenden findet die Konzeptentwicklung statt, in deren Rahmen wesentliche Designentscheidungen getroffen werden, welche die Kompensationsstrategien (s. Abschnitt 5.2), deren zugrundeliegenden Indikatoren (s. Abschnitt 5.3) sowie das Anforderungskompensationssystem als Ganzes betreffen (s. Abschnitt 5.5). Darauf aufbauend werden in Abschnitt 7.4 Anforderungen an die Systemqualität diskutiert.

5.1 Ausgangssituation und Zielsetzung

Die Ausgangssituation dieser Arbeit sieht Endanwender vor, die Anforderungen an ein gewünschtes, individuelles Softwaresystem beschreiben und eine fertige Software als Resultat der Komposition von *Services* zurückerhalten (s. Abschnitt 1.1 sowie 1.4). Dies ist im Gegensatz zum klassischen *Requirements Engineering* (RE) ein agiler Prozess, der impulsiv und aus dem Bedarf heraus geschieht. Darüber hinaus kann die Anforderungsaufnahme beispielsweise am Mobiltelefon geschehen (vgl. Abbildung 5.1), wo im Vergleich zum PC, deutlich kürzere Beschreibungen zu erwarten sind, die dazu eine erhöhte Fehlerrate in Rechtschreibung und Grammatik aufweisen.

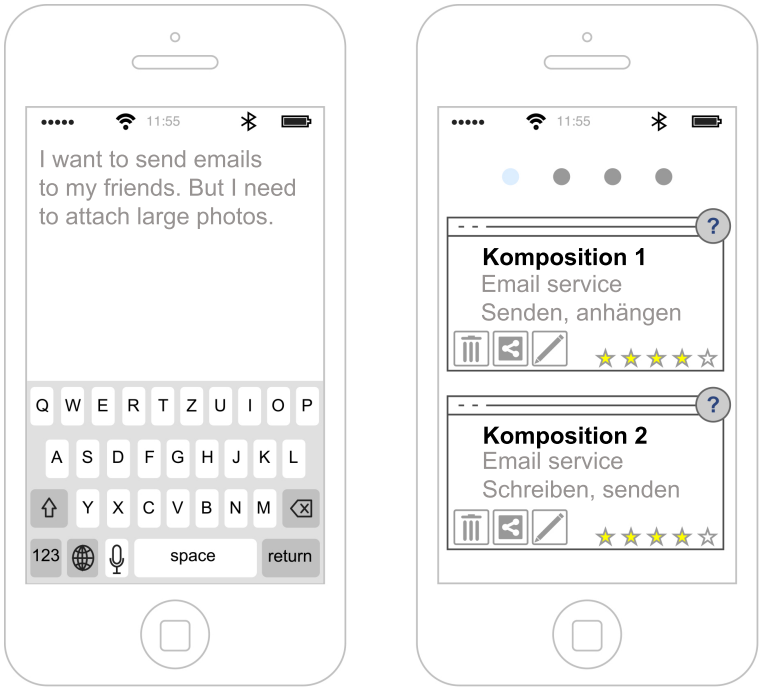


Abbildung 5.1: Smartphone als Benutzerschnittstelle (*Mockup*)

Gegenstand dieser Arbeit sind demnach Anforderungsbeschreibungen , die in Qualität und Umfang stark variieren. Unter diesen Bedingungen treten unweigerlich

Ambiguität und Unvollständigkeit als Phänomene natürlichsprachlicher Anforderungsdokumentation auf (s. Kapitel 2), die von existierenden Kompensationsansätzen nicht oder nur teilweise kompensiert werden können (s. Abschnitt 3.3). Diese Situation wird durch Rahmenbedingungen des OTF-*Computings* (z. B. kurze Ausführungszeit, minimale Benutzerinterakation) verschärft, die eine Anwendung bestimmter Ansätze ausschließen (s. Abschnitt 3.4). So können beispielsweise Ansätze, die zwar Ambiguität aufzeigen aber nicht kompensieren, nicht angewendet werden, da Endanwender ohne Hilfestellung (z. B. weiterführende Informationen) nicht fähig sind, Defizite in ihren Anforderungsbeschreibungen zu identifizieren und zu beheben.

Die Interaktion mit dem System, die über die initiale Eingabe der Anforderungsbeschreibung hinausgeht, ist, insbesondere im Hinblick auf die Gesamtperformanz sowie drohender Überforderung der Anwender, auf ein Minimum zu begrenzen. In Fällen, in denen Benutzerinteraktion unvermeidbar ist, sind Programmausgaben notwendig, die Benutzerinteraktion initiieren und steuern können. So reicht es beispielsweise nicht, ein unvollständiges Prädikat (s. Abschnitt 2.3) zur Hervorhebung anzukreuzen, wie in Abbildung 5.2 (1) dargestellt, da nicht davon auszugehen ist, dass die Unvollständigkeit ohne Zusatzinformationen seitens der Endanwender selbstständig behoben werden kann. Vielmehr ist die gewünschte Funktionalität, wie in Abbildung 5.2 (2) dargestellt, mit Hinweis auf fehlende Details (z. B. beispielhafte Argumente, Über- und Unterbegriffen oder semantische Kategorien) zu präsentieren. Auf diese Weise werden Endanwender schneller auf unvollständige Eingaben hingewiesen und können anhand der zusätzlichen Angaben ursprüngliche Defizite punktuell verbessern.

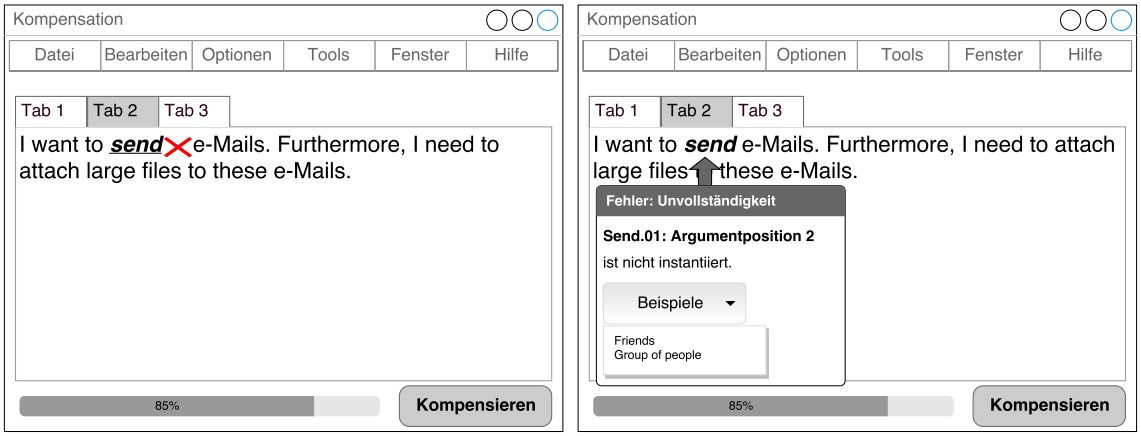


Abbildung 5.2: Erweiterte Benutzerinteraktion (*Mockup*).
Entnommen aus Bäumer und Geierhos (2016, S. 550)

Dem Ziel der Entwicklung eines Anforderungskompensationssystems folgend (s. Kapitel 4), ist die strategiebasierte Verkettung von Softwarekomponenten zur Erkennung und bedarfsgerechten Kompensation von Ambiguität und Unvollständigkeit (s. Kapitel 2) in Anforderungsbeschreibungen (s. Abschnitt 1.4) Gegenstand dieses Kapitels. Dies erfolgt unter Berücksichtigung der Rahmenbedingungen des OTF-*Computings*. Die Entwicklung von Strategien und Indikatoren ist erforderlich, um flexibel auf die schwankende Qualität der Anforderungsbeschreibungen reagieren zu können. Sie dienen dabei insbesondere der Steuerung der heterogenen, bisher iso-

liert betrachteten Verarbeitungskomponenten (z. B. Unvollständigkeitskompensation). Hierbei ist zu beachten, dass es nicht „die eine Anforderungsbeschreibung“ gibt und es daher auch nicht „die eine Strategie“ geben kann. Daher müssen die Strategien flexibel auf Inhalt, Form und Umfang der Beschreibungen reagieren können. Darüber hinaus sind Informationen bereitzustellen, die zum einen die Verarbeitung sowie Kompensation für den Endanwender transparent darstellen und zum anderen eine maschinelle Weiterverarbeitung für Folgekomponenten ermöglichen.

5.2 Strategiekonfiguration

Anforderungsbeschreibungen können in ihrer Qualität stark variieren und erfordern daher eine flexible Vorgehensweise in der Verarbeitung und Kompensation, die in dieser Arbeit über Strategien umgesetzt wird (vgl. Definition 5.2.1). Die Qualität wird dabei über Hinzunahme zuvor definierter Indikatoren (z. B. syntaktische Muster) in einer Anforderungsbeschreibung festgestellt (s. Abschnitt 5.3). Ziel ist es dabei, nicht irgendeine Strategie zu wählen, sondern diejenige, die ausschließlich notwendige Kompensationsschritte in optimaler Reihenfolge ausführt.

Definition 5.2.1 (Strategie)
Strategien umfassen Kompensationsmethoden, die auf spezifische Indikatoren einer gegebenen Anforderungsbeschreibung reagieren (z. B. Auswahl geeigneter Verfahren). Sie unterscheiden sich dabei in der Auswahl, Ausführung und Interaktion der Methoden.

Die Notwendigkeit verschiedener Strategien ergibt sich aus den möglichen Indikatorkombinationen. So kann eine Anforderungsbeschreibung beispielsweise frei von Ambiguität sein und bedarf daher keiner entsprechenden Kompensation. Sehr wohl aber ist die Extraktion von FA und die strukturierte Ergebnisausgabe erforderlich. Gäbe es nur eine einzige Strategie, würde die Verarbeitung und Kompensation ineffizient ausgeführt, da Kompensationsschritte initiiert würden, die für eine reine Anforderungsextraktion nicht notwendig sind.

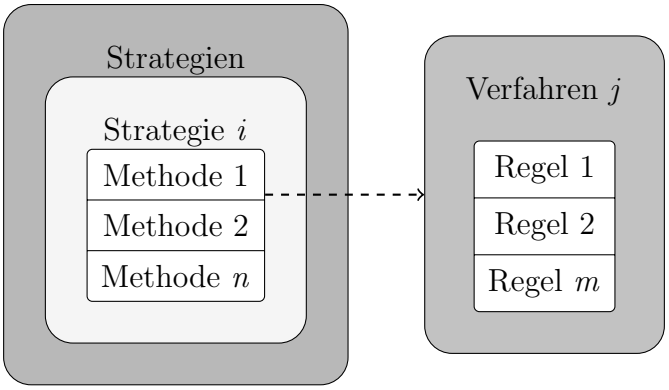


Abbildung 5.3: Logischer Aufbau von Strategien

Der Zusammenhang zwischen Strategien, Methoden und Verfahren ist in Abbildung 5.3 dargestellt. Methoden umfassen Verfahren zur Kompensation von Ambiguität und Unvollständigkeit. Exemplarisch könnte eine Methode der lexikalischen

Disambiguierung und eine weitere Methode zur Kompensation von Unvollständigkeit dienen. Somit wird ersichtlich, dass es sich bei den Methoden jeweils um Softwarekomponenten handelt, deren Verfahren auf einen spezifischen Verarbeitungs- oder Kompensationsprozess zugeschnitten sind. Hierbei können Verfahren beispielsweise auf Regeln (vgl. Abbildung 5.3) oder auf ML-Ansätze zurückgreifen. Strategien bedienen sich unterschiedlicher Methoden und wenden diese bedarfsgerecht auf die Anforderungsbeschreibung an, wobei dies auf Wortebene (z. B. Lexikalische Disambiguierung), auf Basis der Sätze (z. B. Syntaktische Disambiguierung) oder auf der gesamten Beschreibung (z. B. Referentielle Disambiguierung) erfolgen kann. Das Zusammenwirken der Indikatoren und Strategien ist in Abbildung 5.4 dargestellt.

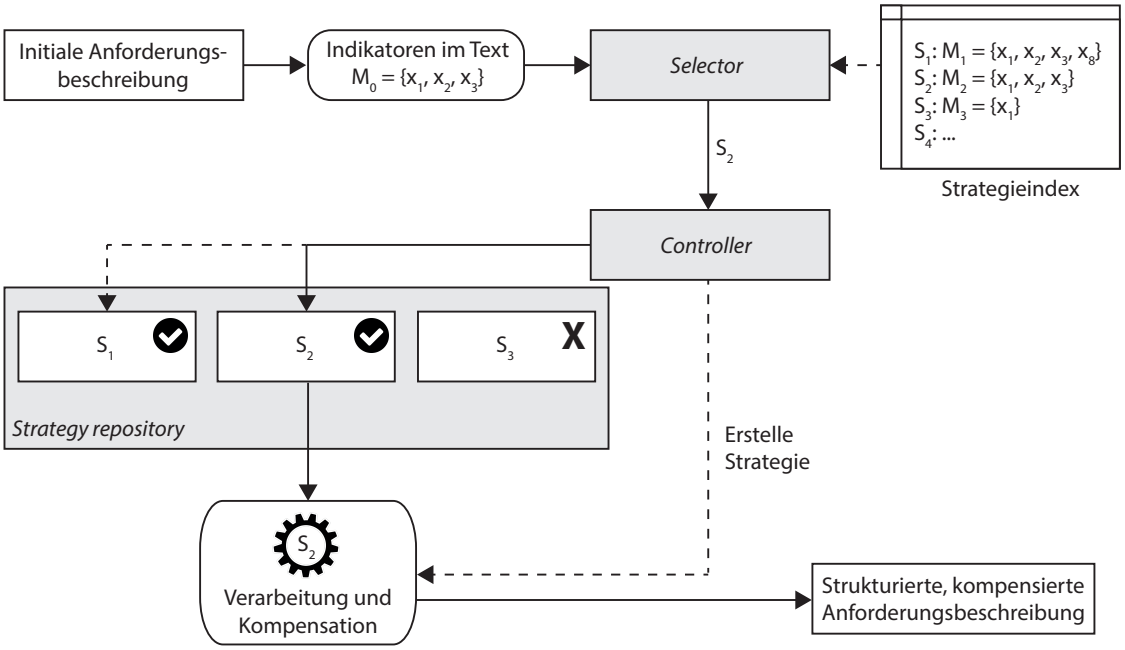


Abbildung 5.4: Selektion und Anwendung von Strategien auf Indikatorbasis

Im Mittelpunkt von Abbildung 5.4 stehen die Komponenten *Selector* und *Controller*. Die *Selector*-Komponente sucht nach Indikatoren (x_i) in Anforderungsbeschreibungen, die Hinweise auf die Beschreibungsqualität geben können und wählt daraufhin geeignete Strategien aus dem *Strategy repository* aus, sofern diese vorhanden sind. Geeignet sind Strategien dann, wenn alle erkannten Indikatoren ($M_0 = \{x_1, x_2, x_3\}$) abgedeckt sind (z. B. $M_0 = M_1$). Die Menge gefundener Indikatoren zeigt somit auf, welche Verarbeitungs- und Kompensationsschritte notwendig sind, um zum einen die Qualität der initialen Anforderungsbeschreibung als solche zu verbessern und zum anderen eine strukturierte Ausgabe zu ermöglichen, die maschinell weiterverarbeitet werden kann. Die *Controller*-Komponente überwacht gewählte Strategien und kann zum Beispiel auf Verarbeitungsfehler einzelner Verarbeitungskomponenten reagieren. Diese Fehler oder die Erkenntnis, dass ein erkannter Indikator nicht die Anwendung einer Strategie rechtfertigt (z. B. existieren ambige Lexeme, diese sind aber für die erkannten FA irrelevant), können den Strategiewechsel durch den *Controller* auslösen. Darüber hinaus ist der *Controller* berechtigt, Widersprüche in Verarbeitungsergebnissen einzelner Methoden durch den Grundsatz „*Expert first*“ aufzulösen. Dies bedeutet,

dass im Falle widersprüchlicher Ausgaben die Ausgaben der Expertenkomponente herangezogen werden, in der Annahme, dass diese korrekt sind.

Wie das Beispiel in Abbildung 5.4 zeigt, wird die Indikatormenge M_0 prinzipiell von zwei Strategien unterstützt (S_1, S_2), wobei es sich im Falle der Strategie S_2 um eine exakte Übereinstimmung der geforderten und der seitens der Strategie unterstützten Indikatoren handelt. Strategie S_2 ist demnach gegenüber Strategie S_1 zu bevorzugen, da nur Kompensationsschritte ausgeführt werden, die erforderlich sind.

Wird im *Strategy repository* keine geeignete Strategie gefunden, besteht die Möglichkeit, automatisch eine grundlegende Strategiekonfiguration zu erstellen. Hierbei ist grundsätzlich die Aufnahme weiterer sowie die Revidierung bestehender Konfigurationen vorgesehen. Motiviert wird dies zum einen durch eine angenommene Steigerung der Effizienz, da Strategien, die aus dem *Strategy repository* abgerufen werden können, schneller bereitstehen als jene, die erst zum Zeitpunkt der Verarbeitung und Kompensation konfiguriert werden müssen. Zum anderen können Strategien, die zum Entwicklungszeitpunkt bedarfsgerecht waren, gegebenenfalls unter veränderten Konditionen (z. B. Wechsel auf kontrollierte Sprache) obsolet werden.

Die Aufnahme neuer Strategiekonfigurationen in das *Strategy repository* erfolgt dabei automatisiert. Ein Kriterium für die Aufnahme in das *Repository* kann beispielsweise die häufige Ausführung gleicher Strategiekonfigurationen sein. Dies erfordert die Speicherung von Konfigurationsaufrufen sowie die Festlegung eines Grenzwerts, dessen Überschreitung die Aufnahme einer Strategiekonfiguration in das *Strategy repository* initiiert. Demgegenüber trägt das *Update* von Strategiekonfigurationen zum Wegfall von Strategien aus dem *Strategy repository* bei. Grundsätzlich ist dieser Schritt notwendig, da, wie dargestellt, die Aufnahme neuer Strategien vorgesehen ist und das *Strategy repository* ohne entsprechende Maßnahmen nur an Umfang zunehmen würde, obwohl existierende Strategien nach einiger Zeit nicht mehr benötigt werden. Dies hätte einen negativen Einfluss auf die Gesamtlaufzeit, da alle Strategien mit den Indikatoren einer Anforderungsbeschreibung abzugleichen sind.

In dieser Arbeit sind demnach Kompensationsstrategien vorkonfiguriert, die sich hinsichtlich Laufzeit und Abdeckung unterscheiden. Daneben existiert die Möglichkeit einer bedarfsgerechten, automatisch selbstkonfigurierenden Strategie. Strategien sind dabei in einen definierten Verarbeitungskontext eingebettet (vgl. Abbildung 5.5): Vor der Strategieanwendung wird ein *Preprocessing* (s. Abschnitt 5.5.2 und Anhang C.1) durchgeführt. Anschließend werden die Ergebnisse strukturiert gespeichert.

Abbildung 5.5: Strategieeinbettung in den Verarbeitungskontext

In den folgenden Abschnitten werden die einzelnen Strategiekonfigurationen dargestellt. Die einzelnen Strategien bauen aufeinander auf und werden daher hinsichtlich ihrer zunehmenden Komplexität erläutert. Eine Übersicht aller Strategiekonfigurationen findet sich in Abbildung 5.6. Sie gibt sowohl Auskunft über die Konfiguration als auch über die Laufzeit einzelner Strategien.

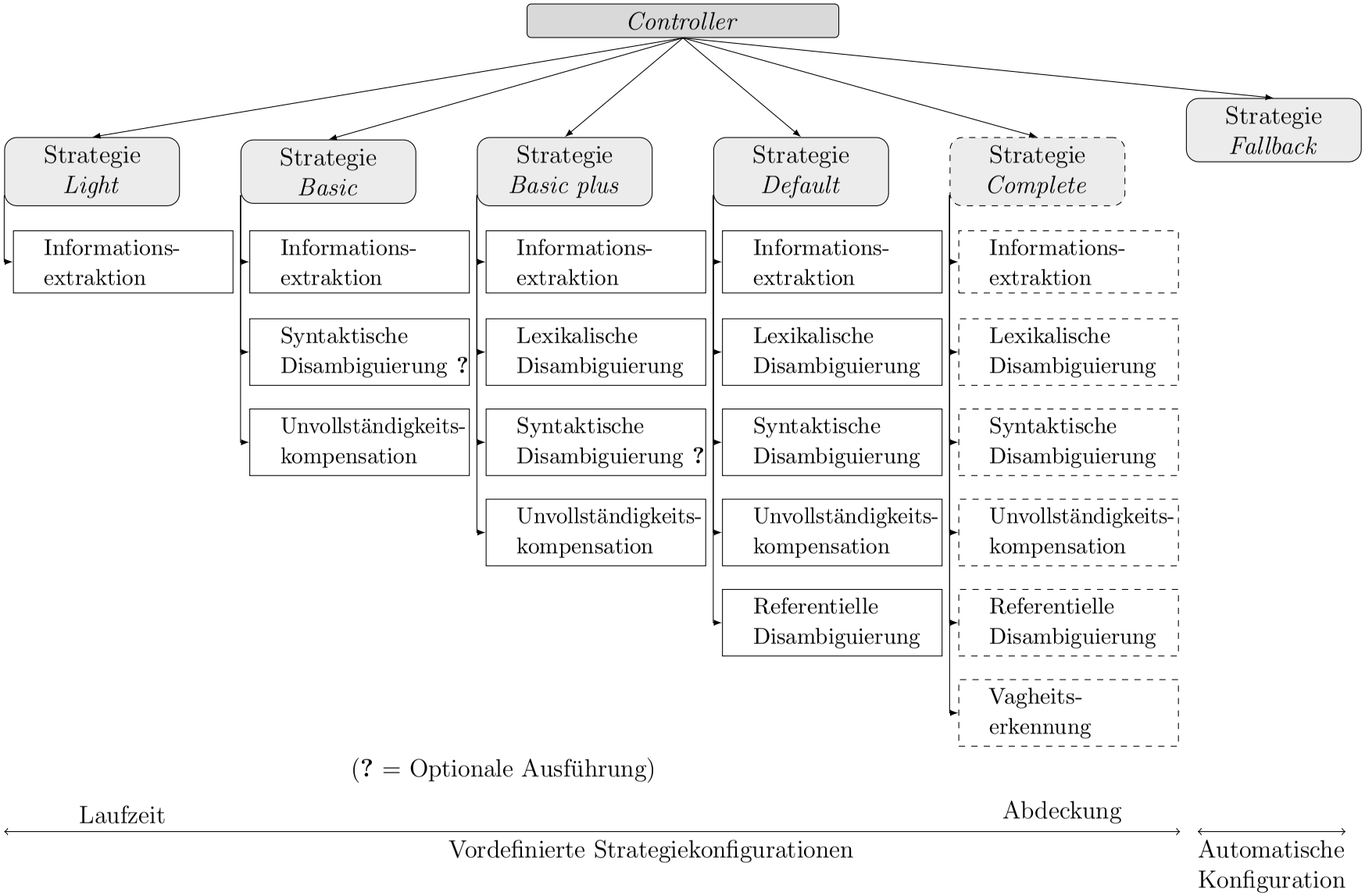


Abbildung 5.6: Strategiekonfigurationen

5.2.1 Light-Strategie

Abbildung 5.7: *Light*-Strategie

Bei der *Light*-Strategie handelt es sich um eine reine Anforderungsextraktion, deren Ziel es ist, FA ohne weitere Verarbeitungs- und Kompensationsschritte in Anforderungsbeschreibungen zu identifizieren, klassifizieren und zu strukturieren. Voraussetzung für die Anwendung der Strategie ist, dass keine Indikatoren in der Anforderungsbeschreibung vorliegen, die die Hinzunahme von Kompensationsschritten begründen. Sie führt daher nur absolut notwendige Verarbeitungsschritte aus und ist damit die Strategie mit der kürzesten Ausführungszeit und der geringsten Kompensation (vgl. Abbildungen 5.6 sowie 5.7).

Wie in Abbildung 5.7 zu erkennen ist, wird das *Preprocessing* vor der Strategieanwendung ausgeführt. Daher kann die IE auf Informationen, die im *Preprocessing* gewonnen wurden, zurückgreifen. Die identifizierte Sprache eines Satzes und dessen Relevanz sind dabei besonders hervorzuheben, da sie dabei helfen, die Strategie sehr gezielt anzuwenden: Nebensächliche und nicht-englische Sätze werden keiner Extraktion unterzogen.

Anwendungsfall

Tabelle 5.1 zeigt die strukturierte Ergebnisausgabe für Beispiel 5.2.1, in dessen Mittelpunkt das Prädikat „*save*“ steht. Dieses wurde durch die IE korrekt erkannt und als Aktion annotiert.

Beispiel 5.2.1
„I want to save$_{Aktion}$ unknown email addresses.“

Auch weitere semantische Informationen, wie die Rolle oder das Objekt einer FA, werden erkannt und strukturiert gespeichert. Die extrahierten semantischen Informationen sind von hoher Relevanz für komplexere Strategien.

Lemma	POS	Sem. Info.	Bemerkung
i	PRON	**Rolle**	Wer will oder macht etwas?
want	VERB	**Priorität**	Welche Priorität hat das Verlangte?
to	PART	_	
save	VERB	**Aktion**	Welche Aktion ist gefordert (Prädikat)?
unknown	ADJ		
email	NOUN	**Objekt**	Welches Objekt ist betroffen?
address	NOUN		

Tabelle 5.1: Beispielhafte Ergebnisausgabe der IE (*Light*-Strategie)

5.2.2 Basic-Strategie

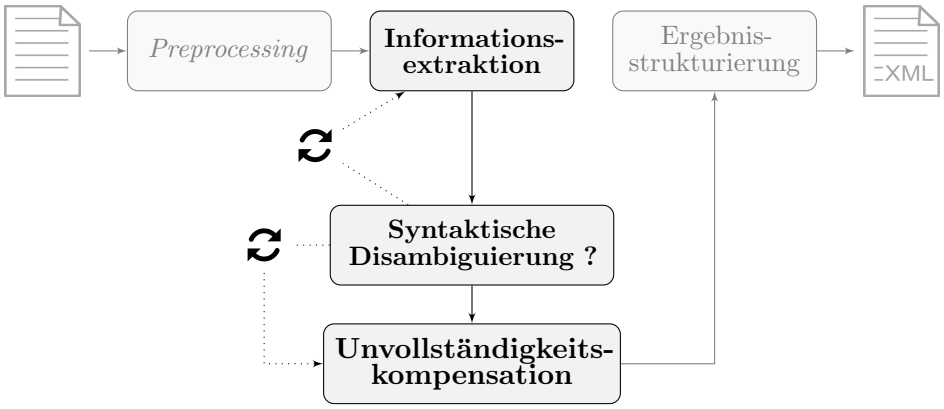


Abbildung 5.8: *Basic*-Strategie

Die *Basic*-Strategie erweitert die zuvor vorgestelle Strategie *Light* um die Kompensationsschritte syntaktische Disambiguierung (s. Abschnitte 2.1.2 und 5.5.4) und Unvollständigkeitskompensation (s. Abschnitte 2.3 und 5.5.5). Ziel ist es, FA zu extrahieren, strukturelle Ambiguität aufzulösen, Unvollständigkeit zu kompensieren und die Ergebnisse der Einzelmethoden miteinander in Einklang zu bringen.

Auf die IE folgend, werden relevante, englischsprachige Sätze zuerst disambiguiert. Eine Besonderheit der syntaktischen Disambiguierung ist dabei, dass zunächst überprüft wird, ob überhaupt Hinweise für strukturelle Ambiguität vorliegen, ansonsten wird der Satz ohne weitere Analysen übersprungen. Diese optionale Ausführung ist in Abbildung 5.8 mit einem Fragezeichen gekennzeichnet.

Auf die Disambiguierung folgt die Kompensation unvollständiger Satzaussagen. Prädikate, die in der IE als relevant für eine FA identifiziert worden sind, werden auf die Vollständigkeit ihrer Argumente hin überprüft und – sofern notwendig – komplementiert. In der *Basic*-Strategie steht die Disambiguierung damit im Zentrum, welche den Ergebnisabgleich einzelner Methoden koordiniert und gegebenenfalls notwendige Korrekturen vornimmt (🔄). So wird zum einen das Ergebnis der IE hinsichtlich fehlerhafter syntaktischer Zuordnung untersucht und zum anderen die erkannten, fehlenden Argumente auf syntaktische Korrektheit geprüft. Demonstriert wird dies im Folgenden an Beispiel 5.2.1, welches um eine PP erweitert wird.

Anwendungsfall

Die Methode der IE erkennt sowohl das Prädikat „*save*“ als Aktion einer funktionalen Anforderung als auch die PP „*in a new contact group*“ als Verfeinerung der erkannten Aktion. Die Ambiguitätsprüfung der syntaktischen Disambiguierungsmethode erkennt potentielle Ambiguität durch PP-Anbindung und gibt die Disambiguierung frei.

Beispiel 5.2.2
„I want to $\boldsymbol{save}_{Aktion}$ *unknown email addresses in a new contact group“.*

Das Ergebnis der syntaktischen Disambiguierung ist in Abbildung 5.9 dargestellt und spiegelt das Ergebnis der IE wider. Auch hier wurde die PP dem Prädikat „*save*“

zugeordnet und nicht etwa der NP „*unknown email addresses*“. Eine Korrektur der extrahierten funktionalen Anforderung ist möglich (↻), in diesem Fall aber aufgrund der übereinstimmenden Ergebnisse nicht notwendig.

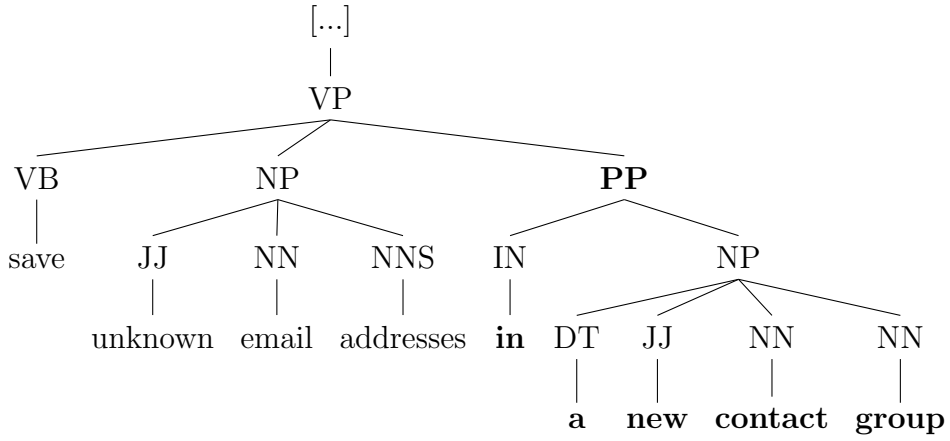


Abbildung 5.9: Ergebnis der syntaktischen Disambiguierung

Die Kompensationsmethode extrahiert Argumente des Prädikats aus dem Satzkontext, um eine Kompensationsanfrage zu erstellen (s. Abschnitt 5.5.5). In diesem Fall betrifft dies das Prädikat „*save*“ in seiner semantischen Funktion als Aktion. Dafür werden folgende Argumente zurückgeliefert:

- (Arg_0) **collector** „*I*“
- (Arg_1) **thing saved** „*unknown email addresses in a new contact group*“

Da die Kompensation der Unvollständigkeit als eigenständige Methode unabhängig von der syntaktischen Disambiguierung arbeitet, können durch die Kompensation syntaktische Ambiguitäten auftreten und zu fehlerhaften Ergebnissen führen. Aus diesem Grund sieht die Strategie auch einen Ergebnisabgleich und -anpassung der beiden Kompensationsschritte vor (↻). Hierbei gilt das *Expert first*-Prinzip. Divergieren demnach die Ergebnisse aufgrund syntaktischer Ambiguität, wird das Ergebnis der Expertenkomponenten zur Disambiguierung übernommen, da davon ausgegangen wird, dass dieses Ergebnis korrekt ist.

In der Tat wurde seitens der Methode zur Unvollständigkeitskompensation das erste Argument des Prädikats „*save*“ falsch erkannt, da die PP an die NP gebunden wurde. Ein Umstand, der von der Methode zur syntaktischen Disambiguierung erkannt und korrigiert wird. Durch die Korrektur der Argumentanbindung kann die Kompensationskomponente eine korrekte Kompensationsanfrage stellen und erzeugt so in Fällen, in denen Unvollständigkeit vorliegt, ein in den Satzkontext passendes Kompensationsergebnis.

5.2.3 Basic Plus-Strategie

Die *Basic Plus*-Strategie stellt eine Erweiterung der *Basic*-Strategie um die Methode der lexikalischen Disambiguierung dar (s. Abschnitte 2.1.1 und 5.5.4). Wie

in Abbildung 5.10 zu sehen ist, reiht sich die lexikalische Disambiguierung vor der syntaktischen Disambiguierung ein. Ziel dieser Ergänzung ist die Erweiterung sowie Prüfung und Korrektur der IE-Ergebnisse auf lexikalischer Basis (🔄). Die Aufgaben der lexikalischen Disambiguierung sind somit vielfältig.

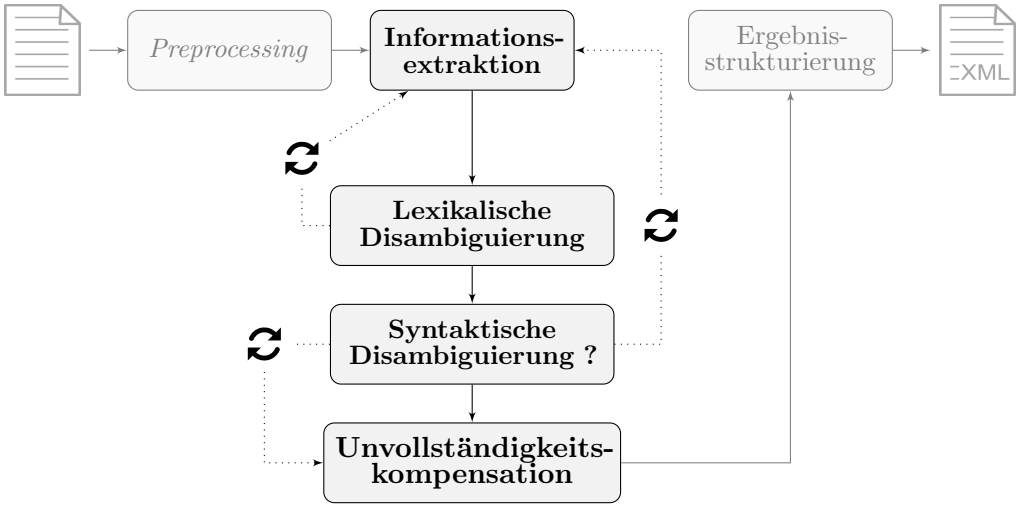


Abbildung 5.10: *Basic Plus*-Strategie

Die Kernaufgabe der lexikalischen Disambiguierung ist die Bestimmung der wahrscheinlichsten Lesart für ein Lexem aufgrund seiner Einbettung in den Kontext (s. Abschnitt 2.1.1), wobei nur die Lexeme disambiguiert werden, denen zuvor durch die IE eine semantische Funktion in der FA zugeschrieben wurde. Darüber hinaus ist die Methode zur lexikalischen Disambiguierung in der Lage, die Ergebnisse der IE zu modifizieren. Diese Korrekturfunktion basiert im Wesentlichen auf erkannten POS-*Tags* sowie semantischen Informationen. Stehen diesbezüglich die Ergebnisse der IE und der lexikalischen Disambiguierung im Widerspruch, kann eine Korrektur erfolgen. Wie wichtig dieser Ergebnisaustausch sein kann, wird im Folgenden an konkreten Beispielen aufgezeigt.

Anwendungsfall

Beispiel 5.2.3 enthält eine Anforderungsbeschreibung, die von der IE verarbeitet wurde (vgl. Tabelle 5.2). Die semantischen Informationen wurden korrekt zugeordnet (z. B. „*sort*“ als Aktion). Allerdings wurde dabei bisher nicht berücksichtigt, dass einzelne Lexeme mehrere Lesarten haben können. Dies ist beispielsweise bei „*sort*“ (sechs Lesarten) sowie bei „*folders*“ (zwei Lesarten) der Fall.

Beispiel 5.2.3
„I want to $sort_{Aktion}$ *emails into separate folders“.*

Durch die lexikalische Disambiguierung kann das Ergebnis der IE somit um wertvolle Informationen ergänzt werden. Im Falle von „*sort*“ handelt es sich bei vier der sechs Lesarten um Nomina, die *per se* keine Aktion darstellen können und ausgeschlossen werden. Im nächsten Schritt entscheidet sich die Methode zur Disambiguierung nicht

für die Lesart[73] „*examine in order to test suitability*“ sondern für „*order by classes or categories*“. Dies ist eine korrekte Zuordnung, welche die Ambiguität der Aussage weiter minimiert. Tabelle 5.2 zeigt erneut Beispiel 5.2.3, ergänzt um die Beschreibung der disambiguierten Lesart.

#	Lemma	Sem. Info.	Disambiguierung
1	i	**Rolle**	
2	want	**Priorität**	*Have a desire for; want strongly*
3	to		
4	sort	**Aktion**	*order by classes or categories*
5	email	**Objekt**	*World-wide electronic communication*
6	into	**Verfeinerung**	
7	separate	**Verfeinerung**	*Independent; not united or joint*
8	folder	**Verfeinerung**	*covering that is folded over [...]*

Tabelle 5.2: Beispielhafte Ausgabe der IE, ergänzt um Disambiguierung

Neben der Disambiguierung von Lesarten ist bisher davon ausgegangen worden, dass die Zuordnung von Wortarten (POS) durch die IE-Methode korrekt durchgeführt wurde. Beispiel 5.2.4 hingegen enthält einen Satz, der durch die IE fehlerhaft verarbeitet wird (vgl. Tabelle 5.3).

Beispiel 5.2.4
„I want to automatically sort sharepoint emails into separate mail folders“.

Die Methode zur Informationsextraktion annotiert das Wort „*automatically*“ korrekt als Adverb (ADV) und dennoch findet eine semantische Zuordnung als Aktion statt. Dies führt in der Weiterverarbeitung zu Problemen, da „*automatically*“ nun als eigenes Prozesswort bzw. Aktion geführt wird. In solchen Fällen greift die lexikalische Disambiguierung korrigierend ein.

#	Lemma	POS-U	Sem. Info.
1	i	PRON	Rolle
2	want	VERB	Priorität
3	to	PART	_
4	**automatically**	**ADV**	**Aktion**
5	sort	VERB	Aktion
6	sharepoint	NOUN	Objekt
7	email	NOUN	Objekt
	[...]		

Tabelle 5.3: Ausgabe der IE mit fehlerhafter Aktionsangabe (*Basic Plus*)

So ergibt die Disambiguierung, dass es sich bei „*automatically*“ in der Tat um ein ADV handelt, welches aber isoliert keine Aktion darstellt. Aus diesem Grund wird die semantische Annotation „Aktion“ entfernt und durch „Verfeinerung“ ersetzt[74].

[73] Siehe weiterführend: `http://www.babelfy.org` (Stand: 12.03.17).

[74] Der Begriff der Verfeinerung geht auf die Arbeit von Dollmann (2016) zurück, der ergänzende Informationen (z. B. Modifikatoren) innerhalb einer FA als Verfeinerungen bezeichnet.

Beispiel 5.2.5
„The system should group$_{Aktion}$ emails.“

Beispiel 5.2.5 zeigt eine weitere minimalistische Anforderungsbeschreibung, die einen Fehler durch die IE enthält. Das Wort „*group*“ wird korrekt als Aktion erkannt, allerdings wurde das falsche POS-*Tag* zugeordnet: Die Informationsextraktionskomponente führt „*group*“ als Nomen. Jedoch erkennt die Methode zur lexikalischen Disambiguierung „*group*“ in diesem Kontext in der Lesart als Verb („*Arrange into a group or groups*“) und ersetzt gemäß der Devise *Expert first* die fehlerhafte Wortart.

5.2.4 Default-Strategie

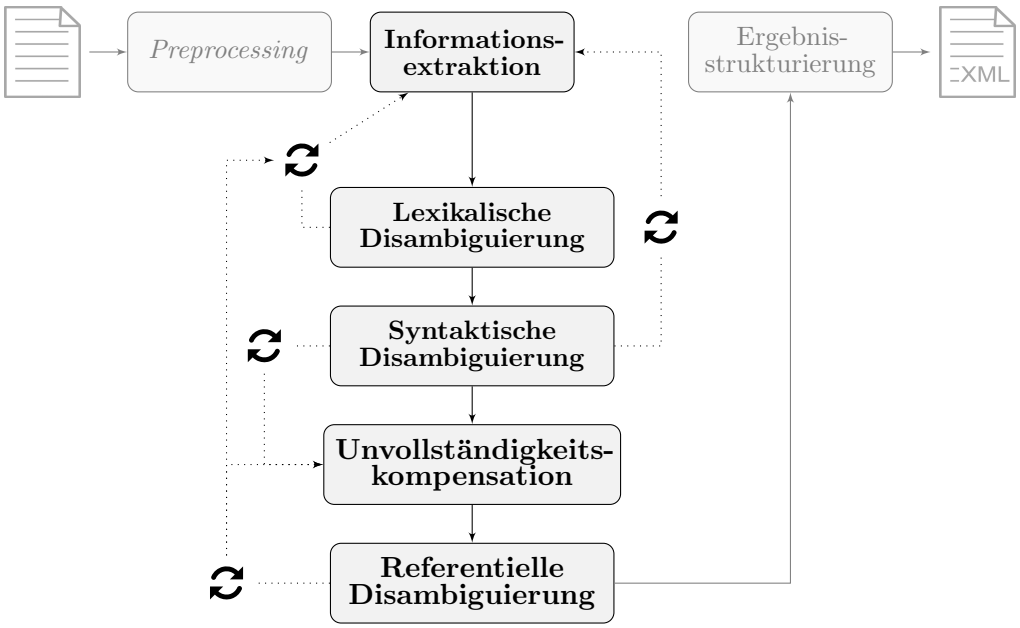


Abbildung 5.11: *Default*-Strategie

Die *Default*-Strategie führt erstmalig auch die Methode zur referentiellen Disambiguierung aus und erweitert damit die zuvor vorgestellte *Basic Plus*-Strategie um einen weiteren Verarbeitungs- und Kompensationsschritt. Die referentielle Disambiguierung hat dabei zum einen die Aufgabe, ambige Referenzen in den Anforderungsbeschreibungen aufzulösen und kann zum anderen Koreferenzketten bilden, um darüber die FA einander zuordnen zu können. Sie ist auch in der Lage, sowohl die Ergebnisse der IE als auch die Kompensationsanfrage der Unvollständigkeitskompensation zu modifizieren. Die Methode zur referentiellen Disambiguierung arbeitet dabei nicht iterativ auf Satzbasis, sondern betrachtet die gesamte Anforderungsbeschreibung, um wiederkehrende Referenzausdrücke satzübergreifend zu erkennen.

Anwendungsfall

Die Anforderungsbeschreibung in Beispiel 5.2.6 besteht aus zwei Sätzen und beschreibt ausgewählte Anforderungen an eine E-Mail-Applikation.

Beispiel 5.2.6
*„**I** want to move$_{Aktion}$ **email spam** and **I** want to delete$_{Aktion}$ the **spam**.*
*The system should report$_{Aktion}$ the **spam** to the administrator.“*

In diesem Beispiel erkennt die Methode in der Anforderungsbeschreibung die beiden Koreferenzketten (K) [*email spam*, *the spam*, *the spam*]$_0$ und [*I*, *I*]$_1$. Während alle Referenzausdrücke in K_0 als semantische Funktion Objekte in den FA darstellen, handelt es sich bei den Referenzausdrücken in K_1 um Rollen. Für die weitere Interpretation der Beschreibung ist es hilfreich zu wissen, welche Rollen in der gesamten Anforderungsbeschreibung wiederholt vorkommen und dass die drei Aktionen auf ein gemeinsames Objekt verweisen, auf welches mittels verschiedener Ausdrücke referenziert wird.

Darüber hinaus ermöglicht es die Methode, die Komplexität von FA zu verringern. Ohne eine referentielle Disambiguierung würde in Beispiel 5.2.7 nach der IE unklar bleiben, was genau verschickt („*send*“) werden soll. Insbesondere, da „*them*“ korrekt als Objekt identifiziert wurde, losgelöst von der ersten Anforderung aber nicht interpretiert werden kann.

Beispiel 5.2.7
*„**I** want to write$_{Aktion}$ **emails** and **I** want to send$_{Aktion}$ **them**.“*

Durch die referentielle Disambiguierung wird deutlich, dass „*them*“ auf das zuvor eingeführte Objekt „*emails*“ referenziert, was in den Ergebnissen vermerkt und bei der Ausgabe berücksichtigt wird. Diese Erkenntnis ist auch bei der Kompensation von Unvollständigkeit von Bedeutung, führt doch die Kompensationsanfrage für das Prädikat „*send*“ viel wahrscheinlicher zu einem passenden Resultat, wenn das berücksichtigte Argument Arg_1 „*emails*“ und nicht „*them*“ lautet.

5.2.5 Complete-Strategie

Die *Complete*-Strategie stellt die letzte vordefinierte Strategie dar. Sie verfolgt das Ziel einer möglichst großen Methodenabdeckung und enthält somit alle Methoden, die bereits in der *Default*-Strategie Anwendung finden (vgl. Abbildung 5.12). Dennoch unterscheidet sie sich in drei Merkmalen: Erstens wird die *Complete*-Strategie um eine Methode zur Erkennung von Vagheit ergänzt. Vagheit ist oftmals für Ungenauigkeit in Anforderungsbeschreibungen verantwortlich (s. Abschnitt 2.2). Zwar ist Vagheit kein zentrales Thema dieser Arbeit, aber aufgrund der hohen Praxisrelevanz und der angestrebten hohen Abdeckung sollte sie für Endanwender sichtbar sein. Demnach kann diese Strategie potentielle Vagheit erkennen und aufzeigen.

Zweitens wird die syntaktische Disambiguierung nicht selektiv (vgl. Abbildung 5.6) sondern auf allen Sätzen einer Anforderungsbeschreibung angewandt. Dies bedeutet, dass keine erneute Prüfung stattfindet, ob ein konkreter Satz disambiguierungsbedürftig ist, sondern alle Sätze syntaktisch disambiguiert werden, sofern zuvor die Notwendigkeit einmalig erkannt wurde. Dies kann unter dem Gesichtspunkt der Laufzeit zu einer Verschlechterung führen, da die syntaktische Disambiguierung rechenintensiv ist (Carstensen et al., 2010, S. 312). Unter dem Ziel einer höchstmöglichen Abdeckung allerdings sind diese zusätzlich gewonnenen linguistischen Informationen

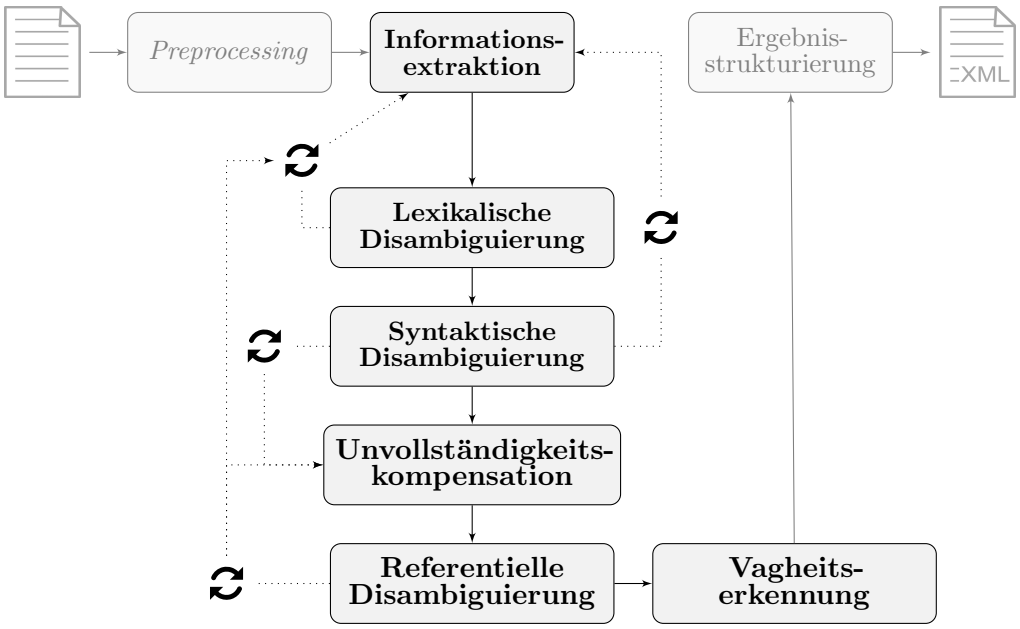


Abbildung 5.12: *Complete*-Strategie

für Folgekomponenten wertvoll, da sie Aufschluss über strukturelle Abhängigkeiten zwischen einzelnen semantischen Informationen einer FA geben (z. B. Modifikatoren).

Drittens ist die strukturierte Ausgabe dieser Strategie erweitert um linguistische Informationen und Methodenprotokolle, die über die notwendigen Angaben hinausgehen (z. B. alternative Lesarten, Kandidaten einzelner Koreferenzketten). Auch dies kann zu einer Verlängerung der Laufzeit führen und verfolgt das Ziel, vorhandene Informationen, die im Verarbeitungsprozess anfallen aber nicht berücksichtigt wurden, im Sinne einer hohen Informationsdichte für Folgekomponenten aufbereitet zur Verfügung zu stellen. Im Folgenden wird sowohl ein Beispiel für Vagheitserkennung als auch für die erweiterte Ergebnisstrukturierung gegeben.

Anwendungsfall

Der in Beispiel 5.2.8 dargestellte Satz einer Anforderungsbeschreibung wird mittels IE verarbeitet. Dabei ist das Adjektiv „*large*“ als Bestandteil des Objekts annotiert worden und wurde bisher nicht durch weitere Kompensationsmethoden verarbeitet.

Beispiel 5.2.8
„The system must be able to send$_{Aktion}$ *large emails*$_{Objekt}$*.“*

Die Vagheitserkennung untersucht die, durch die IE erkannten, semantischen Bestandteile der FA und speichert Hinweise auf potentielle Vagheit zur späteren Ausgabe an der Benutzerschnittstelle. Angewendet auf das Beispiel 5.2.8, erkennt die Methode (s. Abschnitt 5.5.6) das steigerbare Adjektiv „*large*“ als potentiell vage (Löbner, 2003, S. 63) und vermerkt dies zur späteren Ausgabe. Eine Vagheitskompensation findet im Rahmen dieser Arbeit nicht statt (s. Kapitel 2).

Die erweiterte Ergebnisausgabe unterstützt alle Methoden, so beispielsweise auch die lexikalische Disambiguierung: Während bisher die gewählte Lesart und gegebenenfalls die korrigierte Wortart zurückgegeben wurden, werden bei der erweiterten Ausgabe auch potentielle Kandidaten für die Lesart eines Lexems unter Angabe von Wahrscheinlichkeitswerten ausgegeben. Dies befähigt Folgekomponenten eigenständige Korrekturen ohne Hinzunahme weiterer Ressourcen vorzunehmen. Darüber hinaus werden Metainformationen zu einbezogenen Ressourcen (z. B. Name, Version) mit ausgegeben, die sich vor allem an Entwickler von Folgekomponenten richten.

5.2.6 Fallback-Strategie

Wird keine bestehende Strategie der geforderten Indikatorkombination gerecht, greift die *Fallback*-Strategie, die ausgehend von einer reinen Anforderungsextraktion flexibel in der Methodenkonfiguration ist. Ziel ist es, nur notwendige Methoden auszuführen. Hinsichtlich möglicher Abhängigkeiten sowie potentieller Synergien zwischen den Methoden sind allerdings Qualitätseinbußen hinzunehmen, da dies bei der automatischen Konfiguration nicht so umfassend erfolgen kann, wie es bei den vorkonfigurierten Strategien der Fall ist. Da allerdings der *Controller* befähigt ist, weitere Methoden miteinzubeziehen, deren Notwendigkeit über Indikatoren allein nicht ermittelt werden konnte, kann sich der Strategieablauf auch während der Laufzeit anpassen. Nichtsdestotrotz ist die *Fallback*-Strategie, wie der Name bereits verdeutlicht, derzeit als eine Rückfallstrategie zu begreifen, die gewählt wird, wenn eine Indikatorkombination durch bestehende Strategien nicht abgedeckt werden kann. In der Weiterentwicklung des resultierenden Softwaresystems kann angestrebt werden, einzig und allein auf eine automatische Strategie zurückzugreifen. Dies setzt allerdings voraus, dass genügend reale Anforderungsbeschreibungen vorliegen, um mehr über Qualitätsmerkmale und sonstige Defizite in den Text zu erfahren. Darüber hinaus gilt es das Zusammenwirken, Verhalten und die Ergebnisqualität der einzelnen Komponenten besser zu verstehen, was die derzeit vorgenommene Strategieaufteilung übersichtlich ermöglicht.

5.3 Indikatoren der Strategieauswahl

Auf die zuvor beschriebenen Kompensationsstrategien zurückzugreifen ermöglicht es, effizient und flexibel auf die Beschaffenheit einer Anforderungsbeschreibung zu reagieren. Allerdings werfen gerade Flexibilität und unterschiedliche Konfigurationsvarianten der Strategien das Problem der Entscheidungsfindung auf: Welche Strategie ist die Beste für eine konkrete Anforderungsbeschreibung? Die Antwort auf diese Frage ist dabei ohne die Kompensationsmethoden zu finden, die erst nach der Strategiewahl aktiviert werden. Darüber hinaus muss die Strategieentscheidung mit minimalem Zeitaufwand erfolgen.

5.3.1 Begriffsdefinition von Indikatoren

In Abschnitt 5.2 wird bereits der Begriff der „Indikatoren“ eingeführt, die innerhalb einer Beschreibung auftreten können und die die Beschaffenheit ausmachen. Femmer (2013) gibt in diesem Zusammenhang zu bedenken, dass es einfacher ist,

qualitätsmindernde Indikatoren zu finden[75], als welche, die für Qualität stehen, da erstere oftmals konkrete „Spuren“ im Text hinterlassen (Femmer, 2013, S. 1). In diesem Zusammenhang wird auch von „*Requirements smells*“ gesprochen (Femmer et al., 2016a; Femmer et al., 2016b; Femmer et al., 2014; Femmer, 2013).

Definition 5.3.1 (*Requirements smell*)
„A Requirements smell is an indicator of a quality violation, which may lead to a defect, with a concrete location and a concrete detection mechanism.“
(Femmer et al., 2016a, S. 8)

Femmer et al. (2016a) orientieren sich dabei an „*Code smells*“, die als Indikatoren für schlechten Quelltext stehen (nach Fowler et al., 1999). In vier Punkten werden „*Requirements smells*“ von Femmer et al. (2016a, S. 8) genauer definiert und im Folgenden in enger Anlehnung auf die vorliegende Arbeit übertragen:

1. „*Requirements smell*“ ist ein Indikator für eine Qualitätsverletzung eines Anforderungsartefakts. Für diese Definition verstehen wir Anforderungsqualität im Sinne von „*quality-in-use*“, was bedeutet, dass sich schlechte Anforderungsqualität durch die (potentiellen) negativen Auswirkungen auf Aktivitäten im anforderungsbasierten Softwarelebenszyklus manifestiert.

2. „*Requirements smell*“ führt nicht zwingend zu einer Fehlfunktion und ist im jeweiligen Anwendungskontext zu bewerten [...]. Ob „*Requirements smell*“ im jeweiligen Kontext ein Problem darstellt oder nicht, muss individuell entschieden werden und bedarf somit *Reviews* und weiterer Qualitätssicherungsaktivitäten.

3. Ein „*Requirements smell*“ hat eine konkrete Position in einer Anforderungsbeschreibung, z. B. ein Wort oder eine Sequenz. „*Requirements smells*“ sind immer mit einer Positionsangabe ausgestattet, welche die potentielle Fehlerstelle kennzeichnet. Dies ist ein Unterschied zu allgemeinen Qualitätsmerkmalen wie Vollständigkeit, was nur ein abstraktes Kriterium ist.

4. „*Requirements smells*“ ermöglichen spezifische Erkennungsmechanismen, die mehr oder weniger akkurat in der Erkennung sein können.

Auf Grundlage von Definition 5.3.1 sowie der darauf folgenden Begriffsverfeinerung wird im Rahmen dieser Arbeit der Begriff des Indikators wie folgt definiert:

Definition 5.3.2 (Indikator)
Ein Indikator zeigt Qualitätsverletzungen in Anforderungsbeschreibungen auf, die für die Interpretation von Anforderungen sowie die softwaretechnische Umsetzung potentiell schädlich sind. Indikatoren treten an mindestens einer Textposition auf und sind mindestens einem Erkennungs- sowie Kompensationsmechanismus zugeordnet.

Da Indikatoren mehr oder weniger akkurat sein können, rechtfertigen sie die Ausführung einer Methode und Strategie nicht in allen erkannten Fällen. So kann

[75] Femmer (2013) bezieht sich dabei auf Anforderungsartefakte.

ein Indikator, der sehr zuverlässig erkannt werden kann und damit einen sehr konkreten Verdacht auf eine Qualitätsverletzung anzeigt, bereits die Anwendung von Kompensationsmaßnahmen rechtfertigen. Demgegenüber kann ein ungenauerer und unzuverlässigerer Indikator gegebenenfalls nur einen Anfangsverdacht auf eine Qualitätsverletzung begründen.

5.3.2 Bestimmung kontextsensitiver Indikatoren

In diesem Abschnitt werden die Indikatoren, welche die Ausführung einzelner Methoden und damit auch einzelner Strategien rechtfertigen, dargestellt. Es wird von kontextsensitiven Indikatoren gesprochen, da nicht einzelne Lexeme einen Indikator bilden, sondern erst der Kontext bzw. bestimmte Textmuster genug Aussagekraft erzeugen, um als Indikatoren (zuverlässig) zu fungieren. Indikatoren können dabei auf Satzbasis oder auf der gesamten Anforderungsbeschreibung angewendet werden. Sie müssen aber losgelöst von den Erkennungs- und Kompensationsmethoden arbeiten, da diese ausschließlich im nachgelagerten Schritt, demnach bei entsprechendem Bedarf (erkannte Indikatoren) herangezogen werden.

5.3.2.1 Lexikalische Ambiguität als Indikator

Die lexikalische Disambiguierung verfolgt das Ziel, einem Lexem seine korrekte Lesart aus einer Menge von Lesarten zuzuordnen (s. Abschnitt 2.1.1). Ein Indikator hierfür ist Ambiguität, also das Vorhandensein mindestens zweier potentieller Lesarten für ein und dasselbe Lexem. Hierbei stellt sich nun die Frage, ob es wirklich notwendig ist, alle Lexeme eines Satzes oder einer Anforderungsbeschreibung auf lexikalische Ambiguität zu überprüfen. Und darüber hinaus, ob nicht Einschränkungen existieren, die die Menge an potentiellen Lesarten von vornherein minimieren. Diesbezüglich werden folgende Annahmen bereits vor der Indikatorbestimmung getroffen:

Zur Steigerung der Effizienz gelten nur Lexeme als disambiguierungsbedürftig, die sich in *On-Topic*-Sätzen befinden und eine semantische Funktion innerhalb einer FA einnehmen (s. Abschnitt 5.5.3). Letzteres wird eingegrenzt, indem nur solche Lexeme disambiguiert werden, die nicht den semantischen Kategorien „Rolle“ oder „Priorität“[76] zuzuordnen sind. Hier wird die Variabilität in der Wortwahl als so gering angenommen, dass eine Disambiguierung nicht notwendig erscheint. Schlussendlich werden nur Lexeme berücksichtigt, die keine Stoppwörter sind.

Beispiel 5.3.1 zeigt den Satz „*I want to send emails to my family*“, auf dem ein Indikator für lexikalische Ambiguität identifiziert wird.

Beispiel 5.3.1 (Indikatorbestimmung für lexikalische Ambiguität)

(5)				*Aktion*	*Objekt*			*Verfein.*
(4)				*VB*	*NNS*		~~*PRP$*~~	*NN*
(3)	~~*I*~~	~~*want*~~	~~*to*~~	*send*	*emails*	~~*to*~~	*my*	*family*
(2)	*Rolle*	*Priorität*		*Aktion*	*Objekt*		*Verfein.*	*Verfein.*
(1)	*I*	*want*	*to*	*send*	*emails*	*to*	*my*	*family*

[76] Es werden weitere Kategorien wie „Subpriorität“ etc. gefiltert, vgl. Abbildung 5.18.

In Schritt (1) ist der Originalsatz zu sehen. Dieser wird in Schritt (2) um semantische Informationen durch die IE erweitert. Stoppwörter, die darüber hinaus keine semantische Funktion innerhalb der FA haben (in diesem Fall „*to*“), werden in Schritt (3) zusammen mit Lexemen der Funktionen „Rolle“ und „Priorität“ entfernt. Als Stoppwortliste eignet sich beispielsweise die von der *Apache Foundation* zur Verfügung gestellte Liste[77]. Anschließend werden in Schritt (4) die POS-*Tags* der verbleibenden Lexeme annotiert. Hier werden weitere Stoppwörter, auch auf Basis der *Tags*, entfernt, die bislang aufgrund ihrer Zugehörigkeit zu semantischen Kategorien unangetastet blieben. Schritt (5) zeigt die verbleibenden Lexeme, die daraufhin als Kandidaten für eine Disambiguierung in Frage kommen: „*send*“, „*emails*“ und „*family*“.

Lexem	**Lesart**	**Beschreibung**
send	send.01	*cause to go somewhere*
	send.02	*to cause or order to be taken, directed,...*
	send.03	*cause to be directed or transmitted to another place*
	...	
	send.08	*broadcast over the airwaves, as in radio or television*
email	email.01	*a system of world-wide electronic communication*
family	family.01	*a social unit living together*
	family.02	*primary social group; parents and children*
	family.03	*a collection of things sharing a common attribute*
	...	
	family.08	*an association of people who share common beliefs...*

Tabelle 5.4: Potentielle Lesarten verbleibender Disambiguierungskandidaten

Auf diese Vorauswahl von Kandidaten folgt ein Abgleich mit WordNet, was Informationen über die Anzahl möglicher Lesarten enthält (s. Abschnitt 3.3.1.1). Die Anfragen an WordNet bezüglich möglicher Lesarten der drei Kandidaten können durch Hinzunahme der bereits bekannten POS-*Tags* besser spezifiziert werden. Dies erhöht die Aussagekraft erheblich, sind doch nur die Lesarten von Interesse, die auch die Wortart des Kandidaten teilen (s. Auszug aus dem Anfrageergebnis in Tabelle 5.4). Die Lexeme „*send*“[78] (als Verb) und „*family*“[79] (als Nomen) haben laut WordNet jeweils acht verschiedene Lesarten. Hingegen hat „*emails*“[80] nur eine einzige Lesart und fällt daher als Kandidat weg. Es verbleiben zwei Kandidaten für potentielle lexikalische Ambiguität. Das bisherige Ergebnis dieses Indikatorchecks ist demnach, dass bei zwei von acht Lexemen Ambiguität vorherrschen kann.

Es stellt sich jedoch zum einen die Frage, ob wirklich alle gefundenen Lesarten relevant sind, oder ob nicht eine weitere Einschränkung erfolgen muss. Zum anderen gilt es zu klären, wie die Indikatoren nun im Kontext der Anforderungsbeschreibung zu bewerten sind: Reicht der Verdacht auf Ambiguität bei zwei Lexemen aus, um den spezifischen Satz der Anforderungsbeschreibung in Gänze einer lexikalischen Disambiguierung zu unterziehen? Da die zugrundeliegenden Kandidaten bereits vorgefiltert

[77] Siehe: `http://snowball.tartarus.org/algorithms/english/stop.txt` (Stand: 16.02.17).
[78] Siehe weiterführend: `http://wordnetweb.princeton.edu/perl/webwn?s=send` (Stand: 11.01.17).
[79] Siehe weiterführend: `http://wordnetweb.princeton.edu/perl/webwn?s=family` (Stand: 11.01.17).
[80] Siehe weiterführend: `http://wordnetweb.princeton.edu/perl/webwn?s=email` (Stand: 11.01.17).

wurden und es sich somit definitiv um für eine FA relevante, semantische Kernaussagen handelt (z. B. „*send*“ = Aktion), ist es hinreichend zu wissen, dass mindestens eine dieser ambig ist und es im Zuge der Weiterverarbeitung zu Fehlinterpretationen führen kann. Die Disambiguierung ist demnach gerechtfertigt. Nun könnte zwar argumentiert werden, dass zum Beispiel „*send*“ unter Betrachtung des Kontextes nicht acht verschiedene Lesarten hat, sondern eigentlich nur zwei. Nämlich „*cause to be directed or transmitted to another place*“ (send.03) und „*broadcast over the airwaves, as in radio or television*“ (send.08), die beide von WordNet als *verb.communication* geführt werden. Jedoch bedeutet dies zum einen keine Ergebnisveränderung, da das Lexem auch mit zwei potentiellen Lesarten als ambig gelten würde. Zum anderen sei an dieser Stelle noch einmal an die Aufgabe der Indikatoren erinnert, die nicht darin besteht, eine Disambiguierung durchzuführen, sondern vielmehr deren Notwendigkeit aufzuzeigen. Es ist zu diesem Zeitpunkt vollkommen hinreichend zu wissen, dass mehrere Lesarten vorliegen und ein gegebenes Lexem unter allen derzeit berücksichtigten Faktoren als ambig gilt.

5.3.2.2 Syntaktische Ambiguität als Indikator

Da es sich bei syntaktischer Ambiguität um eine strukturelle Mehrdeutigkeit handelt, sind Indikatoren auf Basis von syntaktischen Mustern naheliegend (s. Kapitel 2). Auf diese Weise lässt sich sowohl Koordinations- als auch PP-Anbindungsambiguität erkennen. Die Identifikation der Indikatoren auf Satzbasis ermöglicht eine Lokalisierung möglicher Ambiguitäten innerhalb der Anforderungsbeschreibung.

Indikatoren zur Erkennung von Koordinationsambiguität

Für den Fall der Koordinationsambiguität werden Konjunktionen sowie syntaktische Muster als Indikatoren herangezogen. Koordinationsambiguität, wie sie in dieser Arbeit verstanden wird, entsteht zum einen durch die Verwendung von Konjunktionen zusammen mit Modifikatoren, was auf Basis von Muster erkannt werden kann. Zum anderen entsteht sie durch die Verschachtelung von Konjunktionen, deren Erkennung durch den systematischen Abgleich von *Token* und definierten Mustern erreicht wird (s. Abschnitt 2.1.2). In beiden Fällen ist die Existenz von Konjunktionen (Berry et al., 2003, S. 11 sowie Chantree et al., 2005, S. 2 benennen explizit „*and*“ und „*or*“) ausschlaggebend und wird überprüft, bevor Muster pro Satz gesucht werden (vgl. Beispiel 5.3.2). Diese Reihenfolge dient vor allem einer schnelleren Verarbeitung.

Beispiel 5.3.2 (Koordinationsambiguität)

(A)

PRP	*VBP*	*NNS*	***CC***	*NNS*	***CC***	*NNS*	*VBP*	*PRP*	.
I	*use*	*crawlers*	*and*	*spiders*	*and*	*users*	*report*	*me*	.

(B)

PRP	*VBP*	*TO*	*VB*	***JJ***	*NNS*	***CC***	*NNS*	.
I	*want*	*to*	*send*	*large*	*emails*	*and*	*tasks*	.

Wie in Beispiel 5.3.2 ersichtlich wird, enthalten beide Arten der Koordinationsambiguität mindestens eine Konjunktion. Dies wird als Vorauswahlkriterium herangezogen. Darauf folgend werden Sätze, die mindestens eine Konjunktion enthalten auf weitere Konjunktionen überprüft. Der Indikator erkennt eine potentielle Koordinationsambiguität, wenn mindestens zwei Konjunktionen vorliegen. Es folgt die musterbasierte Erkennung (POS-*Tags*) von Ambiguität durch die Kombination von Modifikatoren und Konjunktionen. Wie in Beispiel 5.3.2 (B) ersichtlich wird, eignet sich beispielsweise das Muster „JJ NNS CC NNS“ im Falle von zwei Nomina im Plural.

Indikatoren zur Erkennung von PP-Anbindungsambiguität

Im Fall von PP-Anbindungsambiguität existieren bereits syntaktische Muster zur Erkennung potentieller Ambiguität, wie beispielsweise „V NP PP“ von Agirre et al. (2008, S. 318) bzw. Nadh und Huyck (2009, S. 2), sodass diese hier adaptiert werden können. Dieses Muster ermöglicht es, Präpositionalphrasen zu finden, die auf Nominalphrasen in Objektposition folgen. Die hierzu notwendigen syntaktischen Informationen können durch *Shallow Parsing*-Ansätze (auch: *Chunking*) erzeugt werden, die als sehr performant und ausreichend zuverlässig gelten (Carstensen et al., 2010, S. 276). Allerdings stellt sich die Frage, ob das Muster im Sinne hoher Performanz noch weiter eingeschränkt werden kann. Eine Überlegung wäre zum Beispiel, bestimmte Präpositionen auszuschließen, die nicht als (hochgradig) ambiguitätsfördernd gelten. Beim Blick in Lexika des Englischen zeigt sich, dass sehr viele Präpositionen existieren[81], sodass der Fokus auf die Gängigsten genügt. Darum werden in Tabelle 5.5 die 15 meistgenutzten Präpositionen aufgeführt (Davies, 2016).

(1) *of*	(2) *in*	(3) *to*	(4) *for*	(5) *with*
(6) *on*	(7) *at*	(8) *from*	(9) *by*	(10) *about*
(11) *as*	(12) *into*	(13) *like*	(14) *through*	(15) *after*

Tabelle 5.5: Die häufigsten 15 Präpositionen der englischen Sprache

Die häufigste Präposition „*of*“ ist zugleich eine, die in der Literatur als nicht ambig gilt, da sie in nahezu allen Fällen an die NP gebunden wird, und daher oftmals aus Disambiguierungsverfahren ausgeschlossen wird (z. B. Ratnaparkhi, 1998, S. 1081). Die Präposition „*of*“ wird demnach auch in dieser Arbeit nicht behandelt, weshalb das syntaktische Muster folgerichtig noch um mindestens eine Konstituente zu ergänzen ist („V NP PP“ | $PREP \neq$ „*of*“).

5.3.2.3 Referentielle Ambiguität als Indikator

In diesem Abschnitt werden Indikatoren in zweierlei Hinsicht bestimmt: Zum einen muss festgestellt werden, ob Ambiguität vorliegt, um entsprechend eine Methode zur Disambiguierung zu starten. Zum anderen soll festgestellt werden, ob Koreferenzen im Text existieren, um auch deren Erkennung zu forcieren.

[81] DELA führt 124 Präpositionen, Davies (2016) listet 196 und Essberger (2012, S. 6) wiederum führt 150 unter dem Verweis auf, dass es viele weitere Präpositionen gibt.

Indikator zur Erkennung referentieller Ambiguität

Ein mögliches Vorgehen der Erkennung referentieller Ambiguität (s. Abschnitt 2.1.3) wäre, auf Basis einzelner Lexeme zu prüfen, ob beispielsweise Pronomina vorliegen. Das ist naheliegend, weil diese Ausdrücke klassischerweise auch als „Stellvertreter“ von Nomina bezeichnet werden (Dittmann und Thieroff, 2009, S. 400 ff.) und die Gefahr mit sich bringen, falsch zugeordnet zu werden (IEEE, 2011, S. 12). Diese Prüfung kann sowohl mittels POS-*Tagging* geschehen als auch auf Basis von Wortlisten. Eine Einschränkung zur Sicherung der Performanz wäre dabei beispielsweise, nur hochfrequente Pronomina abzugleichen. Hierfür stellt Tabelle 5.6 die 15 Pronomina der englischen Sprache dar, die am häufigsten verwendet werden (Davies, 2016).

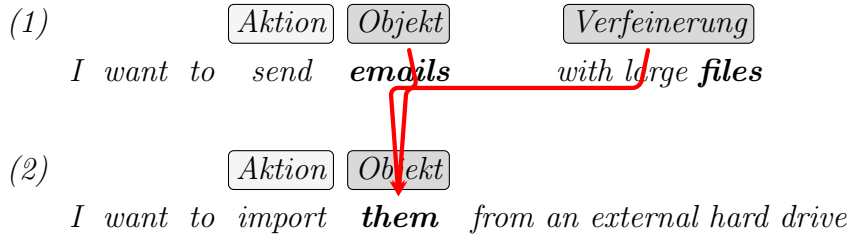
(1) *it*	(2) *I*	(3) *you*	(4) *he*	(5) *they*
(6) *we*	(7) *she*	(8) *who*	(9) *them*	(10) *me*
(11) *him*	(12) *one*	(13) *her*	(14) *us*	(15) *something*

Tabelle 5.6: Die häufigsten 15 Pronomina der englischen Sprache

Diese Lösung ist aber unter drei Gesichtspunkten unbefriedigend: Einerseits enthält Tabelle 5.6 viele Personalpronomina (z. B. „*I*“, „*she*“ oder „*it*“), wobei zu klären wäre, wie häufig Personalpronomina in Anforderungsbeschreibungen auftreten und ob sie hinsichtlich der Anforderungsqualität schädigend sind. Beispielsweise enthält bereits der Satz „***I*** *want to read emails and* ***I*** *need to print* ***them***“ drei Personalpronomina, was die Frage aufwirft, ob das bloße Vorhandensein bereits negativ zu interpretieren ist und wenn ja, ab welcher Anzahl. Ferner ist, unter der Annahme, dass es zumindest eine bekannte Standardrolle in Anforderungsbeschreibungen gibt, nämlich den *User*, auch das Personalpronomen „*I*“ als unkritisch anzusehen. Was noch übrig bleibt, ist das Wort „*them*“, das auf „*emails*“ referenziert und als Koreferenzkette abzubilden ist. Ambig ist diese Konstellation aber nicht. Darüber hinaus ist ein Abgleich aller Wörter, unabhängig von ihrer semantischen Funktion innerhalb der FA, nicht performant.

Beispiel 5.3.3 enthält einen Auszug einer Anforderungsbeschreibung, die aus zwei aufeinanderfolgenden Sätzen besteht. Satz (2) enthält als Objekt das Personalpronomen „*them*“, wobei unklar bleibt, auf was sich dieses bezieht. Es könnte sowohl das Objekt als auch die semantische Kategorie der Verfeinerung in Satz (1) sein, da beide Nomina im Plural vorliegen.

Beispiel 5.3.3 (Indikator für referentielle Ambiguität)



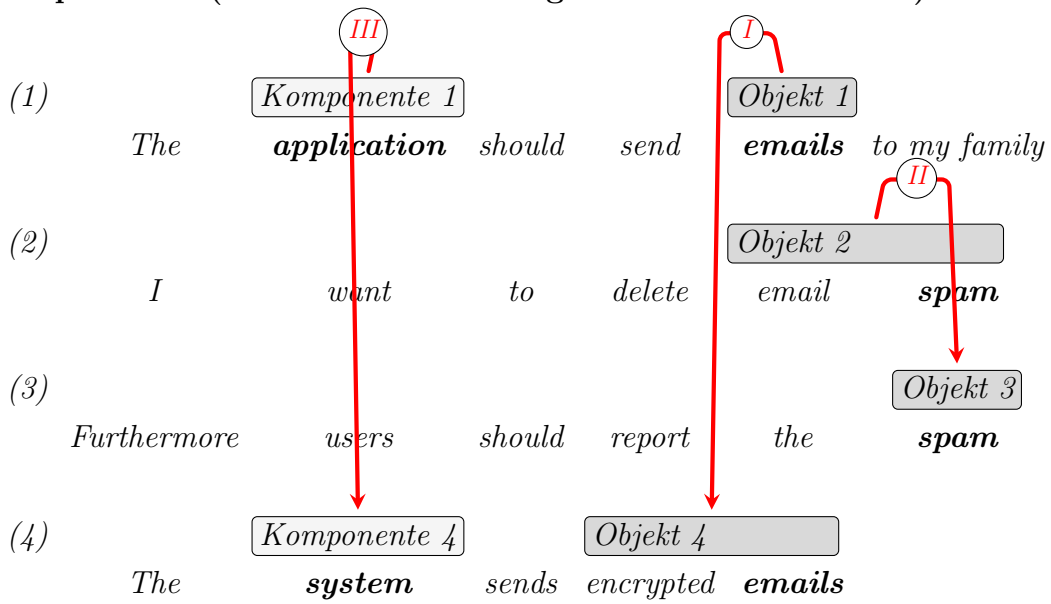
Es wird deutlich, dass die Indikatorbestimmung nicht auf Satzbasis erfolgen kann, da Antezedens und der direkte anaphorische Verweis sowohl im selben, als auch in aufeinander folgenden Sätzen auftreten können. In Beispiel 5.3.3 greift das Muster

„NNS+NNS+*them*“ satzübergreifend. Es lässt sich erweitern, indem auch Antezedenzien im Singular berücksichtigt werden, wie beispielsweise in „*I want to send an email with an attachment. It is a very large one*“ und statt „*them*“ im Muster das POS-*Tag* für Pronomen gewählt wird („NN(S)+NN(S)+PRP“). Allerdings greift das Muster nur, wenn entweder zwei Antezedenzien im selben Satz mit einem Pronomen oder im Satz zuvor genutzt werden. Eine Aufteilung der Antezedenzien auf zwei Sätze wird nicht berücksichtigt, da davon ausgegangen wird, dass in diesen Fällen auf das zuletzt genannte Antezedens referenziert wird.

Indikator zur Erkennung von Koreferenzen

Um auch Koreferenzen aufdecken zu können, wird im Folgenden eine Kombination aus Pronomina, semantischen Informationen der FA und der Ähnlichkeit zwischen Objekten, Komponenten, Rollen etc. gewählt, sofern davon mehrere existieren. Beispiel 5.3.4 illustriert das an einer um semantische Kategorien erweiterten Anforderungsbeschreibung[82], die aus insgesamt vier Sätzen besteht. Während die Anforderungen in den Sätzen (1) und (4) aus der Komponentensicht verfasst wurden (Was soll eine Komponente tun?), zeigen die Sätze (2) und (3) die Nutzerperspektive (Was soll der Nutzer tun können?).

Beispiel 5.3.4 (Indikator zur Bildung von Koreferenzketten)



Es liegen – markiert durch die Pfeile I-III – Koreferenzen vor, die sich in einzelnen Merkmalen unterscheiden und die es zu erkennen gilt, sodass die referentielle Disambiguierung zur Erstellung von Koreferenzketten durchgeführt wird. Gegenstand des Indikators in diesem Fall sind die semantischen Kategorien und die Ähnlichkeiten der dahinter befindlichen Wörter. Durch die IE sind semantische Informationen inklusive POS-*Tags* bekannt: Die Komponenten umfassen $\{application_{NN}, system_{NN}\}$ und zu den Objekten gehören $\{emails_{NNS}, email_{NN}\ spam_{NN}, spam_{NN}, encrypted_{JJ}\ emails_{NNS}\}$. Folgend werden Nomina einer semantischen Kategorie sowie verwandten Kategorien

[82] Aus Gründen der besseren Lesbarkeit werden nur relevante Kategorien dargestellt.

untereinander verglichen (vgl. Tabelle 5.7). Besteht eine Kategorie aus mehreren Nomina, werden diese jeweils einzeln miteinander abgeglichen (z. B. „$email_{NN}$ $spam_{NN}$“). Vollständige sowie partielle Übereinstimmungen in den verglichenen Kategorien gelten hier als ausreichend für die Ausführung der referentiellen Disambiguierung.

In Beispiel 5.3.4 existiert eine direkte Übereinstimmung zwischen den Nomina von Objekt 1 und 4 („*emails*“) sowie eine direkte Übereinstimmung zwischen Objekt 2 und 3 („*spam*“). Darüber hinaus liegt eine partielle Übereinstimmung zwischen Objekt 2 und den Objekten 1 und 4 vor („*email*“). Bei den Komponenten 1 und 4 hingegen handelt es sich um einen Sonderfall. Ein direkter Abgleich zwischen „*system*“ und „*application*“ führt ins Leere, wobei beide Begriffe auf die identische Entität referenzieren. Um dennoch einen Hinweis auf potentielle Koreferenz zu finden, wird eine Synonymliste herangezogen, die hochfrequente Begriffe enthält. Durch diese Liste wird erkennbar, dass die beiden Wörter zusammenhängen und Koreferenz gegeben ist – der Indikator greift demnach auch in diesen Fällen, da durch die Hinzunahme der Synonyme eine gewisse Ungenauigkeit in Kauf genommen wird.

Sem. Kategorie	**Abgleich**	**Hinreichend**
Priorität	○	—
Subpriorität	○	—
Aktion	○	—
Aktionsargument	○	—
Subaktion	○	—
Subaktionsargument	○	—
Sonstiges		
Motivation	○	—
Bedingung	●	○
Rolle	●	○
Subrolle	●	○
Komponente	●	●
Komponentenverfeinerung	●	●
Objekt	●	●
Objektverfeinerung	●	●
Subobjekt	●	●
Subobjektverfeinerung	●	●

Tabelle 5.7: Gruppen semantischer Kategorien zum Ähnlichkeitsabgleich. Gruppenoberbegriffe sind fett gedruckt hervorgehoben

Nun stellt sich auch bei diesem Indikator die Frage, ob bereits ein einmaliger Hinweis auf potentielle Koreferenz ausreicht, um die Ausführung der referentiellen Disambiguierung zu rechtfertigen oder ob weitere Voraussetzungen erfüllt sein müssen. In der Tat bietet es sich hier an, zwischen den semantischen Kategorien zu unterscheiden. Wie in Tabelle 5.7 ersichtlich ist, werden nicht alle Kategorien für einen Abgleich hinzugezogen. So wird beispielsweise auf „Priorität“, „Aktion“ und „Motivation“ verzichtet, wobei Prioritäten (z. B. „*want*“, „*must*“) und Aktionen („*send*“) *per se* nicht koreferent sind. Auf die Kategorie „Motivation“ wird aus Effizienzgründen

verzichtet, da sie als nicht hochgradig relevant für die FA eingestuft wird – anders als zum Beispiel die Kategorie „Bedingung“, welche wiederum allein nicht hinreichend ist, um einen positiven Indikator darzustellen. Ebenfalls sind die Kategorien rund um „Rolle“ isoliert nicht ausreichend, um die Methodenanwendung zu rechtfertigen, da üblicherweise nicht mehr als zwei unterschiedliche Rollen innerhalb einer Anforderungsbeschreibung vorkommen und diese überwiegend Standardrollen sind (z. B. „*User*“, „*I*“). Wenn die Notwendigkeit der Methodenausführung bei „Komponente“ und „Objekt“ sowie deren Ausprägungen erkannt wird, wird davon ausgegangen, dass eine Wiederaufnahme bereits eingeführter Referenten erfolgt ist.

5.3.2.4 Indikatoren zur Erkennung von Unvollständigkeit

Die Bestimmung der Indikatoren für Unvollständigkeit gestaltet sich vergleichsweise schwierig, da das Fehlen von Angaben zu prüfen ist. Als Vorteil erweist sich hierbei, dass es sich um partiell unvollständige Anforderungen handelt (s. Abschnitt 2.3). Es ist demnach zu überlegen, ob die bestehenden Informationen Rückschlüsse auf die fehlenden Angaben zulassen. Wie auch bei den anderen Indikatoren muss dies allerdings ohne die entsprechende Kompensationsmethode geschehen. Da die Unvollständigkeit auf Basis der Prädikate ermittelt wird, stehen diese im Zentrum der nachfolgenden Überlegungen. Und auch hier werden die semantischen Kategorien als Grundlage der kontextsensitiven Indikatoren herangezogen.

Ein erster Indikator für Unvollständigkeit ist die fehlende semantische Kategorie „Aktion“ (vgl. auch Tabelle 5.7). Diese lässt sich allerdings nur schwerlich kompensieren, bildet sie doch zum einen die Ausgangslage[83] der eigentlichen Kompensation und zum anderen die semantische Kernaussage einer Anforderung[84]. Eine Anforderung ohne „Aktion“ bzw. Prozesswort ist demnach nicht als Anforderung zu behandeln. In den meisten Fällen sollte hier bereits die *Off-Topic*-Klassifikation während des *Preprocessings* den entsprechenden Satz der Anforderungsbeschreibung als irrelevant kennzeichnen. Neben den Prozesswörtern existieren die semantischen Kategorien für Rolle, Komponente und Objekt. Diese Kategorien stellen Argumente eines Prädikats dar. Das Subjekt einer Anforderung wird durch die Rolle oder Komponente ausgedrückt, wobei es ausreicht, wenn eine der beiden semantischen Kategorien in einem Satz vorzufinden ist. Fehlt das Subjekt, muss die Kompensation greifen, da Unvollständigkeit angenommen werden kann. Ein fehlendes Subjekt kann bereits bei der maschinellen Verarbeitung (IE) auftreten, insbesondere bei einer Aufzählung von Prozesswörtern, wie das Beispiel 5.3.5 zeigt. Hier wird das Subjekt („*I*“) korrekt dem Prädikat „*write*“ zugeordnet aber beispielsweise bei „*send*“ als Argument ignoriert.

Beispiel 5.3.5 (Auszug einer Anforderungsbeschreibung)
„$\boldsymbol{I}_{Rolle}$ *want to* $\boldsymbol{write}_{Aktion}$, $read_{Aktion}$ *and* $\boldsymbol{send}_{Aktion}$ $e\text{-}mails_{Objekt}$“

Ein weiteres Argument des Prädikats ist die semantische Kategorie „Objekt“, wie zum Beispiel „*e-mails*“ in Beispiel 5.3.5. Im Gegensatz zum Subjekt und Prädikat ist das Objekt hier nicht erforderlich, um einen wohlgeformten Satz zu bilden

[83] Die Aktion stellt das zentrale Verb einer FA dar. Ihr Fehlen verhindert die PAS-Ermittlung.
[84] Siehe hierzu auch Rupp (2007, S. 219 ff.).

(vgl. z. B. Bakshi, 2000, S. 3 f.). Es wird aber oftmals benötigt, um einen aussagekräftigen Satz zu konstruieren. Darüber hinaus wird angenommen, dass auch in den überwiegenden Fällen bei Anforderungsbeschreibungen ein Objekt erforderlich ist. Eine Anforderung wird demnach auch dann als unvollständig angesehen, wenn die semantische Kategorie „Objekt“ fehlt.

5.3.2.5 Indikatorabhängigkeiten

Wie dargestellt, bedürfen die Kompensationsmethoden unterschiedlicher Indikatoren, da auf spezifische Qualitätsmerkmale in Anforderungsbeschreibungen zurückgegriffen wird. Eine Besonderheit ist dabei, dass die Mehrzahl der Indikatoren insbesondere auf die Semantik als Quelle linguistischer Charakteristika zurückgreift, nämlich auf die durch die IE bestimmten semantischen Kategorien. Abbildung 5.13 zeigt die möglichen semantischen Kategorien (vgl. Tabelle 5.7) und deren Einfluss auf Indikatoren unterschiedlicher Kompensationsmethoden (auch: Indikatorabhängigkeit).

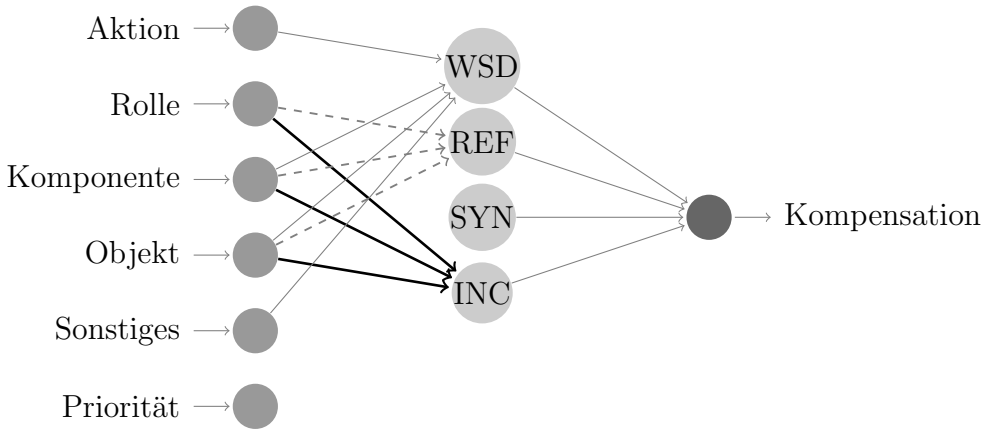


Abbildung 5.13: Einfluss semantischer Kategorien auf Indikatoren.
WSD = Lexikalische Ambiguität; REF = Referentielle Ambiguität
SYN = Syntaktische Ambiguität; INC = Unvollständigkeit

Beispielhaft hervorgehoben (schwarz, fett gedruckt) sind in Abbildung 5.13 die Pfeile ausgehend von den semantischen Kategorien „Rolle“, „Komponente“ und „Objekt“ zum Indikator für Unvollständigkeit. Dieser Indikator berücksichtigt demnach drei semantische Kategorien, die wiederum auch von Indikatoren der lexikalischen und referentiellen Ambiguität herangezogen werden (mit Ausnahme der Kategorie „Rolle“ im Fall der lexikalischen Ambiguität). Im Umkehrschluss bedeuten mehrfache Indikatorabhängigkeiten auch, dass ein Fehler in einer semantischen Kategorie nicht nur einen, sondern alle Indikatoren betrifft, die diese Kategorie in die Entscheidungsfindung miteinbeziehen. Wird nun die Frage nach der Zuverlässigkeit einzelner Indikatoren gestellt, ist eine gleichzeitige Betrachtung der zugrundeliegenden Informationen und deren Zuverlässigkeit ratsam (s. Abschnitt 8.2.3).

Es fällt darüber hinaus auf, dass die Indikatoren für syntaktische Ambiguität nicht auf die semantischen Kategorien zurückgreifen. Dies lässt sich dadurch erklären, dass sich syntaktische Ambiguität im gesamten Satz manifestiert und daher auch nur eine Betrachtung auf Basis morpho-syntaktischer Charakteristika, losgelöst von

semantischen Funktionen, ausreichend ist. Um die Nachvollziehbarkeit für Anwender zu sichern, werden gefundene Indikatoren erläutert (vgl. Abbildung A.1 im Anhang).

5.4 Strategieindex

Der in Abbildung 5.4 dargestellte Strategieindex greift sowohl die Strategien als auch die Indikatoren auf, indem jeder Strategie eine Indikatorkombination zugeordnet wird. Diese Indikatorkombination dient daraufhin sowohl der Beschreibung einer Strategie (Strategieumfang) als auch der Auswahl geeigneter Strategien hinsichtlich der Indikatorkombination einer gegebenen Anforderungsbeschreibung durch den *Selector*. Dabei kann jede Kombination von Indikatoren derzeit nur einmal im Strategieindex vorkommen (im Sinne eines einmaligen Schlüssels in relationalen Datenbanken). Grundsätzlich ist es dabei aber denkbar, diese Limitation aufzuheben und die Bezeichnung der Strategien (z. B. *Basic Plus*) oder eine fortlaufende Identifikationsnummer als einmaliges Merkmal zu nutzen. Offen bleibt dann jedoch die Frage, wie zwischen zwei Strategien zu entscheiden ist, die die gleiche Indikatorkombination unterstützen. Denkbar wäre hier die Entscheidung für oder gegen eine Strategie auf Grundlage der Endanwenderpräferenz hinsichtlich Präzision und Performanz.

Im Rahmen dieser Arbeit kann jede Indikatorkombination durch genau eine Strategie abgedeckt werden. Tabelle 5.8 stellt hierzu den initialen Strategieindex dar.

Strategie	**Indikatorkombination**	**Sem. Kategorien**	**Zusatzinformationen**
Light	–	–	–
Basic	SYN+INC	Aktion Komponente Objekt Rolle	*Chunking*
Basic Plus	SYN+INC+WSD	jegliche	*Chunking, WordNet*
Default	SYN+INC+WSD+REF	jegliche	*Chunking, WordNet*, POS
Fallback	jegliche	jegliche	*Chunking, WordNet*, POS

Tabelle 5.8: Initialer Strategieindex

Die *Basic*-Strategie hat beispielsweise den eindeutigen Schlüssel SYN+INC und führt somit die syntaktische Disambiguierung und die Unvollständigkeitskompensation aus, sofern entsprechende Indikatoren vorliegen. Die Indikatoren wiederum ergeben sich in diesem Beispiel aus vier semantischen Kategorien und unter Zuhilfenahme des *Chunkings*. Es liegen hier keine Überschneidungen in den zugrundeliegenden Informationen vor. Anders sieht dies bei der *Default*-Strategie aus. Hier werden jegliche semantische Kategorien herangezogen, die von unterschiedlichen Indikatoren genutzt werden. Fehlerhafte Informationen können in dieser Strategie demnach zu mehreren falschen Indikatoren führen. Darüber hinaus sind die semantischen Kategorien allein nicht mehr ausreichend, sodass WordNet als Ressource zur Erkennung lexikalischer Ambiguität und ein Verfahren des POS-*Taggings* zur Erkennung referentieller Ambiguität hinzugezogen werden muss.

Die Darstellung in Tabelle 5.8 dient dabei auch der Zusammenfassung bisheriger Systembestandteile, bevor im Folgenden auf die Methoden eingegangen wird.

5.5 Geplantes Vorgehen und Methodik

In diesem Abschnitt wird der Verarbeitungsprozess sequenziell geschildert. Auf diese Weise können die Komponenten unabhängig von den Strategien erläutert werden, die die Verarbeitungsreihenfolge bestimmen bzw. verändern können (s. Abschnitt 5.2).

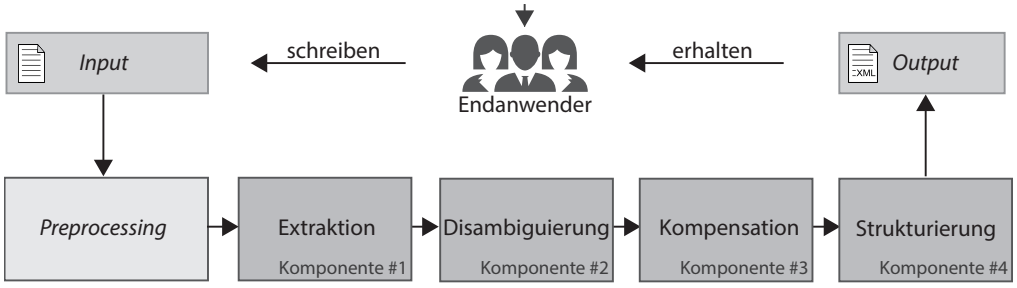


Abbildung 5.14: Informationsverarbeitung (vereinfachte Darstellung)

Ausgangspunkt für die Informationsverarbeitung sind Anforderungsbeschreibungen, die von Endanwendern (s. Abschnitt 1.1) formuliert und über eine Benutzerschnittstelle (s. Abschnitt 5.5.1) an das Softwaresystem (auch: System)[85], CORDULA, übermittelt werden (vgl. Abbildung 5.14). Aufgrund der gravierenden Unterschiede in der Beschreibungsqualität werden die Anforderungsbeschreibungen zuerst einem *Preprocessing* unterzogen (s. Abschnitt 5.5.2) und dann an die Anforderungsidentifikation und -extraktion (Komponente #1) weitergeleitet.

Die Extraktion der Anforderungen umfasst in dieser Arbeit primär die binäre Klassifikation von Anforderungen und nebensächlichen Aussagen im Fließtext. Die Extraktion von FA, was die Klassifikation von FA und NFA voraussetzt, zielt vor allem auf die Performanzsteigerung des Gesamtsystems ab. Dieser Schritt wird als erster bei der Informationsverarbeitung umgesetzt (s. Abschnitt 5.5.3), da davon auszugehen ist, dass eine Beschränkung auf relevante Aussagen zu einer erheblichen Verringerung der Gesamtkomplexität führt. Darauf folgen die Komponenten #2 und #3 zur Disambiguierung und zur Kompensation der Unvollständigkeit, die in Abbildung 5.14 ebenfalls sequenziell dargestellt sind. Die Ergebnisse werden von Komponente #4 vor der finalen Ausgabe für den Endanwender strukturiert[86]. Abschließend erhalten die Endanwender ihre kompensierte Anforderungsbeschreibung zur Ansicht zurück.

5.5.1 Design der Benutzerschnittstelle mit Eingabemaske

Um Endanwender zu befähigen, Anforderungsbeschreibungen möglichst einfach an das Softwaresystem zu übermitteln, sind Benutzerschnittstellen erforderlich, die intuitiv zu bedienen sind und somit keine bis sehr niedrige Nutzungsbarrieren aufweisen.

Bei der Benutzerschnittstelle (vgl. Abbildung 5.15), die standardmäßig vorgesehen ist, handelt es sich um eine Webapplikation, die auf allen internetfähigen Endgeräten

[85] In dieser Arbeit wird der Begriff des „Softwaresystems“ anstelle von „Programm“ genutzt, da ein Softwaresystem „[...] die Gesamtheit aller Softwarebausteine (Module), die sich in einem ganzheitlichen Zusammenhang befinden [, abbildet]“ (Denert, 2013, S. 11) während ein Programm mit einer „[...] einzelnen, kleinen Lösung assoziiert [wird]“ (Denert, 2013, S. 11).

[86] Strukturiertheit wird hier über syntaktische Muster erzeugt (s. Abschnitt 5.5.7).

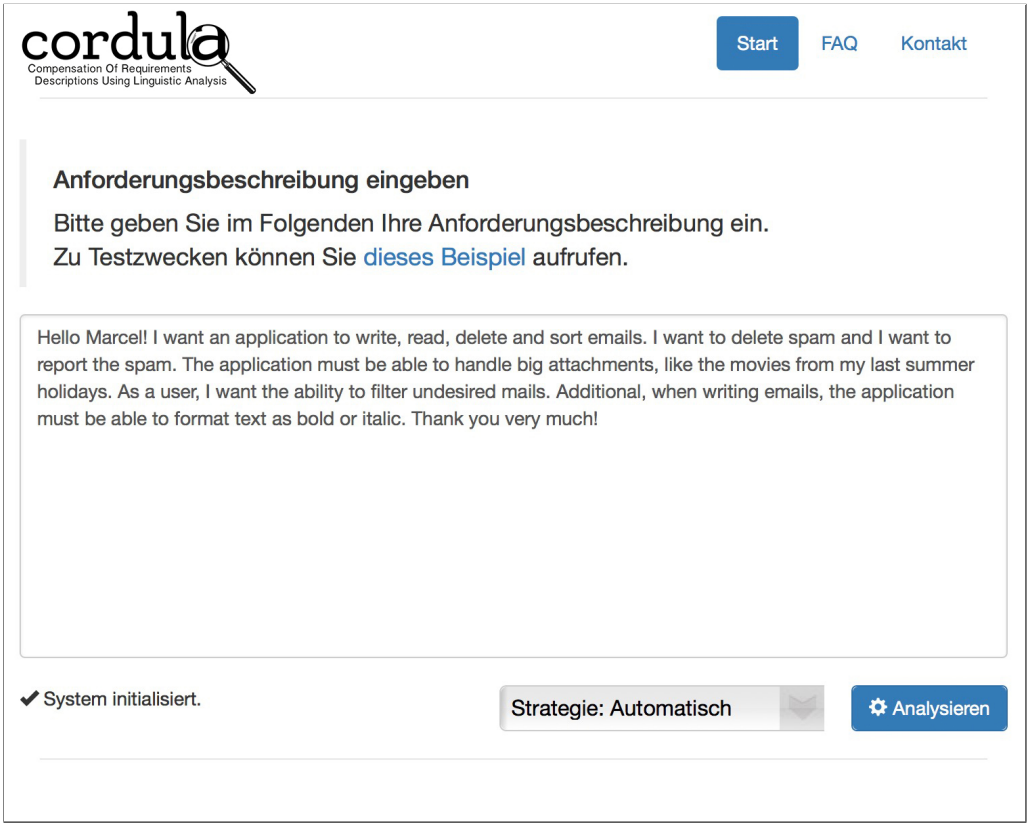


Abbildung 5.15: Benutzerschnittstelle von CORDULA (*Frontend*)

mittels *Webbrowser* genutzt werden kann. Alternative Benutzerschnittstellen, beispielsweise Smartphone-Applikationen, sind denkbar (vgl. Abbildung 5.1), aber auf Grund des responsiven Designs nicht zwingend erforderlich (s. Abschnitt 7.4.2.2).

Abbildung 5.15 stellt das zentrale Eingabeformular dar, welches den Endanwendern zur Eingabe der Anforderungsbeschreibungen präsentiert wird. Grundsätzlich wird lediglich ein Textfeld und ein weiteres Bedienelement zur Initialisierung der Verarbeitung und Kompensation benötigt. Da keine Formatierung der Anforderungsbeschreibungen vorgesehen ist, sind Menüpunkte wie „Fettdruck“ oder „Kursiv“, wie sie klassische Textverarbeitungsapplikationen erwarten lassen, hier obsolet. Entsprechend handelt es sich bei dem *Input*, der an das System über die dargestellte Benutzerschnittstelle übergeben wird, um unformatierte Beschreibungen.

5.5.2 Textvorverarbeitung

Unter Textvorverarbeitung (auch: *Preprocessing*) wird die Durchführung diverser Schritte zur Textoptimierung und -analyse, die in Abhängigkeit der zu erwarteten Eigenschaften des *Inputs* zu konfigurieren sind, verstanden (s. Anhang C.1). Dabei sind nicht alle existierenden Verfahren des *Preprocessings* (z. B. *Chunking*, HTML-*Stripping*) zwangsläufig anzuwenden. Zur Diskussionen stehen dabei die neun folgend aufgeführten Verfahren:

(1) Normalisierung
(2) Sprachenidentifikation
(3) Grammatikprüfung
(4) Tokenisierung
(5) Synonymerkennung
(6) Textbereinigung
(7) Rechtschreibkorrektur
(8) Satzendeerkennung
(9) POS-*Tagging*

Die Anforderungsbeschreibungen, die an das Softwaresystem übergeben werden, können beliebige Textzeichen enthalten, sehen jedoch keine *Markups*[87] vor (s. Abschnitt 5.5.1). Nichtsdestotrotz ist der *Input* von jeglichem *Markup* aus Gründen der Systemsicherheit zu bereinigen (Bhargav und Kumar, 2010, S. 267 ff.). Der *Input* ist demnach einer **(1) Normalisierung**, die ungültige Zeichen[88] erkennt und entfernt sowie einer **(6) Textbereinigung** (z. B. HTML-*Stripping*) zu unterziehen. Die Textnormalisierung ist notwendig, da ungültige Zeichen zu Fehlern in den folgenden Komponenten führen können. Verzichtet wird auf die Normalisierung von Groß- und Kleinschreibung, da Großschreibung zum Beispiel als ein Indiz bei der Erkennung von benannten Entitäten genutzt werden kann.

Die **(7) Rechtschreibkorrektur** und **(3) Grammatikprüfung**, deren Notwendigkeit sich aus den zu erwarteten Ungenauigkeiten in den Anforderungsbeschreibungen ergibt (s. Abschnitt 1.4), folgt auf die Normalisierung. Da keine Benutzerinteraktion im *Preprocessing* vorgesehen ist, müssten Rechtschreibfehler automatisch korrigiert werden. Diese automatische Korrektur ist bei Grammatikfehlern aufgrund der Sprachkomplexität ohne Benutzerinteraktion nicht zuverlässig umzusetzen. Und auch die automatische Rechtschreibkorrektur geht mit der Gefahr einher, mehr Fehler zu erzeugen als zu eliminieren, da zum Beispiel Dateiendungen und Fachtermini in den Anforderungsbeschreibungen fälschlicherweise korrigiert werden könnten. Dennoch werden Rechtschreib- und Grammatikprüfung durchgeführt, da diese Erkenntnisse zur Verbesserung der Folgekomponenten (z. B. lexikalische Disambiguierung) herangezogen werden können. Da keine weiteren Vorverarbeitungsschritte auf dem gesamten Fließtext anzuwenden sind, kann aufbauend auf der Rechtschreibkorrektur und Grammatikprüfung die **(8) Satzendeerkennung** angewendet werden. Diese ist notwendig, da einzelne Folgekomponenten die satzweise Eingabe der Anforderungsbeschreibung erwarten, beispielsweise die Klassifikation nach *On-* und *Off-Topic*.

Als Folgekomponente der Satzendeerkennung ist die **(2) Sprachenidentifikation** (s. Anhang C.1) zu diskutieren. Ihre Notwendigkeit ergibt sich aus sprachspezifischen NLP-Komponenten wie der Disambiguierung oder der Kompensation von Unvollständigkeit, die zwar in dieser Arbeit auf die englische Sprache angewendet werden, grundsätzlich aber adaptierbar sind (s. Abschnitt 7.4.2). Handelt es sich beim gesamten Text um eine nicht unterstützte Sprache, muss das System den Verarbeitungsvorgang ergebnislos abbrechen.

Aus den eben genannten Komponenten ergibt sich die in Abbildung 5.16 dargestellte *Preprocessing pipeline*. Wie ersichtlich wird, handelt es sich um einen sequenziellen Vorgang. Die Ausgabe einer Komponente ist somit stets die Eingabe der Folgekomponente. Eine strukturelle Änderung am Fließtext findet erst durch die Satzendeerkennung statt. Das bedeutet, dass als Ausgabe dieser Vorverarbeitung

[87] Weder Hervorhebungen (z. B. Fettdruck) noch HTML-Auszeichnungen sind vorgesehen.
[88] Unter gültigen Zeichen werden in dieser Arbeit alle druckbaren Zeichen des ASCII verstanden.

eine bestimmte Anzahl an Sätzen steht, die sich durch Textoptimierungsmaßnahmen vom initialen Fließtext des Endanwenders unterscheiden können.

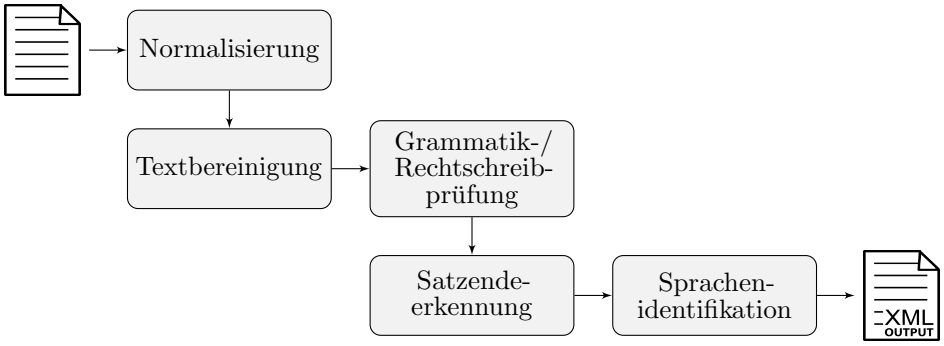


Abbildung 5.16: Ablauf des *Preprocessings*

Verzichtet wird an dieser Stelle der Arbeit auf die Tokenisierung, Synonymerkennung und das POS-*Tagging*. Diese Entscheidung wird getroffen, da die Tokenisierung als auch das POS-*Tagging* bereits feste Bestandteile der meisten NLP-Komponenten sind. Das Erkennen von Synonymen beim *Preprocessing* wird ausgelassen, da dieser Schritt in der lexikalischen Disambiguierung bereits integriert ist.

5.5.3 Anforderungsextraktion

Die Extraktionskomponente übernimmt zwei Aufgaben. Zum einen muss, aufgrund der zu erwartenden niedrigen Textqualität der Anforderungsbeschreibungen, *On-Topic* von *Off-Topic* getrennt werden. Zum anderen muss das Wesentliche auf semantische Textelemente zur Beschreibung von FA[89] (z. B. Rollen, Aktionen) herunter gebrochen werden. Wie in Abschnitt 3.2 aufgezeigt wird, existieren nur wenige Arbeiten zur Anforderungsextraktion auf qualitativ stark variierenden Fließtexten. Eine Ausnahme stellt das von Dollmann und Geierhos (2016) entwickelte *Requirements Extraction and Classification Tool* (REaCT) dar, das beide genannten Aufgaben erfüllt und in dieser Arbeit zur Anforderungsextraktion herangezogen wird. Es eignet sich besonders zur Anwendung, da es einerseits UGC wie Anforderungsbeschreibungen (s. Abschnitt 1.4) unterstützt, sich bei der Analyse sehr stark an der englischen Grammatik orientiert und andererseits eine strukturierte Datenausgabe vorsieht. Darüber hinaus wird es aktiv weiterentwickelt (Dollmann und Geierhos, 2016).

Abbildung 5.17 zeigt den Ablauf der Anforderungsextraktion. Dollmann und Geierhos (2016) arbeiten dabei auf Satzebene[90], sodass die durch das *Preprocessing* erhaltenen Sätze direkt in die Klassifikationskomponente zur Anforderungsidentifikation übergeben werden können. Dollmann (2016, S. 64) stellt eine Auswahl geeigneter Klassifikatoren vor und nutzt schlussendlich für die *On-Topic*- und *Off-Topic*-Klassifikation den *ExtraTreeClassifier* (Geurts et al., 2006), der eine Erweiterung der *Random-Forests*-Methode darstellt, zusammen mit einer *Feature*-Kombination aus

[89] Auch: Semantische Kategorien.

[90] In dieser und den folgenden Abbildungen wird als *Input* ein XML-Dokument angezeigt, um hervorzuheben, dass es sich nicht um Ursprungstexte handelt, sondern um vorverarbeitete Sätze.

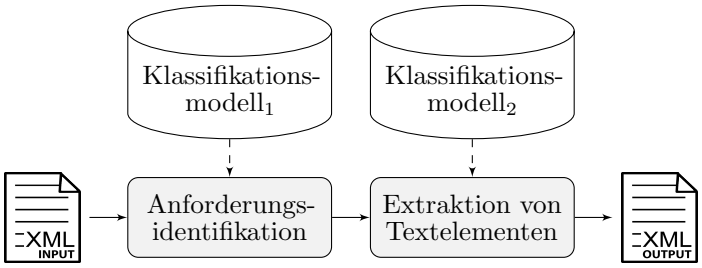


Abbildung 5.17: Anforderungsidentifikation und -extraktion

Bag-of-Words und Satzlänge. Das so entwickelte System erzielt einen F_1-*Score* von 89% bei der Differenzierung von Anforderungen und nebensächlichen Sätzen. Handelt es sich bei einem klassifizierten Satz um *On-Topic* und damit um Anforderungen, wird die Extraktion von Attribut-Wert-Paaren vorgenommen, mit dem Ziel, das zuvor definierte *Template* iterativ zu befüllen (vgl. Abbildung 5.18).

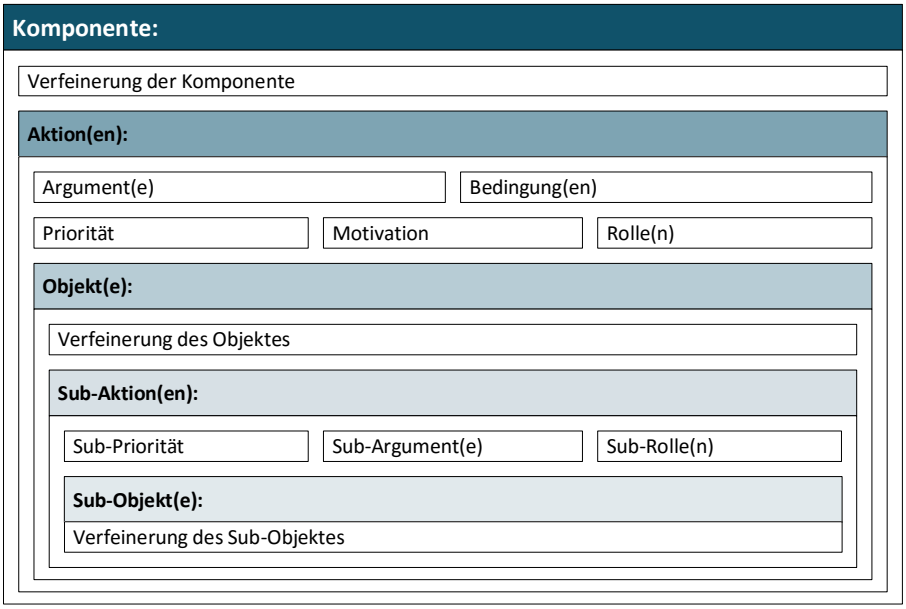


Abbildung 5.18: *Template* funktionaler Anforderungen.
Entnommen aus Dollmann (2016, S. 54)

Die wichtigsten Elemente des *Templates* sind die Komponente (Subjekt), die Aktion (Prädikat) und das Objekt. Aktionen beschreiben, was eine Komponente leisten soll und Objekte geben an, worauf sich die Aktionen beziehen. Komponenten sowie Objekte können in den Anforderungsbeschreibungen weiter konkretisiert werden, wofür die Felder Verfeinerung der Komponente und Verfeinerung des Objektes vorgesehen sind (vgl. Abbildung 5.18). Darüber hinaus können Vor- und Nachbedingungen (z. B. Zeitrestriktionen) existieren, die für die Ausführung von Aktionen gelten sollen und die im *Template* als Bedingungen angegeben sind (Dollmann und Geierhos, 2016). Die in den Abschnitten 5.2 und 5.3 vorgestellten Strategien und Indikatoren greifen auf die abgebildeten semantischen Kategorien zurück. Ein Überblick dazu findet sich in Tabelle 5.7.

5.5.4 Disambiguierung

Wie in Abschnitt 2.1 dargestellt, handelt es sich bei Ambiguität um ein facettenreiches Phänomen in der Anforderungsbeschreibung, für das keine Allzwecklösung existiert. Beispielsweise hat lexikalische Ambiguität einen anderen Ursprung als syntaktische Ambiguität und muss daher auf eine andere Weise erkannt und kompensiert werden. Um dieser Herausforderung flexibel zu begegnen, ist es erforderlich die Disambiguierungskomponente modular aufzubauen, um auf den jeweiligen Ambiguitätstyp reagieren zu können.

5.5.4.1 Lexikalische Disambiguierung

Verfahren der lexikalischen Disambiguierung sind in Abschnitt 3.3.1 aufgeführt. In dieser Arbeit wird Babelfy zur lexikalischen Disambiguierung herangezogen (Moro et al., 2014a; Moro et al., 2014b). Babelfy kann unter Hinzunahme der Ressource BabelNet (s. Abschnitt 3.3.1.1) auf eine Vielzahl weiterer Ressourcen (z. B. Wikipedia, WordNet) zur Disambiguierung zurückgreifen und entsprechende Annotationen vornehmen (s. Abschnitt 3.3.1.1). Hierbei werden Komposita und Eigennamen unterstützt (vgl. Abbildung A.3 im Anhang), was zur hohen Erkennungsqualität beiträgt.

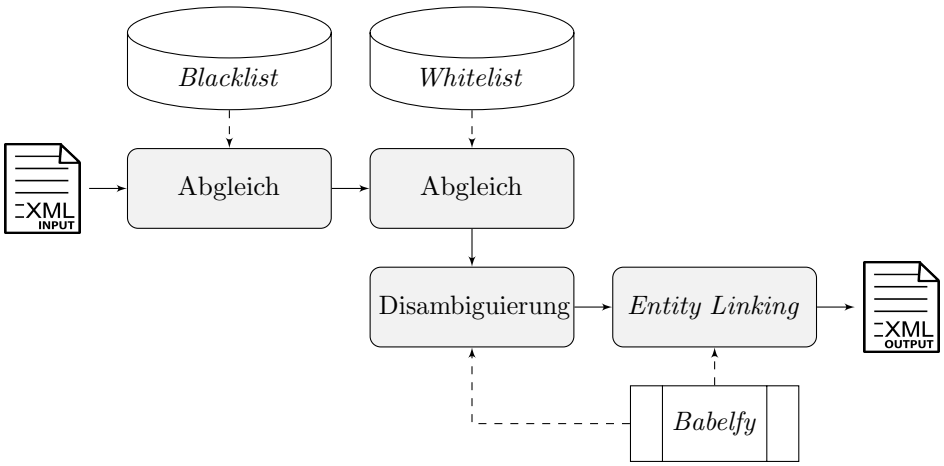


Abbildung 5.19: Funktionsweise der lexikalischen Disambiguierung

Der Ablauf der lexikalischen Disambiguierung und Annotation ist in Abbildung 5.19 dargestellt. Da diese auf Basis von *Token* durchgeführt werden soll, ist eine Tokenisierung und ein POS-*Tagging* erforderlich. Einzelne *Token* können im Folgenden der Disambiguierung unterzogen werden, wobei nur die *Token* disambiguiert werden, die nicht auf der Stoppwortliste stehen und die zuvor von der Anforderungsextraktion als relevant erkannt wurden. Die Stoppwortliste (*Blacklist*) enthält Lexeme in ihrer Grundform, die nicht bedeutungstragend sind (z. B. Funktionswörter). Ziel dieser Einschränkung ist das Erreichen einer höheren Verarbeitungsgeschwindigkeit, da eine Disambiguierung nur bei bedeutungstragenden *Token* erforderlich ist. Die gleiche Intention gilt bei der *Whitelist*, die Lexemen aufgrund des Vorkommens in einer spezifischen Domäne (Domänenkorpus) eine zuvor definierte Lesart zuweist. Die Endanwender können sich das Ergebnis der lexikalischen Disambiguierung in der

Benutzerschnittstelle ausgeben lassen. Abbildung 5.20 zeigt dies exemplarisch für den Begriff „*application*“.

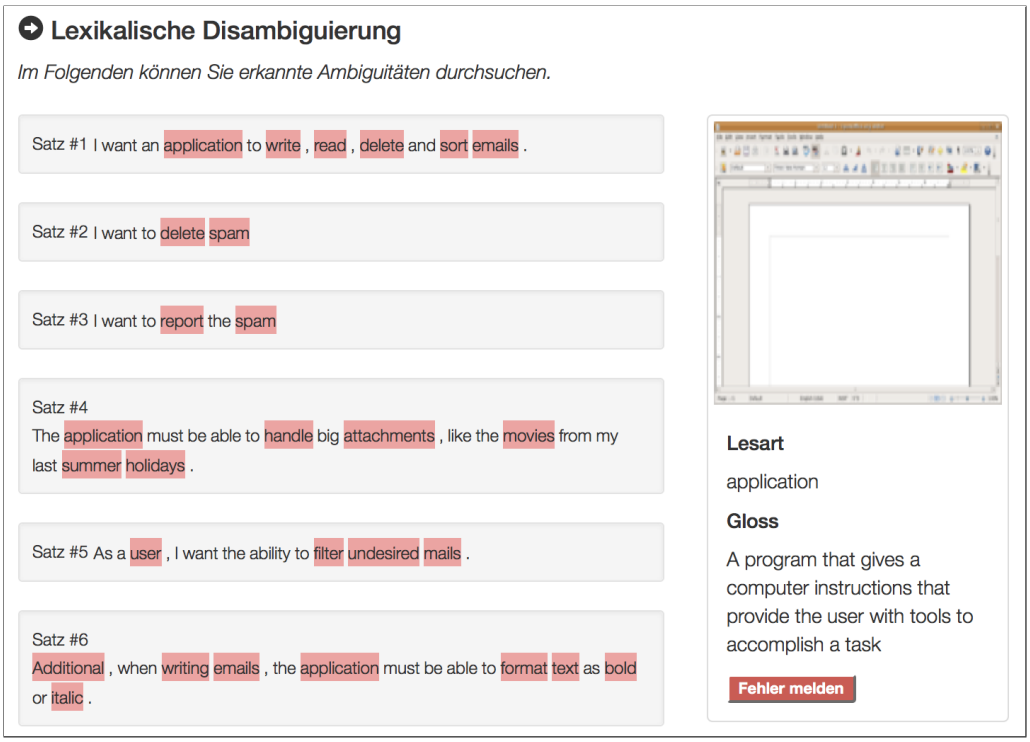


Abbildung 5.20: Lexikalische Disambiguierung (*Frontend*)

Ein korrigiertes POS-*Tag* in der Anforderungsextraktion zeigt beispielhaft Abbildung A.4 im Anhang. Neben reinen Hervorhebungen sind Regeln notwendig, die flexibel auf folgende Ergebnisse der lexikalischen Disambiguierung reagieren können:

- **Disambiguierung nach domänenspezifischen Kategorien**
  Das *Token* „Arbeitskollegen“ („*colleagues*“) kann bei minimaler Kontextinformation auch zu „*Woollahra Colleagues Rugby Football Club*“ mit der Kategorie „*Rugby union teams in Sydney*“[91] als Lesart führen. Hier sind Regeln zur Fehlerbehebung auf Kategorienebene erforderlich.

- **Umgang mit falschen POS-*Tags***
  Durch falsche POS-*Tags* kann eine falsche Disambiguierung erfolgen. So kann aus „[...] *in order to* [...]„ eine Order[92] werden. Hier kann durch eine Reihe vordefinierter Regeln eine falsche POS-Zuweisung aufgedeckt und korrigiert werden. Daraufhin muss die Disambiguierung erneut ausgeführt werden.

Auf die lexikalische Disambiguierung folgt die syntaktische Disambiguierung, die ebenfalls auf Satzbasis arbeitet.

[91] Siehe weiterführend: `http://babelnet.org/synset?word=bn:14854243n` (Stand: 11.01.17).
[92] „[...] *a command given by a superior that must be obeyed*“. Siehe weiterführend: `http://babelnet.org/synset?word=bn:00059303n` (Stand: 11.01.17).

5.5.4.2 Syntaktische Disambiguierung

Wie in Abschnitt 3.3.1 dargestellt, umfasst die Satzanalyse die „Beschreibung des syntaktischen Baus von Sätzen durch Ermittlung elementarer Grundeinheiten wie Morphem, Wort, Satzglied und ihre Beziehung untereinander“ (Bußmann, 1983, S. 445). In dieser Arbeit wird dazu das *parse*-Modul verwendet, das Bestandteil vom *Stanford CoreNLP* ist. Hierbei handelt es sich um einen probabilistischen NL-*Parser*, der eine vollständige syntaktische Analyse (sowohl auf Basis von Konstituenten- als auch Dependenzgrammatiken) unterstützt (Manning et al., 2014, S. 4). Diese syntaktischen Informationen sind an mehreren Stellen dieser Arbeit von Bedeutung: So wird neben der Erkennung von Koordinations- und Anbindungsambiguität (s. Abschnitt 2.1.2) beispielsweise das Ergebnis des SRL im Rahmen der Unvollständigkeitskompensation verbessert (s. Abschnitt 5.2.2) oder semantische Informationen der Anforderungsextraktion korrigiert (vgl. Abbildung A.2 im Anhang). Darüber hinaus wird auf Grundlage der syntaktischen Informationen die Satzvereinfachung durchgeführt.

Abbildung 5.21 zeigt einen beispielhaften Dependenzbaum für den Satz „*I want to send emails to my colleagues*“ in Anlehnung an Beispiel 2.3.1. Hervorgehoben ist das Prädikat „*send*“ sowie zugehörige Argumente. Wie zu erkennen ist, ermöglicht die Dependenzgrammatik die Darstellung der Abhängigkeit zwischen zwei Wörtern.

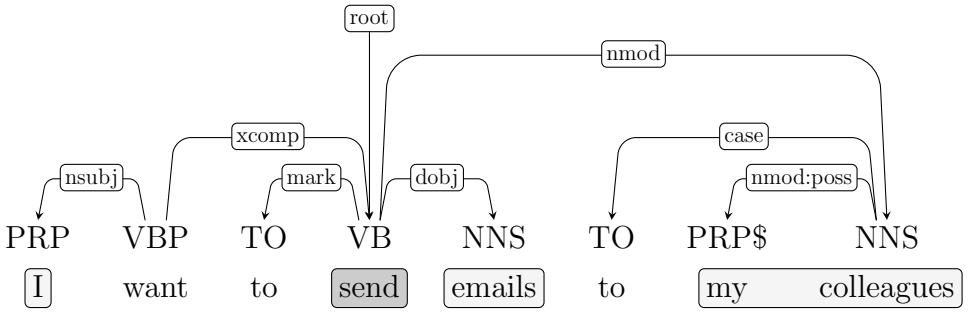
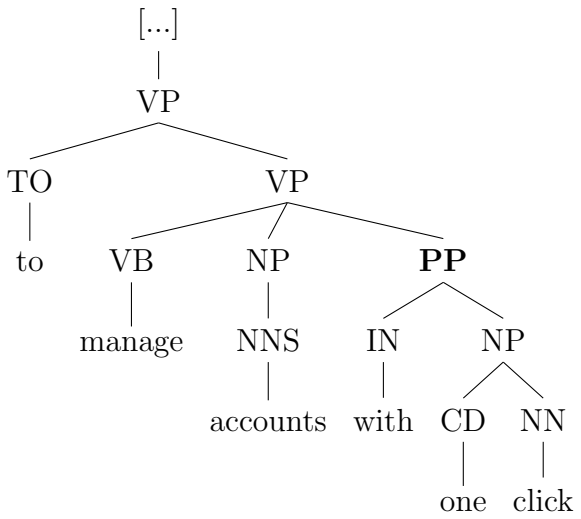


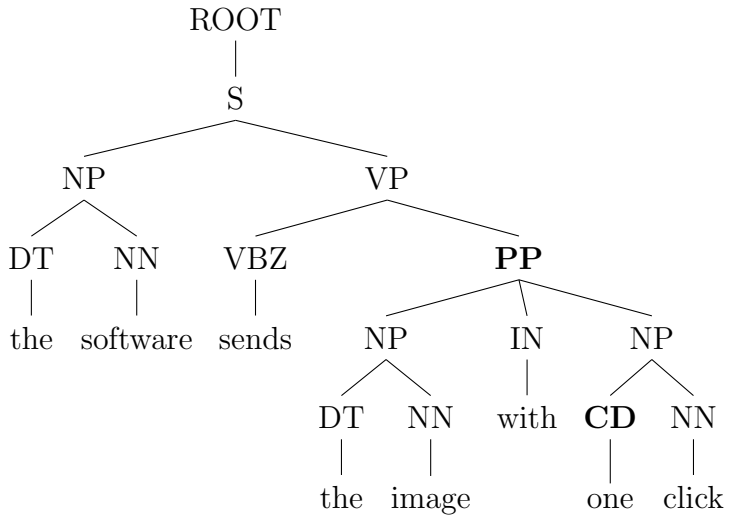
Abbildung 5.21: Beispielhafter Dependenzbaum (*Stanford CoreNLP*)

Nach Mehl et al. (1998) ist die „PP-Zuordnung [...] ein typisch computerlinguistisches Problem, weil zu seiner Lösung komplexes semantisches Wissen erforderlich ist, das in keinem sprachverarbeitenden System zur Verfügung steht“ (Mehl et al., 1998, S. 2). Bei den analysierten Parsebäumen handelt es sich demnach um die syntaktisch wahrscheinlichsten Bäume – generiert auf Grundlage von Trainingsdaten und nicht auf Grundlage von Weltwissen (vgl. Abschnitt 3.3.1.1). Nichtsdestotrotz kann der Herausforderung der **PP-Anbindungsambiguität** mittels dieser Form der Disambiguierung begegnet werden. In Abbildung 5.22 ist hierzu ein beispielhafter Parsebaum zu sehen. Grundsätzlich kann die Präpositionalphrase in diesem Fall sowohl als Konstituente der NP mit dem Kopf „*accounts*“ analysiert werden als auch als Konstituente der VP. Die Anbindung erfolgt hier an der VP. Hier sei erneut darauf hingewiesen, dass die Ergebnisse probabilistischer *Parser* nicht zwangsläufig korrekt sind (d. h. Sonderfälle existieren). Beispielhaft ist dies in Abbildung 5.23 ersichtlich, wo fälschlicherweise eine PP erkannt wurde. Diese Zuordnung könnte im genannten Fall – wenn auch spekulativ – auf die *Cardinal Number* (CD) zurückzuführen sein, die den *Parser* irritieren könnte. Wird statt „*one click*“ zum Beispiel „*a click*“ angegeben, findet die Zuordnung korrekt statt.



„*I want to manage accounts with one click.*“

Abbildung 5.22: Beispielhafter Parsebaum (*Stanford CoreNLP*)



„*The software sends the image with one click.*“

Abbildung 5.23: Fehlerhafter Parsebaum (*Stanford CoreNLP*)

Neben der PP-Anbindungsambiguität wird im gleichen Verarbeitungsschritt potentielle **Koordinationsambiguität** untersucht. Werden Indikatoren für Koordinationsambiguität innerhalb einer Anforderungsbeschreibung erkannt, gilt es, die wahrscheinlichste Lesart zu identifizieren und zu speichern. Wird beispielsweise der Satz „*I want to send large files and pictures*“ herangezogen, ist ohne Weiteres domänenspezifisches Wissen nicht zuverlässig zu disambiguieren, ob „*large*“ nur „*files*“ oder auch „*pictures*“ modifiziert. Die *parse*-Methode gibt den Dependenzbaum in Abbildung 5.24 aus. Auch diese Disambiguierung entscheidet sich dabei für die wahrscheinlichste Lesart, was im Rahmen dieser Arbeit hinreichend ist. Zu

diesem Zeitpunkt ist wichtig, dass die Entscheidung getroffen wurde, welche Lesart weitergegeben wird.

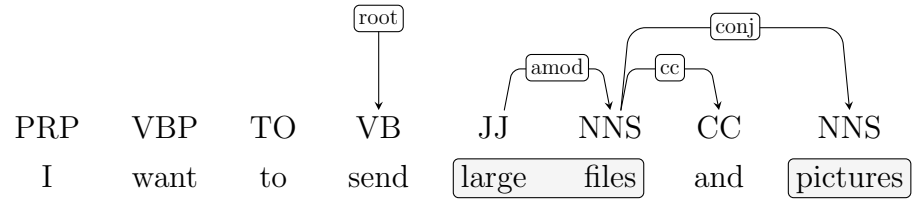


Abbildung 5.24: Koordinationsambiguität im Dependenzbaum

Als Unterthema der syntaktischen Disambiguierung wird in dieser Arbeit die **Satzvereinfachung** betrachtet. Ziel ist es, die Weiterverarbeitung komplexer Sätze zu erleichtern, indem diese in einfache Sätze übertragen werden[93], ohne die Aussage des ursprünglichen Satzes zu schädigen. Abbildung 5.25 zeigt diesbezüglich eine beispielhafte Hauptsatzreihe als Parsebaum, deren Hauptsätze S_1 und S_2 durch die Konjunktion „*and*“ verbunden und syntaktisch gleichwertig sind (Kürschner, 2008, S. 206). Gleichwertig bedeutet hier, dass die Sätze auch allein stehen könnten.

In diesem Beispiel ist eine Unterteilung der Hauptsatzreihe an der Konjunktion ausreichend, um zwei einfache Sätze zu erhalten. Wie bereits angedeutet, ist darauf zu achten, dass die Aussage des Satzes dabei nicht geschädigt wird. So stellt die Konjunktion nicht nur die syntaktische Verbindung zwischen den Sätzen her, sondern gibt auch die logische Beziehungen zwischen den Aussagen an. Konkret bedeutet das, dass beide genannten Anforderungen vom Endanwender gewünscht werden – anders als bei der Konjunktion mittels „oder“.

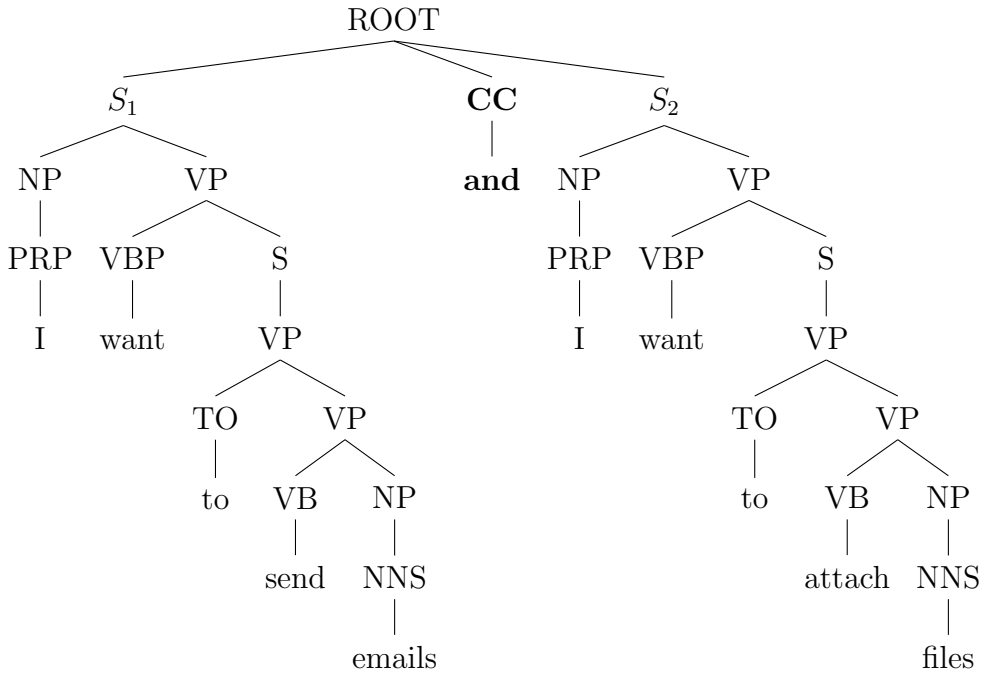
5.5.4.3 Referentielle Disambiguierung

Natürlichsprachliche Texte enthalten eine erhebliche Anzahl an Diskursreferenten und Referenzausdrücken (Bußmann, 1983, S. 32), die in der maschinellen Verarbeitung aufgelöst werden müssen. In dieser Arbeit wird die *dcoref*-Methode aus *Stanford CoreNLP* herangezogen, die zum einen eine sehr gute Performanz aufweist und zum anderen konfigurierbar und damit in einem gewissen Maße an die Softwaredomäne anpassbar ist. Darüber hinaus sind alle Ressourcen frei zugänglich.

Anders als beispielsweise bei der lexikalischen Disambiguierung findet keine Methodenanwendung auf Satzebene statt. Vielmehr wird aufgrund der Tatsache, dass Referenzen satzübergreifend auftreten, die vollständige Anforderungsbeschreibung auf Referenten und Referenzausdrücke untersucht, wenngleich auch festgehalten wird, in welchem Satz und an welcher *Token*-Position die jeweiligen Ausdrücke auftreten.

Abbildung 5.26 zeigt den Ablauf innerhalb der Komponente: Nach Anwendung der *dcoref*-Methode werden mittels eines eigenen Verfahrens weitere potentielle Referenzausdrücke gesucht. Dies ermöglicht es, festzustellen, welche Kandidaten für Referenzausdrücke noch existieren und gegebenenfalls in bestehenden Koreferenzketten aber auch in der Disambiguierung zu berücksichtigen sind. Diese können demnach beispielsweise hinzugenommen werden, um festzustellen, ob Ambiguität

[93] Mehr Informationen zu einfachen und komplexen Sätzen gibt Kürschner (2008, S. 206 ff.).



„*I want to send emails and I want to attach files.*“

Abbildung 5.25: Parsebaum mit möglicher Satzvereinfachung

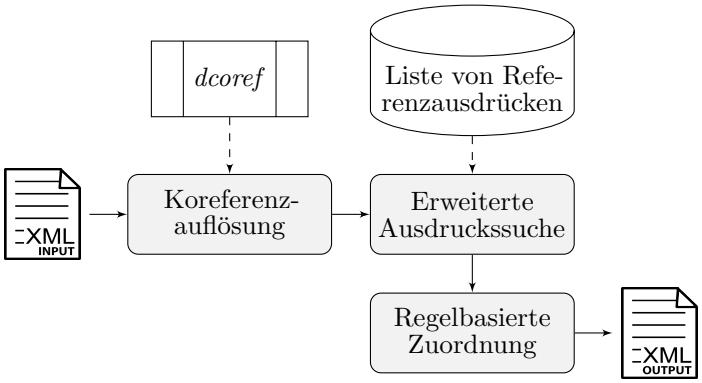


Abbildung 5.26: Auflösung von Koreferenzen

vorliegt, die gegebenenfalls zu einer falschen Koreferenzauflösung geführt hat. Ein weiteres Beispiel sind domänenspezifische Ausdrücke (z. B. „*User*“) die in bestehende Koreferenzketten (z. B. „*I*“, „*my*“) aufgenommen werden können (vgl. Abbildung A.5 im Anhang). Abbildung 5.27 stellt exemplarisch zwei Koreferenzketten, unter Angabe der gefundenen Ausdrücke (z. B. „*my*“) und ihrer Distanz im Text, dar. Es fällt auf, dass bei (A) zwei weitere Kandidaten existieren (grau hervorgehoben). Diese sind aber nicht domänenspezifisch und können sehr zuverlässig von *dcoref* als Kandidaten ausgeschlossen werden. Anders sieht dies bei (B) aus, wo die Sätze „*The emails contain many files. I want to send them.*“ in Anlehnung an Beispiel 2.1.7 der Disambiguierung zu Grunde liegen. In diesem Fall liegt referentielle Ambiguität vor,

da „*them*“ sich sowohl auf „*emails*“ als auch auf „*files*“ beziehen kann. Zwar sind die meisten Verfahren mit Heuristiken und Regeln zur Lösung dieser Ambiguitäten ausgestattet, eine Betrachtung des Kontextes und zusätzliches Hintergrundwissen (insb. Hinzunahme semantischer Kategorien) können aber zur gegebenenfalls notwendigen Korrektur hinzugezogen werden. In diesem Fall könnte die Entscheidung, dass sich „*them*“ auf „*emails*“ bezieht zum Beispiel dadurch bestätigt werden, dass in einem Satz zuvor (nicht abgebildet) „*emails*“ so wie auch „*them*“ in der semantischen Kategorie des Objekts genannt wurde.

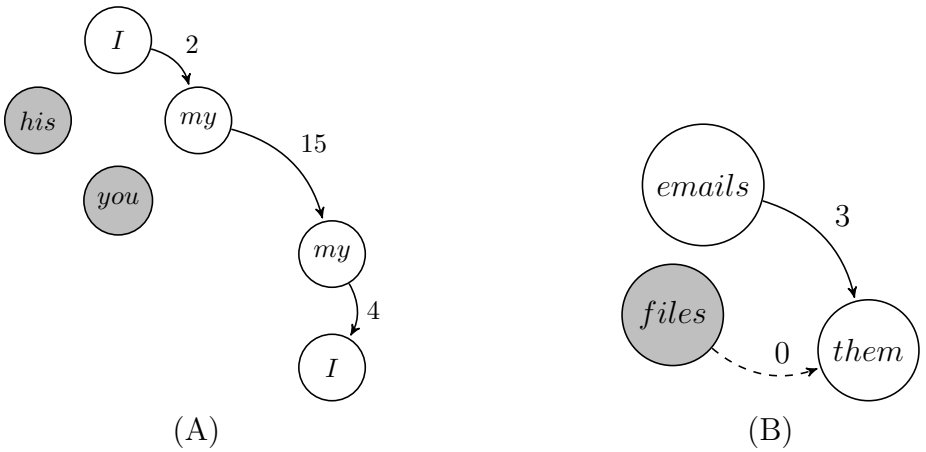


Abbildung 5.27: Koreferenzketten und Kandidaten

5.5.5 Kompensation von Unvollständigkeit

Unvollständigkeit beschreibt in dieser Arbeit die fehlende Instantiierung obligatorischer Leerstellen von Prädikaten (s. Abschnitt 2.3). Obligatorisch und damit kompensationsbedürftig ist eine Instantiierung, wenn sie zur Beschreibung einer FA wesentlichen Beitrag leistet (Bäumer und Geierhos, 2016). Um herauszufinden, wie die Argumentenstruktur einzelner Prädikate zur Beschreibung von Softwarefunktionalitäten genutzt wird, wird ein PAS-Korpus benötigt, das eine Vielzahl akquirierter Anforderungsbeschreibungen enthält, die hinsichtlich ihrer Prädikat-Argument-Struktur annotiert sind. Auf dieser Datenbasis wird ermittelt, welche prädikatspezifischen Argumente vorwiegend angegeben werden. Leerstellen, die selten instantiiert werden, werden als optional markiert und im Rahmen der Kompensation ignoriert. Weiterhin lässt sich feststellen, welche Instanzen (z. B. „E-Mail“) und Arten von Instanzen (z. B. „schriftliche Kommunikationsmittel“) mehrheitlich der Beschreibung dienen. Diese Angaben werden daraufhin genutzt, um Leerstellen zu instantiieren. Allerdings ist sicherzustellen, dass sich das kompensierte Argument auch bestmöglich in den Kontext der ursprünglichen Prädikatverwendung einbettet. Dies bedeutet, dass im Rahmen der Kompensation nicht nur das spezifische Prädikat, sondern auch der Kontext berücksichtigt werden muss (vgl. Abbildung 5.28).

Das Konzept zur Kompensation von Unvollständigkeit lässt sich in Erkennung und Kompensation unterteilen (Bäumer und Geierhos, 2016). Zu Beginn werden in der **Erkennung** die bereits vorverarbeiteten Anforderungsbeschreibungen geladen. Dies ist notwendig, da die Analyse der PAS auf Satzbasis erfolgt und dazu zuerst eine

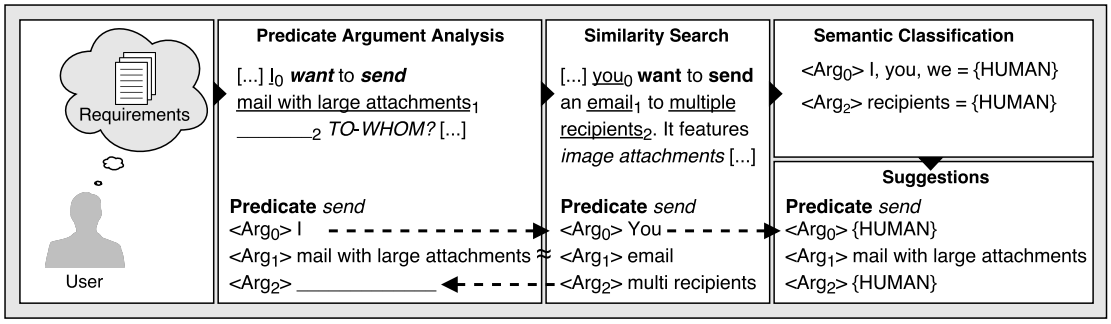


Abbildung 5.28: Prädikatbasierte Kompensation.
Entnommen aus Geierhos und Bäumer (2016, S. 40)

Satzendeerkennung durchgeführt werden muss. Darauf folgend wird die Erkennung von Prädikaten und deren Argumenten mittels SRL gestartet. Sobald ein Prädikat erkannt wird, wird geprüft (sofern es nicht auf der *Blacklist* steht), ob und welche Argumente vorliegen. Die Hinzunahme eines PAS-Korpus gibt Auskunft über die Argumentenstruktur der Prädikate und eignet sich zum Abgleich mit den durch SRL erkannten Argumenten. Wird festgestellt, dass eine Leerstelle nicht instantiiert ist, wird überprüft, ob es sich um ein obligatorisches Argument handelt und die Leerstelle somit zu füllen ist. Diese Prüfung geschieht auf Grundlage des in Abschnitt 6.2 erstellten PAS-Korpus für Anforderungsbeschreibungen. Liegt ein obligatorisches Argument vor, wird die **Kompensation** gestartet.

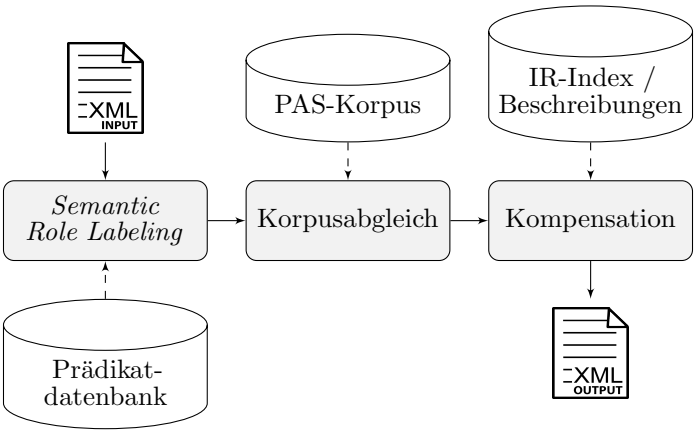


Abbildung 5.29: Erkennung und Kompensation von Unvollständigkeit

Die Kompensation nimmt unvollständige Prädikate entgegen und durchsucht das PAS-Korpus nach ähnlichen Anforderungsbeschreibungen. Die Ähnlichkeit ergibt sich sowohl aus der Ähnlichkeit der gesamten Anforderungsbeschreibung (Kontext) als auch aus der Ähnlichkeit der Sätze, in denen das Prädikat vorkommt, im Vergleich zu den Beschreibungen im Korpus. Dabei wird das Suchergebnis weiter eingeschränkt, sodass nur Anforderungsbeschreibungen zurückgegeben werden, die neben dem spezifischen Prädikat auch eine entsprechende Instanz für die betroffene Leerstelle aufweisen. Wird eine Instanz zum Prädikat mit einem ähnlichen Gesamtkontext

gefunden, wird es zusammen mit dem Prädikat und dem Satz, in dem es vorkommt, ausgegeben. An dieser Stelle sind zwei weitere Situationen denkbar:

- **Mehrere potentielle Instanzen für eine Leerstelle**
  Es können mehrere Instanzen vorgeschlagen werden, die hinsichtlich der Ähnlichkeit als gleichrangig zu betrachten sind. In diesem Fall wird die erste Instanz gewählt.

- **Keine potentiellen Instanzen**
  Es können gar keine ähnlichen Anforderungsbeschreibungen gefunden werden, womit auch keine Instanzen zurückgegeben werden können.

Darüber hinaus ist die zeitliche Gültigkeit der Datenbasis zu bedenken (s. Abschnitt 7.4.2.4). Da Sprache einem natürlichen Wandel unterliegt, ist mit neuen Wörtern bzw. einer neuen Verwendung von etablierten Wörtern zu rechnen. Die Datenbasis, die ab dem Zeitpunkt der Akquise als starr anzusehen ist, verliert somit in Teilen die Anwendbarkeit. Beispielhaft kann dies am Prädikat „*like*“ dargestellt werden, welches vor dem Siegeszug der sozialen Medien überwiegend im Sinne von „etwas mögen“ verstanden wurde und nun, beispielsweise im Duden, als „im Internet eine Schaltfläche anklicken, um eine positive Bewertung abzugeben“ (Dudenredaktion, 2016, S. 1132) geführt wird. Es ist daher erforderlich, Prädikatdatenbanken erweitern zu können und die Datenbasis insgesamt gegen eine aktualisierte Version austauschen zu können.

5.5.6 Erkennung von Vagheit

Vagheit als Form von Ungenauigkeit ist ein hochgradig relevantes Thema im RE (s. Abschnitt 2.2), wenngleich auch nicht primärer Gegenstand dieser Arbeit. Nichtsdestotrotz wird die Erkennung von Vagheit in dieser Arbeit durchgeführt, was vor allem der methodischen Abdeckung dient. Hierzu werden zwei von Geierhos und Bäumer (2017) beschriebene Testverfahren herangezogen: Der „*Intensifier test*“ und der „*Gradability test*“. Beide Verfahren greifen auf linguistische Merkmale in Anforderungsbeschreibungen zurück, um vage Lexeme zu erkennen und in Teilen zu kompensieren. Letzteres ist nicht Teil dieser Arbeit, daher ist an dieser Stelle auf Geierhos und Bäumer (2017) zu verweisen.

Der ***Intensifier test*** basiert darauf, Ausdrücke mit intensivierender Funktion (Hoffmann, 2009, S. 397) in Anforderungsbeschreibungen als Hinweis auf Vagheit zu erkennen (z. B. „*very*“ in „*very large emails*“). Intensitätspartikel werden genutzt, um die „von einem Adjektiv oder Adverb ausgedrückte Charakterisierung intensivierend-steigernd oder abschwächend-abstufend [zu] modifizieren“ (Breindl und Donalies, 2012). Der zentrale Vertreter dieser Intensitätspartikel ist dabei „sehr“ (engl. *very*), weitere sind „ausgesprochen“, „beileibe“ und „überaus“ (Hoffmann, 2009, S. 397). Eine Auflistung von Intensitätspartikeln sowie eine differenziertere Betrachtung findet sich bei Hoffmann (2009, S. 397 ff.). Geierhos und Bäumer (2017) nutzen zur Erkennung von Vagheit den Intensitätspartikel „*very*“, der nur eine einzige Bedeutung hat (demnach nicht ambig ist) und sowohl Adjektive als auch Adverben modifizieren kann. In dieser Arbeit wird zur Erkennung der folgende reguläre Ausdruck aus Geierhos und Bäumer (2017) herangezogen: „`\s(very\s[a-z].*?)\s`“

Darüber hinaus wird der ***Gradability test*** herangezogen. Hierbei wird der bereits in Abschnitt 2.2 angeführte Umstand ausgenutzt, dass die meisten Adjektive vage sind. Adjektive, die zuvor mittels POS-*Tagging* annotiert wurden, werden einem Lexikonabgleich (engl. *dictionary look-up*) unterzogen, um steigerbare Adjektive unter den erkannten Adjektiven zu identifizieren. In Anlehnung an Geierhos und Bäumer (2017) wird dabei eine modifizierte Variante des DELA-Lexikons herangezogen[94].

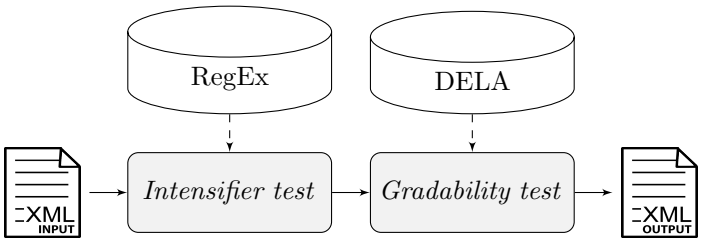


Abbildung 5.30: Erkennung von vagen Ausdrücken

Abbildung 5.30 zeigt die sequenzielle Anwendung beider Tests in einem Verarbeitungsschritt. Grundsätzlich sind dabei die dargestellten Ressourcen austauschbar bzw. erweiterbar. In der jetzigen Form können die beiden Tests auch in Kombination nur einen Hinweis auf Vagheit in Anforderungsbeschreibungen geben und sind ein erster Schritt in Richtung Vagheitsauflösung, aber von der finalen Lösung noch weit entfernt. Nichtsdestotrotz können diese Tests zum einen die Aufmerksamkeit der Endanwender auf mögliche Fehlerquellen lenken. Zum anderen werden potentiell vage Ausdrücke als solche für die weitere maschinelle Verarbeitung markiert.

5.5.7 Definition der Ausgabeformate

Der *Output* wird unterteilt in eine Ausgabe, die sich an die Endanwender richtet und somit für alle lesbar und verständlich ist. Daneben gibt es eine weitere Version, die der maschinellen Weiterverarbeitung dient (s. Abschnitt 4.1).

5.5.7.1 Ausgabe an der Benutzerschnittstelle

Die Ausgabe an der Benutzerschnittstelle muss mehrere Bedürfnisse der Endanwender für eine bessere Transparenz der angewandten Verarbeitungs- und Kompensationsschritte befriedigen. So müssen beispielsweise sowohl das Ergebnis als solches als auch Erläuterungen zu den Ergebnissen für Endanwender abrufbar sein. Die Ausgabe kann daher in drei Kategorien unterteilt werden:

- Ergebnis
- Erläuterungen zur Verarbeitung und Kompensation
- Verarbeitungs- und Kompensationsprotokoll

[94] Modifiziert bedeutet hier, dass DELA auf steigerbare Adjektive reduziert wurde.

Das **Ergebnis**, wie in Abbildung 5.31 exemplarisch zu sehen, umfasst eine Ausgabe erkannter FA, dargestellt in kontrollierter Sprache (s. Abschnitt 1.3.1) sowie Angaben zu ausgewählten Verarbeitungs- und Kompensationsschritten (z. B. deutet das Symbol der Schere auf die durchgeführte Satzvereinfachung hin).

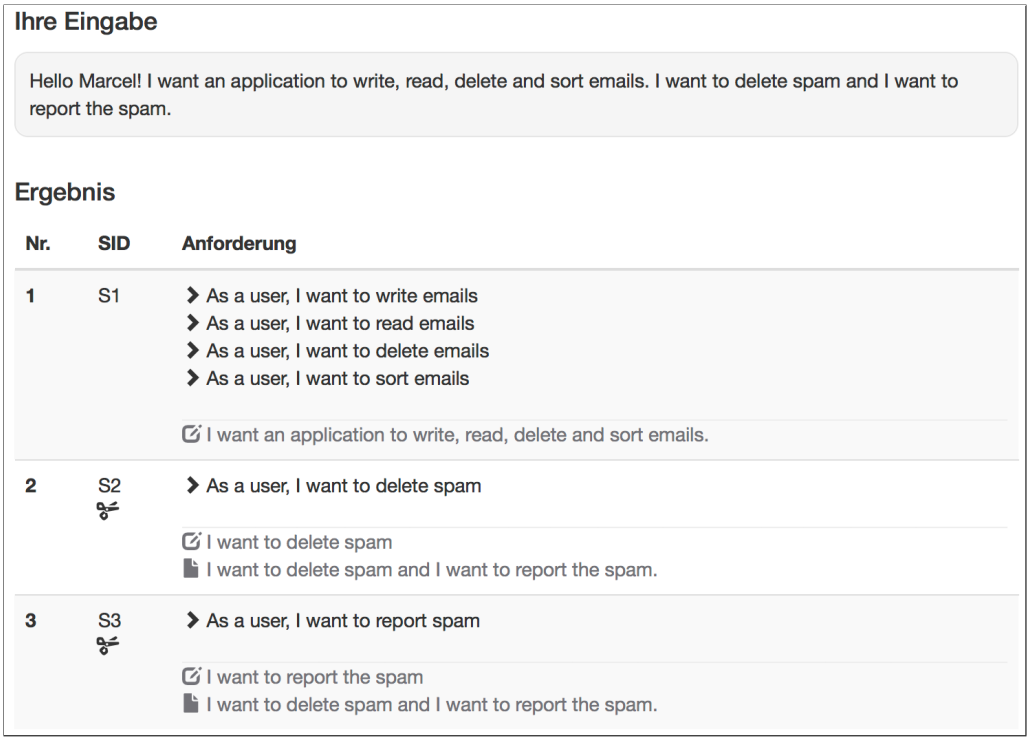


Abbildung 5.31: Ergebnisausgabe (*Frontend*)

Bei der Ausgabe in kontrollierter Sprache, die primär eine Übersicht für die Endanwender darstellt und die Frage beantworten soll, ob alle funktionalen Anforderungen vom System erkannt wurden, werden zwei Perspektiven unterstützt: Die **Nutzersicht** (z. B. *User*, Administrator etc.) und die **Systemsicht** (z. B. Anwendung, System). Die kontrollierte Syntax für die Nutzersicht ist zum Beispiel als „*As <role>, <pronoun> <priority> <action> <object>*“ definiert und orientiert sich an dem *Template* von Dollmann (2016, S. 53 f.), dargestellt in Abbildung 5.18.

Beispiel 5.5.1 (Ausgaben in kontrollierter Sprache)
Eingabe: „An administrator should be able to send and receive emails.“
→ „As an administrator$_R$, I should be able$_P$ to send$_A$ emails$_O$.“
→ „As an administrator$_R$, I should be able$_P$ to recieve$_A$ emails$_O$.“

Beispiel 5.5.1 zeigt hierfür sowohl die Eingabe in das System als auch die kontrollierte Ausgabe. Es fällt auf, dass die Rolle des Anwenders („*administrator*“) sowie die erwarteten Funktionen („*send*“, „*receive*“) mitsamt Objekt („*emails*“) übernommen wurden. Da in der kontrollierten Sprache nur eine Aktion pro Satz vorkommen darf, erstellt das System zwei kontrollierte Anforderungen aus der Eingabe. Der Grad der Normalisierung kann dabei frei bestimmt werden. In diesem Fall wird die Ausgabe lediglich durch das Pronomen „*I*“ ergänzt. Eine Normalisierung der Priorität (z. B. limitiert auf „*want*“, „*must*“) wäre denkbar.

Über das Ergebnis hinaus sind **Erläuterungen zur Verarbeitung und Kompensation** notwendig, die es den Endanwendern ermöglichen, die zuvor gesichteten Resultate besser zu verstehen. Ziel ist es, Ergebnisse einzelner Verarbeitungs- und Kompensationsschritte darzustellen. So ist es möglich, Fehler, die sich durch einzelne Schritte ergeben und das Resultat negativ beeinflussen, zu identifizieren (z. B. nicht erkannte Prädikate oder Argumente). Dargestellt werden folgende Informationen:

- Klassifikation von *On-* und *Off-Topic*
- Erkannte Sprachen
- Erkannte FA und ihre entsprechenden semantischen Informationen
- Ambige Lexeme und gewählte Lesart
- Ambige Satzstrukturen und gewählte Lesart
- Erkannte Koreferenzketten
- Unvollständige Prädikate mit komplettierten Argumenten
- Potentiell vage Ausdrücke

Diese Darstellung ermöglicht es Endanwendern, in kurzer Zeit einen Überblick über die angewandten Verarbeitungs- und Kompensationsschritte zu erhalten. Es ist dem Endanwender aber bisher, mangels einer strukturierten Gegenüberstellung, nicht möglich, einen Vergleich zwischen zwei Ergebnissen zu ziehen. Darüber hinaus wird explizit auf *Debugging*-Informationen sowie Zwischenergebnisse einzelner Schritte verzichtet. Solche sehr technischen Informationen finden sich im **Verarbeitungs- und Kompensationsprotokoll**. Dieses Protokoll stellt eine Ergänzung zu Ausgaben der Benutzerschnittstelle dar. Ziel ist es, Strategiewahl und -anwendung sowie die einzelnen Kompensationsschritte zu protokollieren, um zum einen die Nachvollziehbarkeit der angewendeten Methoden zu erhöhen und zum anderen den Vergleich zwischen Kompensationsergebnissen zu ermöglichen. Letzteres ist dem modularen Aufbau der Informationsverarbeitung geschuldet, der einen einfachen Austausch von Komponenten vorsieht und unweigerlich zu der Frage führt, welche Veränderungen im Gesamtergebnis sowie in den Ergebnissen der einzelnen Komponenten durch eine veränderte Strategie bewirkt worden sind. Das Protokoll umfasst neben den bereits zuvor aufgeführten Resultaten:

- Merkmale der Strategiewahl
- Gewählte Strategie(n)
- Zwischenergebnisse einzelner Verarbeitungs- und Kompensationsschritte
- Einzelentscheidungen regelbasierter Verfahren (z. B. Vagheitserkennung)
- Fehlerprotokolle

Allerdings eignet sich das Verarbeitungs- und Kompensationsprotokoll nicht für die maschinelle Weiterverarbeitung, da es zwar einzelne Verarbeitungsschritte protokolliert, aber nicht sämtliche Ergebnisse enthält und auch nicht in einem maschinenlesbaren Dateiformat vorliegt[95].

5.5.7.2 Maschinenlesbare Ausgabe

Grundsätzlich ist bei der maschinellen Ausgabe zuerst an die strukturierte Ausgabe der Verarbeitungs- und Kompensationsergebnisse zu denken. Doch darüber hinaus existieren noch zwei weitere denkbare Fälle der strukturierten Ausgabe:

- Zeitanalyse aller Komponenten (s. Abschnitt 7.3.2.5)
- Serialisierung der Zwischenergebnisse (s. Abschnitt 7.3.3.1)

Grundlage der maschinenlesbaren Ergebnisausgabe bildet die Anforderungsextraktion. Um die Weiterverarbeitung durch Drittanwendungen zu ermöglichen, wird die ursprüngliche Anforderungsbeschreibung, ergänzt um die extrahierten Anforderungen und Kompensationergebnisse, im XML-Format[96] ausgegeben. Ein Auszug einer solchen Ausgabe ist in Beispiel 5.5.2 abgebildet, während weitere Details der Umsetzung in der Implementierung zu finden sind (s. Abschnitt 7.3.2.5).

Beispiel 5.5.2 (Strukturierte Ausgabe, Auszug)

```
<description id=„1“ timestamp=„2017-02-18 15:30:29.461“>
 <original>I want to send large emails.</original>
 <coreference/>
  <sentences>
   <sentence lang=„en“ ontopic=„true“ sid=„1“/>
    <wsd>
     <token BabelNetURL=„http://babelnet.org/rdf/s00093485v“ .../>
     <token BabelNetURL=„http://babelnet.org/rdf/s00029345n“ .../>
    </wsd>
    <vagueness>
     <token POS=„ADJ“ TokenOffset=„5“ rule RuleID=„2“/>
    </vagueness>
    <srl>
     <pred ArgIst=„0“ ArgIstTotal=„2“ ArgSoll=„5“ Sense=„want.01“/>
  [...]
</description>
```

[95] Maschinenlesbarkeit bezeichnet hier insb. ein strukturiertes Dateiformat, welches ein automatisiertes *Parsing* ermöglicht. XML gilt hier als Standard (Hammer und Bensmann, 2011, S. 113).

[96] Im Gegensatz zur datenorientierten *JavaScript Object Notation* (JSON), das eine bessere Lesbarkeit durch Menschen und Maschinen sowie eine kleinere Dateigröße verspricht (Crockford, 2006), können mit XML natürlichsprachliche Texte in semi-strukturierte Dokumente übertragen werden (Mehler und Lobin, 2004, S. 3 f.). Die Möglichkeit, mit XML *Inline*-Annotationen vornehmen zu können, ist wesentlicher Bestandteil der automatischen Textverarbeitung.

Die Zeitanalyse dient insbesondere der Evaluation des Gesamtsystems. Zum Beispiel lassen sich somit Komponenten identifizieren, die eine überdurchschnittlich lange Verarbeitungszeit aufweisen. Um eine ausführliche Analyse zu ermöglichen, sind dabei nicht nur die Gesamtausführungszeiten der Komponenten anzugeben, sondern die Ausführungszeiten der einzelnen Verarbeitungsschritte sowie der Strategien und Indikatoren. Zusätzlich ist die in Anspruch genommene Zeit der Komponenteninitialisierung abzubilden (s. Abschnitt 7.3.2.5).

5.5.8 Analyse möglicher Verarbeitungsfehler

Die zu erwartenden Ergebnisse werden insbesondere von zwei Gegebenheiten beeinflusst: Zum einen wird von qualitativ stark schwankenden Anforderungsbeschreibungen ausgegangen, welche die Ergebnisse aller Komponenten maßgeblich beeinflussen. Zum anderen handelt es sich um ein Konzept, das auf einer Vielzahl heterogener Verfahren beruht und diese kombiniert. Die Komponenten sind demnach hinsichtlich ihrer Ergebnisse nicht gänzlich isoliert zu betrachten. Im Folgenden werden mögliche Verarbeitungsfehler der Einzelkomponenten skizziert.

Am Anfang der Informationsverarbeitung steht die **Textvorverarbeitung** (auch: *Preprocessing*), welche die initiale Anforderungsbeschreibung als Eingabe entgegennimmt und relevante (*On-Topic*) Sätze ausgibt. Die Komponente kann als sehr robust angesehen werden, da viele der genutzten Einzelkomponenten vielfach erprobt sind und zum Standard beinahe aller NLP-Anwendungen gehören. So sind beispielsweise bei der Normalisierung keine Fehler zu erwarten. Vielmehr werden einzelne Zeichen zuverlässig aus dem Fließtext entfernt und durch normalisierte Zeichen ersetzt. Die darauf aufbauende Sprachenidentifikation erreicht Evaluationswerte von 99% und kann auch mit Fachsprache umgehen, sofern eine Mindestlänge erreicht wird (s. Anhang C.1), wovon in den meisten Fällen auszugehen ist (s. Abschnitt 6.1). Die Satzendeerkennung gilt zwar ebenfalls als robust (s. Anhang C.1), kann aber auf Grund der geringen Textstrukturierung zu Fehlern führen (Read et al., 2012a). Wird beispielsweise eine Anforderungsbeschreibung gänzlich ohne Satzzeichen verfasst, wird die Erkennung von Satzenden zwar nicht unmöglich aber erheblich erschwert (Ho et al., 2016). Dieser Fehler hat Auswirkungen auf alle Folgekomponenten, da bereits die Klassifikation von Anforderungen auf Satzbasis agiert.

Die Komponente zur **Anforderungsidentifikation** nutzt REaCT zur Klassifikation von *On-* und *Off-Topic*-Inhalten, welches eine hohe Treffergenauigkeit erreicht (Dollmann und Geierhos, 2016). Allerdings geht mit fälschlicherweise als nebensächlich klassifizierten Sätzen ein erheblicher Informationsverlust einher, da diese nicht an Folgekomponenten weitergegeben werden (vgl. Abbildung 5.14). Die ebenfalls in REaCT befindliche **Anforderungsextraktion** ist ein wichtiger Bestandteil dieser Arbeit, da die Extraktionsergebnisse beispielsweise bei den Indikatoren zum Einsatz kommen und die Grundlage der strukturierten Ausgabe bilden. Dollmann (2016, S. 79 ff.) merkt an, dass die Evaluationsergebnisse mit einem durchschnittlichen F_1-*Score* von 72,66%, darauf hindeuten, dass eine zuverlässige vollautomatische IE aus den Anforderungsbeschreibungen nicht erreicht werden kann. Allerdings liegt dieser Wert nah am menschlichen Vergleichswert von 80% (Dollmann und Geierhos, 2016), sodass dieser zum jetzigen Zeitpunkt ausreichen muss. Neben der Möglichkeit, die

Klassifikationsqualität durch Erweiterung der Datenbasis zu steigern, sind Strategien zu bedenken, die falsche Klassifikationen regelbasiert erkennen und kompensieren.

Sehr gute Ergebnisse können bei der **lexikalischen Disambiguierung** durch *Babelfy* erwartet werden (Moro et al., 2014b, S. 239 ff.), wobei die Ergebnisse als vorläufig zu betrachten sind, da die Möglichkeit besteht, dass einzelne Lexeme nicht erkannt wurden oder dass eine Entscheidung über die Auflösung von Mehrwortlexemen getroffen werden muss. Darüber hinaus müssen fehlerhafte Disambiguierungen, die aufgrund von minimalem Kontext getroffen wurden, regelbasiert aufgelöst werden.

Die **syntaktische Disambiguierung** basiert sowohl bei der Disambiguierung von PP-Anbindungen als auch im Fall der Koordinationsambiguität auf dem *parse*-Modul des *Stanford CoreNLPs*. Aufgrund der Tatsache, dass es sich um einen probabilistischen *Parser* handelt, der sein Sprachwissen aus einer Menge annotierter Sätze ableitet, ist mit Fehlern in den Ergebnissen zu rechnen. „*These statistical parsers still make some mistakes, but commonly work rather well*“[97]. Da die Kompensationsmethoden nicht im Mittelpunkt dieser Arbeit stehen, wird nicht weiter darauf eingegangen. Allerdings greift beispielsweise die Kompensation von Unvollständigkeit auf die Ergebnisse dieser Expertenkomponente („*Expert first*“, vgl. Abschnitt 5.2) zurück, wodurch Fehler weitreichende Folgen für die Ergebnisqualität haben können.

Die **Erkennung von Referenzausdrücken** und das Bilden von Koreferenzketten mittels *Stanford dcoref* funktioniert zuverlässig – allerdings bedarf es weitreichender domänenspezifischer Regeln, um referentielle Ambiguität zu erkennen und aufzulösen. Zwar kann *Stanford dcoref* auf eine Vielzahl an Regeln zur Auflösung zurückgreifen, es fehlt aber dennoch an domänenspezifischem Wissen, sodass die zu erwarteten Ergebnisse im Ganzen zufriedenstellend aber dennoch fehlerhaft sind. Die lexikalische Disambiguierung kann zusätzliches Wissen zur referentiellen Disambiguierung bereitstellen. Auch dieses muss durch Regeln eingebunden werden.

Die **Kompensation von Unvollständigkeit** basiert in dieser Arbeit auf dem Abgleich lückenhafter Anforderungen mit ähnlichen Anforderungen, mit dem Ziel, die nicht-instatiierten Leerstellen zu füllen. Es ist zu erwarten, dass dies bei hochfrequenten Prädikaten zufriedenstellend geschieht. Allerdings kann nicht mit abschließender Sicherheit bestimmt werden, ob eine gewählte Instanz im Anwendungsszenario des Anwenders stimmig ist. Demgegenüber ist die Kompensation bei nicht frequenten Prädikaten als problematisch zu bewerten. Hier fehlt es schlicht an Beschreibungen im Anforderungsindex. Die durchgeführte Analyse zeigt auf, dass Verarbeitungsfehler zu erwarten sind. Ein transparenter und kritischer Umgang mit möglichen Fehlern ist dabei wichtig für die Weiterentwicklung und die Systemqualität. Bezüglich der Qualität werden in Abschnitt 7.4 weitere Anforderungen erläutert. Im Folgenden wird ein Überblick der bisherigen Arbeit in Form eines Zwischenfazits gegeben, um Implementierungsherausforderungen zu identifizieren und daraufhin in den Implementierungsteil dieser Arbeit übergehen zu können.

[97] Siehe weiterführend: `http://nlp.stanford.edu/software/lex-parser.shtml` (Stand: 11.01.17).

5.6 Zwischenfazit und Ausblick

Aufgrund der zu erwartenden Qualitätsschwankungen in den Anforderungsbeschreibungen ist die Anwendung verschiedener Verarbeitungskomponenten zur Qualitätsverbesserung notwendig (s. Abschnitt 1.4). Diese vielfältigen Komponenten umfassen dabei sowohl Verfahren des *Preprocessings* als auch Kompensationsverfahren wie die Unvollständigkeitskompensation oder Disambiguierungsansätze. Solche Verfahren existieren in den meisten Fällen bereits, sind aber überwiegend für die isolierte Anwendung ausgelegt und unterscheiden sich hinsichtlich Eingabe- und Ausgabeparametern, Ressourcen und schlussendlich auch in der Laufzeit (s. Abschnitt 3.4).

Um die Komponentenauswahl auf gegebene Anforderungsbeschreibungen anzupassen, werden Indikatoren definiert und herangezogen, die jeweils bestimmte Defizite in den Anforderungsbeschreibungen repräsentieren und bei positiver Erkennung die Ausführung entsprechender Verarbeitungskomponenten rechtfertigen (s. Abschnitt 5.3). Die Indikatoren können dabei nicht auf die Ergebnisse der nachgelagerten Verfahren zurückgreifen und sind daher von eigenen Regeln (z. B. syntaktische Muster), Verfahren (z. B. *Chunker*) und Ressourcen (z. B. WordNet) abhängig. Auf Grundlage der Indikatoren erfolgt die Auswahl der Strategien, wobei zwischen vordefinieren und einer automatischen *Fallback*-Strategie zu unterscheiden ist. Die Strategien steuern die notwendigen Verarbeitungskomponenten, koordinieren Möglichkeiten für Synergien und lösen Abhängigkeiten auf. Sie beeinflussen die Verarbeitung somit weitgehend (z. B. Ergebnisse zusammenführen).

Die Verarbeitungskomponenten sind in Abschnitt 5.5 konzipiert worden. Sie bestehen überwiegend aus einer Hauptkomponente (z. B. *Stanford CoreNLP*) und werden um zusätzliche Softwarekomponenten (z. B. Abgleich mit *Blacklist*) erweitert. Auch, um die Kompatibilität zwischen den einzelnen Verarbeitungskomponenten sicherzustellen. Hier sieht das Konzept ein globales Datenobjekt vor (s. Abschnitt 7.3.2.1), dass von allen Komponenten bearbeitet wird, wofür unter anderem Konvertierungsprozesse notwendig sind. Aufgrund der unterschiedlichen Softwarearchitekturen der Komponenten entstehen vielfältige Herausforderungen für die Implementierung (sowohl aus Software- als auch aus Ressourcensicht).

Die für das beschriebene Softwarekonzept notwendigen Ressourcen sind hinsichtlich ihrer Art und Thematik sehr unterschiedlich und liegen überwiegend nicht (in benötiger Struktur bzw. Umfang) vor (s. Abschnitt 4.1.3). Aus diesem Grund beginnt die Implementierung mit der Ressourcenerstellung, was insbesondere die Erstellung des Anforderungsbeschreibungskorpus und des PAS-Korpus umfasst (s. Kapitel 6).

Es folgt die softwaretechnische Implementierung, die von der Frage getrieben wird, welche Softwarearchitektur geeignet ist, um zum einen die Anforderungen der Endanwender zu erfüllen (leichte Bedienbarkeit, Plattformunabhängigkeit, geringes technisches Vorwissen erforderlich) und zum anderen die indikatorbasierte Strategieanwendung und Beschreibungskompensation zu ermöglichen (s. Kapitel 7). Letzteres ist aufgrund der unterschiedlichen Verarbeitungskomponenten (z. B. gewählte Programmiersprachen, Softwarearchitekturen, Netzwerkkommunikation, externe Ressourcen) als nennenswerte Herausforderung zu verstehen.

Teil III

Implementierung und Evaluation

6 Ressourcen

Um die Anforderungsextraktion sowie die Kompensation von Ambiguität und Unvollständigkeit in Anforderungsbeschreibungen zu ermöglichen, sind linguistische Ressourcen notwendig (s. Abschnitt 4.1.3). Deren Umfang und Aufbau unterscheiden sich je nach Anwendung. Im Bereich des REs herrscht ein Mangel an entsprechenden Ressourcen, wie beispielsweise Tichy et al. (2015, S. 161) darstellen. Aus diesem Grund ist es im Rahmen dieser Arbeit notwendig, ergänzende Ressourcen zu Konzeptions-, Test- und Evaluationszwecken aufzubauen. Im Folgenden werden elementare Ressourcen, namentlich das Anforderungsbeschreibungskorpus (s. Abschnitt 6.1) sowie das Prädikat-Argument-Struktur-Korpus (s. Abschnitt 6.2), vorgestellt.

6.1 Anforderungsbeschreibungskorpus

Wie in Abschnitt 4.1.3 dargestellt, ist eine Sammlung von Anforderungsbeschreibungen in dieser Arbeit von besonderer Relevanz, da sie zum einen für Testzwecke und zur Evaluation benötigt wird und zum anderen die Ableitung spezifischer Charakteristika des Textgenres ermöglicht (z. B. Vokabular, Textqualität). Dies ist wiederum wichtig, um die Methoden und Komponenten des Softwaresystems bedarfsgerecht zu entwickeln und zu konfigurieren. Ein Korpus ist dabei eine „[...] endliche Menge von konkreten sprachlichen Äußerungen, die als empirische Grundlage für sprachwiss. Untersuchungen dienen. Stellenwert und Beschaffenheit [...] hängen weitgehend von den jeweils spezifischen Fragestellungen und methodischen Voraussetzungen des theoretischen Rahmens der Untersuchung ab [...]“ (Bußmann, 1983, S. 79).

Wie dargestellt, existieren nur sehr wenige und zudem kleine Textsammlungen, welche die zu erwartenden Eigenschaften von Anforderungsbeschreibungen abdecken (z. B. Dollmann, 2016), sodass eine eigene zusätzliche Akquise von Anforderungsbeschreibungen notwendig ist. Die Akquise bezieht sich in dieser Arbeit auf die Suche nach Anforderungsbeschreibungen im Web via Internet-Suchmaschinen[98] (insbesondere auf einschlägige Entwicklerplattformen). Ziel ist es, das Korpus von Dollmann (2016), welches derzeit 200 FA sowie 492 nebensächliche Sätze umfasst, um weitere 300 FA zu erweitern, um hinsichtlich der Verteilung von FA und nebensächlichen Angaben ein ausgewogenes Korpus zu erhalten. Bei der Auswahl geeigneter Beschreibungen kommt ein zweistufiges Vorgehen zum Einsatz, wie es auch Dollmann (2016, S. 49 ff.) anwendet: Zuerst werden aufgrund von Suchbegriffen sowie Phrasen (z. B. „*I want an application*“) Texte akquiriert, die mögliche Anforderungsbeschreibungen darstellen können (Kandidaten). In einem zweiten Schritt werden diese Kandidaten händisch kontrolliert, ob es sich tatsächlich um FA oder doch um Nebensächliches

[98] Gesucht wird via `https://www.google.de`.

handelt. Die auf diese Weise identifizierten Anforderungsbeschreibungen werden in den Datenbestand übernommen.

6.1.1 Datenbestand

Insgesamt umfasst der zusätzlich zu Dollmann (2016) akquirierte Datenbestand 300 FA in englischer Sprache, die in Qualität und Umfang stark variieren. Beispiel 6.1.1 zeigt eine zufällig ausgewählte Anforderungsbeschreibung[99], die von SourceForge akquiriert wurde und welche den Wunsch eines Endanwenders nach der Erweiterung einer Musikanwendung um eine Kopierfunktion von Audio-CDs beschreibt.

Beispiel 6.1.1 (Anforderungsbeschreibung)

> *I think it would be useful if this great piece of software included the ability to rip MP3 (via grip) within the program: as a user I want to be able to put a CD in and having ripped to MP3, added to the ipod all in one programme. I know this is currently possible using separate programs, but the way that the itunes program works as a one-stop shop, works really well from the usability point of view.*

Hierbei handelt es sich um eine Beschreibung, die bereits Aufschluss über die zu erwartende Qualität und über erste Merkmale gibt. So enthält sie zwar eine vermeintlich klar erkennbare FA (Kopieren von Audio-Dateien), sie weist aber dennoch eine Reihe qualitativer Defizite auf. Exemplarisch werden einige im Folgenden benannt:

Zuerst fällt auf, dass die FA eigentlich eine zusammengesetzte FA, bestehend aus drei Unteranforderungen, ist: (1) CDs sollen eingelegt werden können, (2) die CD soll in das Audioformat MP3 konvertiert und (3) Dateien sollen auf einen iPod übertragen werden. Neben dieser Tatsache finden sich bereits hier sowohl eine Reihe impliziter Annahmen (z. B. soll nicht die CD in MP3 konvertiert werden, sondern auf der CD befindliche Audiostücke) als auch Ellipsen, wie beispielsweise in „*and having ripped to MP3*“, wo das Objekt der Anforderungen ausgelassen wird.

Wenig überraschend ist darüber hinaus der große Anteil an *Off-Topic*-Informationen (z. B. „*I know this is currently possible [...]*“). Auch sind Rechtschreibfehler („*programme*“) vorzufinden, was bei UGC zu erwarten ist. Benannte Entitäten („*iTunes*“, „*iPod*“) existieren ebenfalls, wie im Rahmen der Konzeption und insbesondere im Kontext der lexikalischen Disambiguierung bereits vermutet (s. Abschnitt 5.5.4.1). Eine Besonderheit ist, dass eine Angabe zur Rolle („*as a user*“) getätigt wird. Dies ist nicht der Regelfall aber für den Verarbeitungsschritt der IE besonders wertvoll. Zu erwarten war ferner auch, dass die Anforderungen aus Sicht des *Users* verfasst werden und entsprechende Personalpronomen („*I*“) vorzufinden sind.

Von der Analyse einer einzigen Anforderungsbeschreibung auf den gesamten Datenbestand zu schließen, ist unzureichend. Deswegen stellt Tabelle 6.1 ermittelte Statistiken dar, die sich auf den gesamten Datenbestand beziehen und unter anderem Aufschluss über die Anzahl funktionaler Anforderungen sowie Wort- und Zeichenkonstellationen geben.

[99] Siehe: `https://sourceforge.net/p/gtkpod/feature-requests/42/` (Stand: 11.01.17).

Merkmal	Bäumer (2017)	Dollmann (2016)
Anzahl FA	300	200
Anzahl *Types* / *Token*	1.399 / 5.521	1.266 / 4.414
∅ *Token* / Satz	18	22
Token Min. / Max.	7 / 49	4 / 49
∅ Zeichen / Satz	99	105
Zeichen Min. / Max.	29 / 264	29 / 292

Tabelle 6.1: Zusammensetzung des Datenbestands und Merkmalsgegenüberstellung

Wie in Tabelle 6.1 ersichtlich wird, ähneln sich die Datenbestände in den angegebenen Werten wie Satzlänge in *Token* und Zeichen. Dies ist aber nur als erstes Indiz für eine Ähnlichkeit zu werten, da eine Gegenüberstellung der häufigsten Begriffe zwischen den Korpora (vgl. Frequenzliste in Tabelle 6.2) als auch eine Analyse der semantischen Kategorien noch aussteht (vgl. Tabelle 6.3).

6.1.2 Gegenüberstellung

Zur Gegenüberstellung der hier akquirierten FA und der FA aus Dollmann (2016) empfiehlt sich zuerst ein Blick auf die Wortfrequenz in den jeweiligen Korpora. Auf diese Weise können Auffälligkeiten in der Wortwahl und bei der inhaltlichen Schwerpunktsetzung der Anforderungsbeschreibungen sichtbar gemacht werden (vgl. Tabelle 6.2).

	(A) Bäumer (2017)		(B) Dollmann (2016)		(C) Dollmann (2016)*	
1.	*i*	278	***would***	122	*would*	119
2.	***software***	140	*have*	44	***it***	90
3.	*can*	77	*nice*	44	*i*	52
4.	*should*	75	***could***	40	*nice*	43
5.	***app***	61	*add*	37	*have*	41
6.	*it*	43	*like*	35	*could*	39
7.	*want*	35	***should***	28	*add*	37
8.	*file*	30	*you*	25	*like*	35
9.	***program***	28	*file*	23	*should*	28
10.	*have*	22	***can***	21	*you*	25

Tabelle 6.2: Die 10 häufigsten Begriffe in den Korpora.
In Anlehnung an Dollmann (2016, S. 52)

Tabelle 6.2 zeigt sowohl die Frequenzliste für die akquirierten Daten (A) als auch für die Daten aus Dollmann (2016), bezeichnet als (B). Da Dollmann (2016) aber keine weiteren Angaben zur Erstellung der Frequenzliste macht (insb. nicht zur Anwendung einer Stoppwortliste und Lemmatisierung), wird an dieser Stelle eine eigene Frequenzliste auf Basis der Daten von Dollmann (2016) erzeugt (C)[100], um die Vergleichbarkeit zwischen den Korpora zu sichern.

[100] Zur Anwendung kommt Lemmatisierung und die Entfernung folgender, für den Auszug relevanter, Stoppwörter: „*to*“, „*the*“, „*be*“, „*a*“, „*in*“, „*and*“, „*for*“, „*if*“, „*of*“, „*that*“, „*with*“ und „*or*“.

Es fällt auf, dass in (A) sowohl die semantische Kategorie der Rolle („*i*“) als auch der Komponente („*software*“, „*app*“, „*program*“) im Vordergrund stehen, was bei (B) sowie bei (C) nicht der Fall ist. Bezogen auf die gesamten Korpora zeigt sich, dass Korpus (A) einen höheren Anteil an Wörtern aufweist, die in die semantischen Kategorien „Rolle“ und „Komponente“ fallen[101] (vgl. Abbildungen 6.1 und 6.2). Zu diesem Zeitpunkt ist von vermeintlicher Kategoriezugehörigkeit zu sprechen, da erst die Annotation Aufschluss über die Verteilung semantischer Kategorien gibt.

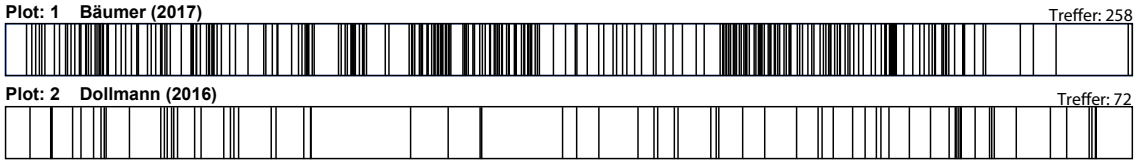


Abbildung 6.1: Wortverteilung der semantischen Kategorie „Rolle“ je Korpus

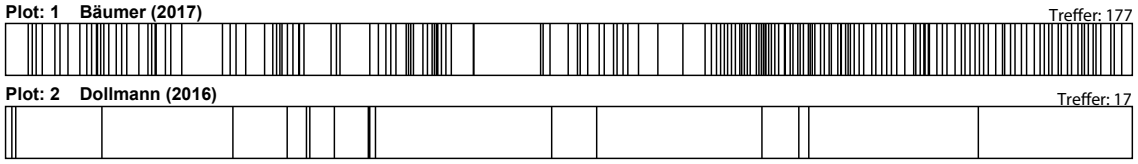


Abbildung 6.2: Wortverteilung der semantischen Kategorie „Komponente“ je Korpus

Zu erklären ist diese Auffälligkeit mit der unterschiedlichen Herkunft. Während Dollmann (2016) seine Daten ausschließlich von der Entwicklungsplattform SourceForge akquiriert, auf der Nutzer ihre FA zur Weiterentwicklung eines existierenden Produkts kommunizieren und dieses nicht mehr explizit in der FA erwähnen, beziehen sich die hier akquirierten FA überwiegend auf neu zu entwickelnde Software, welche daher sehr häufig genannt wird. Dies wird auch bei der Betrachtung frequenter Phrasen[102] deutlich: Während in Korpus (B bzw. C) gehäuft Phrasen wie „*It would be*“ oder „*Would it be possible*“ vorkommen, sind es in Korpus (A) Formulierungen wie „*I need a software*“ und „*App should be able to*“. In diesem Kontext erklärt sich auch die hohe Frequenz der Wörter „*you*“ und „*add*“ im Datenbestand von Dollmann (2016), da sie Teil von Nutzeraufforderung an die Softwareentwickler („*you*“) sind, bestimmte Softwarefunktionen zu ergänzen („*add*“): „*Can you please add exif support to the package?*“[103]. Auch erklärt sich die hohe Frequenz von „*it*“ in Korpus (C), was ursprünglich in Dollmann (2016) als Stoppwort entfernt wurde und oftmals (aber nicht immer) als Referenzausdruck zur bestehenden Softwareapplikation genutzt wird. Mit Bezug zum OTF-*Computing* sind Formulierungen wie in Korpus (A) zu erwarten, da Endanwender mit ihren FA auf eine noch nicht existente Software referenzieren.

Dollmann (2016) merkt bezüglich der Wortfrequenz in seinem Korpus (B) an, dass die semantische Kategorie „Priorität“ unter den frequentesten *Token* stark vertreten ist („*would*“, „*could*“ und „*should*“). Diese Auffälligkeit bleibt auch bei erneuter

[101] Untersucht wurden „*I*“, „*my*“, „*mine*“, „*we*“, „*User*“ bzw. „*Software*“, „*System*“, „*App*“, „*Application*“, „*Tool*“ und „*Program*“.

[102] Erzeugt wurden Frequenzlisten von Tetragrammen auf beiden Korpora.

[103] Siehe: `https://sourceforge.net/p/graphics32/feature-requests/21/` (Stand: 04.02.17).

Erstellung der Frequenzliste in (C) bestehen und findet sich darüber hinaus auch in (A). Dies bedeutet, dass die Priorität in beiden Korpora eine wichtige Funktion einnimmt. Diese Erkenntnis ist deckungsgleich mit den Ergebnissen aus Dollmann (2016, S. 55) bzw. in Tabelle 6.3 (277 bzw. 209 Annotationen).

	(A) Bäumer (2017)	**(B) Dollmann (2016)**
Komponente	**232**	84
Komponentenverfeinerung	26	16
Aktion	**333**	**204**
Argument der Aktion	143	104
Bedingung	30	39
Priorität	**277**	**209**
Motivation	22	19
Rolle	259	42
Objekt	439	**195**
Verfeinerung des Objektes	127	48
$\sum$	1.888	960

Tabelle 6.3: Anzahl annotierter Hauptinformationen nach Kategorie. In Anlehnung an Dollmann (2016, S. 55)

Bezüglich der semantischen Kategorien finden sich in Tabelle 6.3 weiterführende Angaben. So fällt bei den von Dollmann (2016) annotierten FA auf, dass mehr Aktionen und Prioritäten annotiert wurden, als FA vorhanden sind. Scheinbar existieren demnach FA, die mehrere Aktionen und Prioritäten aufweisen. Dies ist nicht ungewöhnlich, entspricht aber nicht der oftmals formulierten Empfehlung im RE, Anforderungen atomar zu formulieren (Pohl und Rupp, 2015, S. 48). Das gleiche Bild zeichnet sich für die semantische Kategorie der Aktionen bei (A) ab und entspricht wahrscheinlich auch dem, was im Kontext des OTF-*Computings* zu erwarten ist, da nicht davon auszugehen ist, dass sich Endanwender an RE-Empfehlungen halten, geschweige denn diese kennen.

Auch lässt sich feststellen, dass die Rolle und die Komponente in Korpus (A) erheblich häufiger vorkommen, als es in Korpus (B) der Fall ist. Dies wurde bereits zuvor auf Basis der Wortfrequenz vermutet. Eine Erklärung hierfür ist die unterschiedliche Datenherkunft. Bei Dollmann (2016) ist sowohl die Kategorie „Komponente“ als auch „Rolle“ innerhalb der FA bereits durch den Kontext (Entwicklungsforum zu bestehenden Produkten) vorgegeben. Weiterhin fällt auf, dass bei Korpus (B) weniger Objekte annotiert wurden, als es FA gibt, wohingegen sich im Korpus (A) 439 Objekte in 300 FA finden. Auch hier gilt, dass die Datenherkunft zu beachten ist, wie folgendes Beispiel aus Korpus (A) aufzeigt: „*Make it possible to import/export themes*“[104] enthält zwei Angaben zur semantischen Kategorie der Aktion, bezieht sich dabei aber nur auf ein Objekt. Demgegenüber stehen Beispiele wie „*I need a good Karoake Software which can remove vocals from mp3 completely and save that*

[104]Siehe: `https://sourceforge.net/p/cmsworks/feature-requests/11/` (Stand: 12.02.17).

karoake file as mp3“[105] aus Korpus (B), in dem mehrere Aktionen und Objekte genannt werden und die sehr häufig in dieser Form vorzufinden sind.

Schlussendlich lässt sich nach dieser Gegenüberstellung zusammenfassen, dass die akquirierten FA ähnlich in Qualität und Umfang zu den Anforderungen sind, die seitens Dollmann (2016) bereitgestellt werden und sich dennoch in wesentlichen Merkmalen unterscheiden (z. B. Häufigkeit semantischer Kategorien). Diese Unterschiede sind dabei von erheblicher Bedeutung für diese Arbeit, da sie zum einen aufzeigen, wie stark natürlichsprachliche Anforderungsbeschreibungen und insbesondere FA in Umfang und Qualität variieren. Zum anderen ermöglichen sie eine umfassende Evaluation des Systems. Durch die Zusammenführung der beiden Datenbestände wird versucht, eine möglichst breite Abdeckung an Formulierungen zu erreichen. Auf diese Weise wird sich den Anforderungsbeschreibungen, die im OTF-*Computing* zu erwarten sind, durch vergleichbare Eigenschaften angenähert.

6.2 Prädikat-Argument-Struktur-Korpus

Das PAS-Korpus enthält Fließtexte zur Beschreibung von Softwarefunktionalitäten. Es dient zum einen dazu, die Frequenz und die Art der domänenspezifischen Prädikatverwendung zu analysieren. Zum anderen wird es benötigt, um instantiierte Argumentpositionen von Prädikaten im jeweiligen Kontext zu extrahieren und für die Kompensation von unvollständigen Angaben in den Anforderungsbeschreibungen des Endanwenders zu nutzen. Aus diesem Grund werden die Texte nach der Akquise weiterverarbeitet (z. B. Satzgrenzenerkennung) und durch weitere Daten (z. B. Hyperonyme) angereichert. Dieses Korpus wird beispielsweise in Bäumer und Geierhos (2016) zur Kompensation von Unvollständigkeit und in Geierhos und Bäumer (2017) zur Erkennung und Kompensation von Vagheit herangezogen.

6.2.1 Datenakquise und -vorverarbeitung

Das Korpus speist sich aus den Daten der Onlineplattform download.com[106], die im Zeitraum von 01. Januar bis 01. Februar 2016 mittels einer eigens entwickelten *Crawler*-Applikation automatisiert heruntergeladen wurden und den Zeitraum von Februar 1995 bis Februar 2016 abdecken. Die Daten umfassen neben Programmbeschreibungen der Hersteller (vgl. Beispiel 6.2.1) auch Bewertungen der Plattformbetreiber sowie Kommentare und Bewertungen von Anwendern. Der so entstandene Ausgangsdatensatz beinhaltet somit überwiegend Texte, in denen die Softwarefunktionalitäten im Mittelpunkt stehen (vgl. Beispiel 6.2.1). Die Qualität der Programmbeschreibungen ist dabei schwankend und reicht von einer reinen Aufzählung von Funktionalitäten bis hin zu ausgeschmückten Werbetexten. Um die Datenqualität zu erhöhen, ist ein *Preprocessing* notwendig, das im folgenden Abschnitt 6.2.1 erläutert wird.

Insgesamt enthält der Datensatz dabei 193.641 Datensätze, aufgeteilt in 23 Softwarekategorien (z. B. Spiele, Sicherheit, Kommunikation) und 253 Unterkategorien (z. B. Arkade-Spiele, Antivirus, E-Mail Software). Die Angaben zu Kategorien und

[105] Siehe: `http://answers.yahoo.com/rss/question?qid=20120427101849AA4y0wl` (Stand: 12.02.17).
[106] Siehe weiterführend: `http://download.cnet.com` (Stand: 11.01.17).

Unterkategorien sind für eine spätere Analyse der Prädikate im Hinblick auf die unterschiedliche Verwendung innerhalb der Softwarekategorien von Interesse.

Beispiel 6.2.1 (Programmbeschreibung, gekürzt)

> *Email Scheduler Tracker is an easy-to-use e-mail management software with which you can: Send e-mails to customers, prospects, webinar participants, etc. in either plain text or HTML text, either immediately or at any time and date in the future. Send automatic reminders to one or more people [...]*

Die akquirierten Programmbeschreibungen werden in mehreren sequenziellen Schritten zur Verbesserung der Datenqualität modifiziert. Dabei liegt der Fokus auf Erkennung und Abbildung der PAS einzelner Sätze, was unter anderem dazu führt, dass die bisherigen komplexen Satzkonstruktionen aufgetrennt und als einzelne elementare Sätze gespeichert werden. Wichtig ist, dass zu jedem Zeitpunkt eine eindeutige Zuordnung von Sätzen und Prädikaten zum Ursprungstext möglich ist. Hierfür werden die folgenden Schritte durchlaufen:

(1) Entfernung von *Hypertext Markup Language* (HTML)
Es können vereinzelt HTML-Auszeichnungen im Text existieren, die in diesem Schritt entfernt werden (z. B. „<b>“), da sie nachfolgende Schritte, wie beispielsweise die Satzendeerkennung, behindern können[107].

(2) Satzendeerkennung
Die Satzendeerkennung unterteilt den Fließtext in einzelne Sätze. Dabei ist die Erkennung von Satzenden wesentlich komplexer als das einfache Erkennen von Interpunktionszeichen (s. Anhang C.1.2). Dies ist zum Beispiel dann der Fall, wenn Aufzählungen in Programmbeschreibungen genutzt werden[108].

(3) Erkennung der Prädikat-Argument-Struktur
Die Erkennung von Prädikaten (z. B. „*send*“) und zugehörigen Argumenten (z. B. „*mailing list*“) in Sätzen wird in diesem Schritt durch einen *Semantic Role Labeler* durchgeführt[109]. Die so gewonnenen Erkenntnisse werden zusammen mit dem jeweiligen Ausgangssatz gespeichert.

(4) Anreicherung semantischer und lexikalischer Beziehungen
In diesem Schritt werden sowohl Prädikate als auch Argumente um eindeutige semantische und lexikalische Beziehungen erweitert[110].

(5) Überführung in ein strukturiertes Ausgabeformat
Die Ausgabe erfolgt hierarchisch strukturiert (Baumstruktur) unter der Nutzung

[107] Die Entfernung erfolgt mittels „jsoup“. Siehe: `http://www.jsoup.org` (Stand: 12.01.17).
[108] Die Erkennung erfolgt mittels „LingPipe“. Siehe: `http://www.alias-i.com/` (Stand: 12.01.17).
[109] Genutzt wird das *Toolkit* „Mate Tools“ (Björkelund et al., 2010).
[110] Erweiterung durch WordNet-IDs. Siehe: `http://wordnetweb.princeton.edu` (Stand: 12.01.17).

der *Extensible Markup Language* (XML). Ausgehend vom jeweiligen Prädikat als Wurzelknoten auf der höchsten Ebene wird sowohl der Satz als auch die erkannten Argumente des Prädikats als Folgeknoten geführt (vgl. Beispiel 6.2.2).

Beispiel 6.2.2 (Ausgabeformat)

<predicate sense="*send.01*" wordnet="*01033289*">
<sentence id="*75891832-2*" sid="*2*" text="*Send e-mails to customers, prospects, webinar participants, etc. in either plain text or HTML text, either immediately or at any time and date in the future.*" tid="*75891832*" version="*1*" path="*downloadcom::communication::email*" alias="*email*">
<arguments>
<arg id="*A1*" wordnet="*06289979*" type="">*e-mails*</arg>
<arg id="*A2*" wordnet="" type="">*customers, prospects, webinar participants*</arg>
</arguments>
</sentence>
</predicate>

Zusatzinformationen wie IDs (z. B. „sid“) sowie semantische und lexikalische Beziehungen („wordnet“) sind als Attribute angegeben. Vor- und Nachteile von XML zur Strukturierung von Korpora werden von Naumann (2003) diskutiert. Zusammengefasst eignet sich die XML-Auszeichnungssprache aufgrund der Standardisierung und der weiten Verbreitung sowie der Verfügbarkeit von Anwendungen zur Verwaltung und Modifikation (Naumann, 2003, S. 379 f.).

6.2.2 Zusammensetzung

Die Zusammensetzung des Korpus ist von Interesse, da sich dadurch die Notwendigkeit von Vor- und Nachbearbeitungsschritten aufgrund von spezifischen Texteigenschaften ergibt (z. B. viele Rechtschreibfehler).

Die durchschnittliche Textlänge der Programmbeschreibungen über alle Kategorien hinweg beträgt 129 Wörter (737 Zeichen). Um spezifische Texteigenschaften zu erkennen, wird eine 300 Sätze umfassende Zufallsstichprobe in der Kategorie „*E-Mail Software / Utilities*“ erstellt und analysiert (Gesamtumfang 2.088 Beschreibungen; 11.187 Sätze). Als positiv ist die hohe Erkennungsrate der Satzgrenzenerkennung zu bewerten. Bei 300 Sätzen sind 274 Satzgrenzen (91,3 %) korrekt erkannt worden, wobei die meisten der 26 falsch erkannten Satzgrenzen auf Aufzählungen zurückzuführen sind. Darüber hinaus scheinen Sätze, die mit „etc.“ enden, zu Erkennungsfehlern zu führen. Weiterhin kann festgestellt werden, dass in 70,8 % der Sätze eine Softwarefunktionsbeschreibung vorgenommen wird.

Die Korpusstatistik bezogen auf die 274 korrekt erkannten Sätze findet sich in Tabelle 6.4. Durchschnittlich beträgt die Satzlänge 19 Token bzw. 120 Zeichen. Dabei ist die Existenz kurzer Sätze wie „*Easy to use*“, die nur drei *Token* bzw. 13 Zeichen umfassen, ebenso interessant wie die sehr umfangreicher Sätze, geben

Merkmal	Häufigkeit
Anzahl *Types* / *Token*	1.288 / 5.118
∅ *Token* / Satz	19
Token Min. / Max.	3 / 57
∅ Zeichen / Satz	108
Zeichen Min. / Max.	13 / 343

Tabelle 6.4: Zusammensetzung der Stichprobe

sie doch Auskunft über die Anwendbarkeit der in Abschnitt 5.5.2 und Anhang C.1 diskutierten *Preprocessing*-Verfahren, beispielsweise der Sprachenidentifizierung.

Tabelle 6.5 stellt Eigenschaften der Stichprobe dar, die für den Anwendungsfall der prädikatbasierten Kompensation von Unvollständigkeit von Bedeutung sind. So beinhaltet die Stichprobe insgesamt 547 Prädikate und durchschnittlich zwei Prädikate pro Satz.

Merkmal	Ausprägung
Prädikate	
Anzahl Prädikate	547
∅ Prädikate / Satz	2
Named Entities	
Anzahl NE	324
Sonstiges	
Anzahl Rechtschreibfehler (Algorithmus)	166
Anzahl Rechtschreibfehler (Gegenprobe, Mensch)	33
Abkürzungen (z. B. IMAP)	132

Tabelle 6.5: Merkmale und ihre Ausprägungen in der Stichprobe

Die weitere Analyse der Sätze zeigt, dass der Kontext der Prädikate, welcher genutzt wird, um fehlende Instantiierung zu kompensieren, eine Vielzahl an NE (324) und Abkürzungen (132) aufweist. Dies ist wichtig, da kompensierte Argumente, die NE und Abkürzungen enthalten, für Endanwender problematisch sein können, da nicht davon ausgegangen werden kann, dass sie diese kennen.

Darüber hinaus wirken sich NE und Abkürzungen auch auf die automatische Rechtschreibkorrektur aus. So erzielt diese insgesamt Ergebnisse, die eine alarmierend hohe Anzahl an Fehlerkennungen und -korrekturen beinhalten. So wurde zum Beispiel „*Winmail*“ fälschlicherweise zum Vornamen „*Ismail*“ korrigiert. Als kritischer sind Korrekturen wie bei dem Satz „*It can Convert .eml to PDF, and .eml to Image*“ zu bewerten, der zu „*It can Convert .XML to PDF, and .XML to Image*“ verändert wurde. Das Dateiformat EML wurde zu XML verändert und somit die Aussage verfälscht. Angesichts dieser Ergebnisse sollte auf eine automatische Rechtschreibkorrektur verzichtet werden. Gegen die Anwendung einer automatischen Rechtschreibkorrektur spricht auch die geringe Anzahl an Rechtschreibfehlern, wie sie die manuelle Rechtschreibkorrektur aufgezeigt hat (33 Rechtschreibfehler).

6.2.3 Umfang des PAS-Korpus

Um einen Eindruck des PAS-Korpus zu erhalten, ist in Tabelle 6.6 eine weitere Korpusstatistik dargestellt, die über die der Stichprobe hinausgeht. Wie bereits angeführt, besteht das gesamte Korpus aus 193.641 Softwarebeschreibungen, die in insgesamt 23 Softwarekategorien und weiter in 253 Unterkategorien unterteilt werden.

Merkmal	**Ausprägung**
Kategorien	23
Unterkategorien	253
Beschreibungen	193.641
∅ *Token* / Beschreibung	123
∅ Zeichen / Beschreibung	738
∅ Sätze / Beschreibung	7
∅ Prädikate / Beschreibung	15
∅ Prädikate / Satz	2

Tabelle 6.6: Merkmale und ihre Ausprägungen im PAS-Korpus

Werden die Softwarebeschreibungen hinsichtlich der Länge betrachtet, so sind sie im Durchschnitt 738 Zeichen bzw. 123 *Token* lang und bestehen aus sieben Sätzen. Dabei schwankt die Länge je nach Softwarekategorie merklich. So sind die Beschreibungen von Software zum Thema Reisen am längsten (∅ 1.055 Zeichen), während sie im Bereich der Bildschirmschoner und Schreibtischhintergründe am kürzesten sind (∅ 449 Zeichen). Auch ist die Verteilung der Softwarebeschreibungen innerhalb der Kategorien nicht gleich. Am meisten Beschreibungen finden sich in den Softwarekategorien Spiele (25.799), Treiber (22.442) und Betriebssystem *Utilities* (16.622). Die aufgrund des E-Mail-Anwendungsfalls relevante Kategorie Kommunikation, zu der auch E-Mail-Kommunikation zählt, umfasst 5.882 Softwarebeschreibungen.

Da es sich um ein PAS-Korpus handelt, ist neben der Verteilung der Softwarebeschreibungen innerhalb der Softwarekategorien auch das Vorkommen von Prädikaten von Interesse. Hier sind im Durchschnitt 15 Prädikate pro Softwarebeschreibung festzustellen, was durchschnittlich zwei Prädikate pro Satz bedeutet – dieser Wert ist somit deckungsgleich zu dem Wert aus der untersuchten Stichprobe.

6.3 Weitere Ressourcen

Neben dem Anforderungsbeschreibungskorpus und dem PAS-Korpus sind Ressourcen notwendig, die als Nebenentwicklungen innerhalb dieser Arbeit zu betrachten sind. Sie stehen somit nicht im Vordergrund dieses Kapitels und werden dennoch der Vollständigkeit halber im Folgenden gebündelt vorgestellt. Es handelt sich dabei sowohl um komplette Eigenentwicklungen, als auch um modifizierte Standardressourcen sowie um Ressourcen, deren Umfang stark beschränkt ist und die ausschließlich der Funktionsdemonstration dienen.

Eine sehr einfache Form von Ressourcen sind Listen, die an verschiedenen Stellen dieser Arbeit zum Einsatz kommen, unter anderem als ***White-* und *Blacklists***. Sie

dienen primär der Performanzsteigerung des Gesamtsystems, indem bestimmte *Token* von der Verarbeitung ausgenommen werden (sei es, weil das Verarbeitungsergebnis vorweggenommen werden kann oder weil ein *Token* als irrelevant eingestuft wird). So enthält beispielsweise die *Blacklist* der Unvollständigkeitskompensation *Token* wie „*want*“, „*like*“ und „*be*“ (s. Abschnitt 5.5.5). Darüber hinaus wird an mehreren Stellen auch die *Blacklist* der *Apache Foundation* eingebunden (z. B. beim Indikator lexikalischer Ambiguität), auf der wenig bedeutungstragende *Token* wie „*thus*“, „*to*“ und „*too*“ vermerkt sind. Neben dieser Listenart existiert eine **Liste vager Adjektive**, die zur Erkennung von Vagheit in Abschnitt 5.5.6 herangezogen wird. Sie umfasst derzeit 1.465 Adjektive (z. B. „*fast*“, „*warm*“). Über eine reine Auflistung von *Token* hinaus gehen Listen, die beispielsweise Synonyme enthalten und daher nicht nur ein *Token* in einem Eintrag führen, sondern mehrere in Form von Aufzählungen. So enthält die **Synonymliste** für den Indikator der referentiellen Ambiguität zum Beispiel den Eintrag: „*application, program, software, system*“ (s. Abschnitt 5.3.2.3). Nichtsdestotrotz ist es nach wie vor ein einfacher Listentyp[111].

Demgegenüber stehen strukturierte Ressourcen, die unterschiedliche Informationen in einen hierarchischen Zusammenhang bringen. An mehreren Stellen in dieser Arbeit (z. B. bei der strukturierten Ergebnisausgabe) kommt hierzu die erweiterbare Auszeichnungssprache XML zum Einsatz. So wird unter anderem bei der Kompensation von Unvollständigkeit (s. Abschnitt 5.5.5) auf eine modifizierte Version der **Propbank** (Palmer et al., 2005) zurückgegriffen, die auf die, für die Kompensation unvollständiger Prädikate notwendigen, Angaben reduziert wurde (vgl. Beispiel 6.3.1).

Beispiel 6.3.1 (Eintrag aus der modifizierten Propbank)

```
<roleset id=„delete.01“ reqroles=„2“ roles=„3“>
  <role f=„PAG“ descr=„entity removing“ req=„1“ n=„0“/>
  <role f=„PPT“ descr=„thing being removed“ req=„1“ n=„1“/>
  <role f=„DIR“ descr=„removed from“ req=„0“ n=„2“/>
</roleset>
```

Die Ressource umfasst derzeit 8.128 Prädikate in mehreren Lesarten (z. B. delete.01) und 21.153 definierte Leerstellen. Eine Besonderheit stellt das Attribut „*req*“ dar, welches neu aufgenommen wird und die notwendige Angabe eines Prädikats im Kontext von Anforderungsbeschreibungen markiert (s. Abschnitt 5.5.5).

Eine weitere Ressource ist der WSD-*Cache*. Dieser wird ihm Rahmen der lexikalischen Disambiguierung zur Minimierung der Laufzeit eingesetzt. Es handelt sich um eine Zwischenspeicherung (in einer MySQL-Datenbank) von Merkmalen einzelner *Token*, die über Babelfy disambiguiert und mittels BabelNet um Zusatzinformationen (z. B. Kategorie, Domäne, Lemma) angereichert wurden (vgl. Tabelle 6.7). Derzeit umfasst der *Cache* 6.827 Einträge, wobei alle Einträge nur eine Haltbarkeit von 14 Tagen haben, bevor sie erneut abgerufen werden (Zeitstempel). Es handelt sich dabei um eine flexible Ressource, die stetiger Veränderung unterliegt.

[111] Die Synonymliste enthält derzeit 803 Wortgruppen mit insgesamt 10.274 Wörtern.

BabelID	Lemma	Kategorie	Domäne	Beschreibung	Zeitstempel
bn:03164709n	*Keyboard shortcut*	GUI_techniques	Computing=0.45	*In computing, a keyboard shortcut is a series of one or several keys that invoke a software or operating system operation when triggered by the user.*	2017-03-18
bn:01664953n	*Selection* (GUI)	GUI_techniques	Computing=0.36	*In computing and user interface engineering, a selection is a list of items on which user operations will take place.*	2017-03-18
bn:03173309n	*Point and click*	GUI_techniques	Video_games=0.63	*Point and click are the actions of a computer user moving a pointer to a certain location on a screen and then pressing a button on a mouse, usually the left button, or other pointing device.*	2017-03-05

Tabelle 6.7: Auszug aus dem WSD-*Cache*

7 Implementierung

Im Folgenden wird das in Kapitel 5 beschriebene Konzept als Prototyp programmiertechnisch realisiert. Hierfür wird in Abschnitt 7.1 die Systemarchitektur erläutert und anschließend das Testsystem vorgestellt (s. Abschnitt 7.2). Die Umsetzung der Informationsverarbeitung wird in Abschnitt 7.3 beschrieben.

7.1 Systemarchitektur

Die Begriffe System- und Softwarearchitektur sind voneinander abzugrenzen. Systemarchitektur schließt „[...] viele Computer, Speichersysteme sowie Netzwerk-Komponenten ein, die durch ihr Zusammenwirken eine Reihe von verfügbaren, sicheren und skalierbaren Diensten [...]“ ermöglichen (Dustdar et al., 2003, S. 10), während Softwarearchitektur „eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Beziehungen“ (Balzert, 2003, S. 4) umfasst. Bei der Systemarchitektur ist die „[...] Betrachtungsweise auf Systeme als Bauteile gerichtet und nicht auf die Software-Bauteile für [...] Systeme. Komponenten einer System-Architektur sind daher anders zu diskutieren als Komponenten und Beziehungen einer Software-Architektur“ (Dustdar et al., 2003, S. 5).

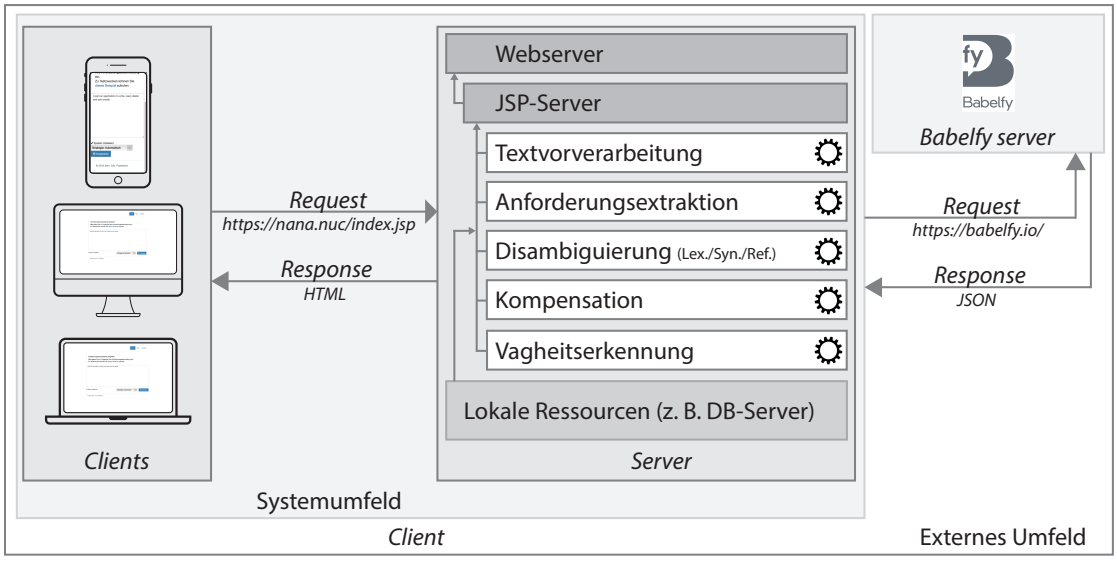


Abbildung 7.1: Überblick über das Softwaresystem

Einen Überblick über das Gesamtsystem und dessen Umfeld gibt Abbildung 7.1. Dargestellt sind neben beispielhaft gewählten *Clients* im Systemumfeld auch der

Server[112] sowie das externe Umfeld. Die Begriffe „Systemumfeld“ und „Externes Umfeld“ beschreiben Formen und Grenzen der Interaktion (*Request – Response*). Während das Systemumfeld die Interaktion zwischen *Clients* und *Servern* innerhalb des Softwaresystems beschreibt, umfasst das externe Umfeld Drittapplikationen (z. B. Babelfy), die vom *Server* angefragt werden. In diesem Fall agiert demnach das Softwaresystem als *Client*. Eine Interaktion zwischen Endanwender und externer Ressource ist nicht vorgesehen. Der in Abbildung 7.1 dargestellte *Server* enthält bereits die vorgesehenen Verarbeitungskomponenten. Auf Grund der Vielschichtigkeit des *Server*-Begriffs wird im Folgenden für Kompensationskomponenten, die auf dem zentralen *Server* oder weiteren Computern bereitgestellt werden und als *Server*-Applikation fungieren, der Begriff eines Dienstes herangezogen. Den serverseitigen Aspekt greift Abbildung 7.2 unter Auslassung der *Client*-Perspektive auf. Abgebildet ist der zentrale *Server*, der Dienste bereitstellt, die zur Kompensation benötigt werden. Diese Dienste greifen auf verschiedene Ressourcen zurück (z. B. Textdateien, Datenbanken), die unterschiedliche Anforderungen an das System stellen können[113].

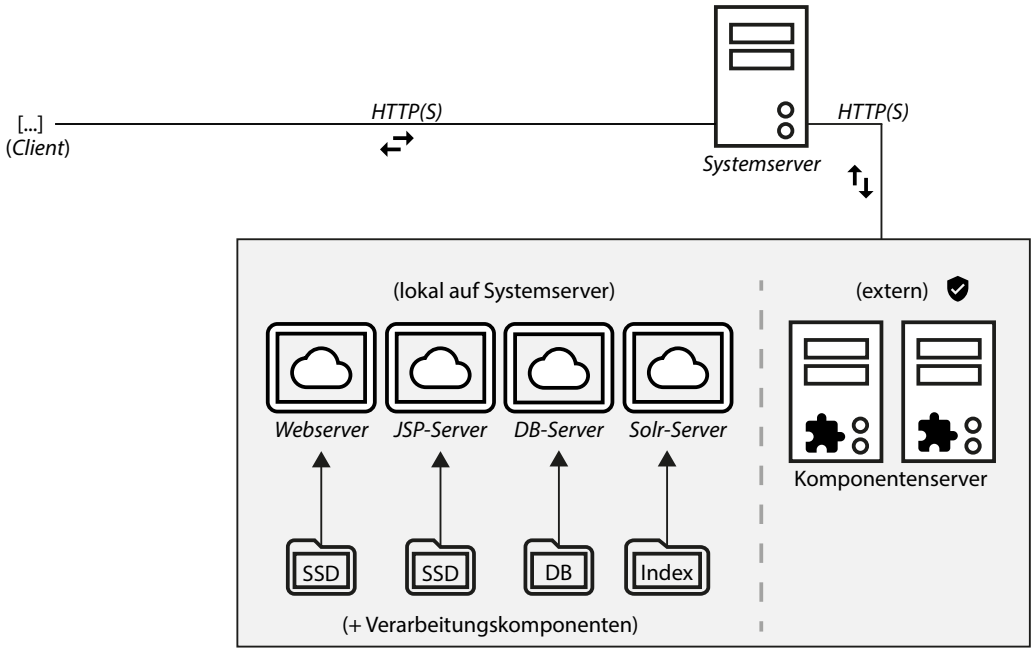


Abbildung 7.2: Serverseitige Systemperspektive

Die Kompensationsdienste kommunizieren über HTTP-Schnittstellen bzw. werden über diese auch von der zentralen Verarbeitungskomponente angesprochen (s. Abschnitt 7.3). Dies bedeutet, dass sich alle eingebundenen Computer, wie es die *Client-Server*-Architektur auch nahelegt, in einem (gemeinsamen) Netzwerk befinden müssen bzw. über einen Zugang zum Internet verfügen. Abbildung 7.2 zeigt somit auch den Aspekt der Skalierbarkeit auf (s. Abschnitt 7.4.2.3). Einzelne Dienste können aufgrund der angesprochenen netzwerkbasierten Kommunikation (⇅) ohne Weiteres

[112] Der Begriff „*Server*“ ist ambig, da dieser sowohl im Sinne der Hardware als auch im Sinne der Software (Software, die auf einem *Server* ausgeführt wird) genutzt werden kann.

[113] Beispielsweise erfordern Modelle, die im Arbeitsspeicher gehalten werden eine entsprechend ausgereifte RAM-Ausstattung, während umfangreiche Lexika eine schnelle Festplatte voraussetzen.

auf zusätzliche Computer (Komponentenserver, ✱) ausgelagert werden (horizontale Skalierung). Die Kommunikation sollte dabei im Sinne der Sicherheit über HTTPS verschlüsselt (🛡) erfolgen (s. Abschnitt 7.4.1.2). In dieser prototypischen Implementierung sind alle Dienste auf einem *Server* verfügbar (s. Abschnitt 7.2). Darüber hinaus greifen Endanwender in diesem Szenario über das Internet auf den *Server* zu.

7.2 Testumgebung

Als Testumgebung wird in dieser Arbeit serverseitig ein Intel NUC[114] (NUC6i3SYH) herangezogen, der die in Tabelle 7.1 dargestellten, relevanten Merkmale aufweist. Die Wahl der Testumgebung ist sowohl beim *Server* als auch beim *Client* der Verfügbarkeit bereits bestehender Hardware geschuldet. Grundsätzlich können alle (mindestens) äquivalenten Hardwarekonstellationen genutzt werden.

Merkmal	**Ausprägung**
Prozessor	Intel Core i3-6100U, 2x 2.30 GHz, 3 MB Cache
RAM	Crucial SO-DIMM Kit 32GB, DDR4-2133
Festplatte	Samsung SSD 850 Pro 512GB, 6 GB/s
Konnektivität	Verbunden über LAN, Gigabit
Betriebssystem	Linux, Debian Stretch
Java	openjdk (1.8.0)

Tabelle 7.1: Testumgebung (*Server*)

Clientseitig wird, im Sinne der Interoperabilität (s. Abschnitt 7.4.2.1), auf verschiedene Systeme und Konfigurationen zurückgegriffen. Primär getestet wird mit einem Apple MacBook Pro (Retina, 13") und einem iPhone 5S (vgl. Tabelle 7.2). Die Geräte eignen sich insbesondere, da sie aufgrund der weiten Verbreitung und eingeschränkten Konfigurierbarkeit bei Defekten mit wenig Aufwand formgleich zu ersetzen sind.

Merkmal	**Ausprägung**
Modell	**MacBook Pro (Retina, 13", Ende 2016)**
Konnektivität	Verbunden über LAN, Gigabit
Auflösung	2560 x 1600
Webbrowser	Safari 10.0; Mozilla Firefox 49.0.1
Betriebssystem	MacOS Sierra 10.12
Modell	**iPhone 5S**
Konnektivität	Verbunden über WLAN
Auflösung	1136 x 640
Webbrowser	Safari 602.1; Google Chrome 54.0
Betriebssystem	iOS 10.2

Tabelle 7.2: Testumgebungen (*Clients*)

[114] Siehe: `http://intel.com/content/www/us/en/nuc/nuc-kit-nuc6i3syh.html` (Stand: 12.01.17).

7.3 Programmiertechnische Umsetzung

Im Folgenden wird zuerst auf das Strukturierungsprinzip des zu entwickelnden Softwaresystems eingegangen, welches die Struktur dieses Abschnitts bestimmt: „Der grundlegende Ansatz zur Strukturierung von Softwaresystemen ist eine Zerlegung in Schichten“ (Dunkel und Holitschke, 2003, S. 16). Weitläufig etabliert hat sich dabei die Unterteilung in drei Softwareschichten, wie sie Abbildung 7.3 zeigt: Präsentationsschicht, Anwendungsschicht und Datenschicht (auch: Persistenzschicht).

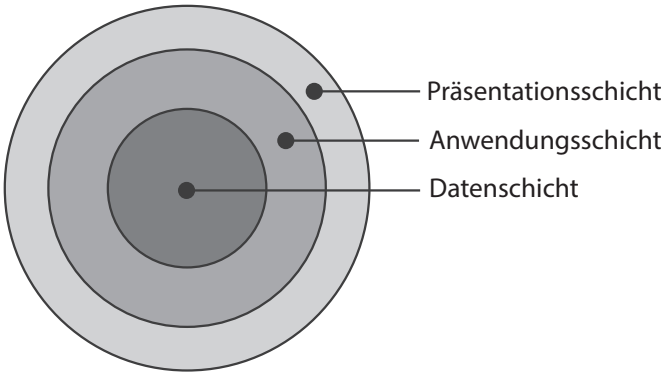


Abbildung 7.3: Drei-Schichten-Architektur als Strukturierungsprinzip von Software

Nach Dunkel und Holitschke (2003, S. 17) beinhalten diese Schichten die wesentlichen Aufgaben einer Software[115]. So enthält die Präsentationsschicht zum Beispiel die Datendarstellung und die Benutzerinteraktion. Die Anwendungsschicht beinhaltet die fachlichen Objekte sowie die fachliche Logik, während die Datenschicht die dauerhafte Datenverwaltung ermöglicht (Dunkel und Holitschke, 2003, S. 17). Eine Schicht kann dabei immer nur auf innere Schichten zurückgreifen. Dies „[führt] somit zu kohärenten und schwach gekoppelten Strukturen und [bietet] die Basis für eine physikalische Verteilung auf verschiedene Rechner“ (Dunkel und Holitschke, 2003, S. 16).

Die Ausprägung der Drei-Schichten-Architektur verteilter Systeme wird unter Betrachtung der *Client-Server*-Struktur deutlich: Je nachdem, wie die Schichten auf die *Clients* und die *Server* verteilt sind, demnach, wie viele Aufgaben auf der einen und auf der anderen Seite vorgesehen sind, wird von schwer- oder leichtgewichtigen *Clients* gesprochen (Dunkel und Holitschke, 2003, S. 22). Dieser Aspekt wird im folgenden Abschnitt 7.3.1 weiter vertieft. Darüber hinaus richtet sich auch der darauffolgende Aufbau dieses Abschnitts an dem dargestellten Prinzip der Drei-Schichten-Architektur aus. So enthält Abschnitt 7.3.1 auch Überlegungen und Angaben zur Implementierung der Präsentationsschicht. Es folgt Abschnitt 7.3.2 mit den fachlichen Objekten sowie der fachlichen Logik. Die Datenschicht wird in Abschnitt 7.3.3 diskutiert. Aufgabe wird es darüber hinaus auch sein, die Umsetzung der in Abschnitt 7.4 benannten Qualitätsmerkmale von Softwaresystemen anhand der genannten Schichten aufzuzeigen und zu diskutieren.

[115] Detaillierte Inhalte zu logischen Softwareschichten geben Dunkel und Holitschke (2003, S. 16 ff.).

7.3.1 Präsentationsschicht

Die Präsentationsschicht bezeichnet das klassische GUI, welches zum einen die Darstellung von Daten ermöglicht und zum anderen Endanwender befähigt, mit dem Softwaresystem zu interagieren. Konzeptionell besprochen wird die Benutzerschnittstelle in Abschnitt 5.5.1. In Abschnitt 7.3.1.1 folgen Überlegungen zur Verteilung der einzelnen Schichten auf die *Clients* bzw. den *Server* sowie Angaben zur programmiertechnischen Umsetzung in Abschnitt 7.3.1.2.

7.3.1.1 Fat vs. Thin Clients

Hinsichtlich der *Clients* ist zwischen schwergewichtigen (engl. *fat*) und leichtgewichtigen (engl. *thin*) *Clients* zu unterscheiden[116]. Während *Fat Clients* alle genannten Schichten beinhalten und nur zur Datenverwaltung auf *Server* zugreifen, sind *Thin Clients* so konstruiert, dass sie die Präsentationsschicht darstellen und sich die weiteren Schichten auf einem oder mehreren *Server(n)* befinden (vgl. Abbildung 7.4).

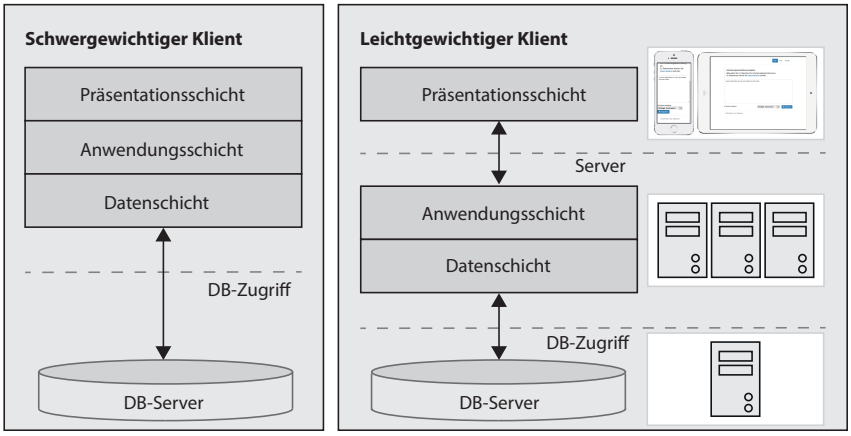


Abbildung 7.4: Unterschiedliche Schichtenaufteilung von *Fat* und *Thin Clients*. In Anlehnung an Dunkel und Holitschke (2003, S. 22)

In dieser Arbeit ergeben sich bereits aus dem Konzept in Abschnitt 5.5.1 folgende Anforderungen an die *Clients*: Zum einen muss das Softwaresystem niedrige Nutzungsbarrieren aufweisen, geräteübergreifend sowie plattformunabhängig arbeiten und intuitiv zu bedienen sein, was beispielsweise komplexere Installationsroutinen, wie sie hier erforderlich sind, ausschließt und eine niedrige Navigationstiefe erfordert. Zum anderen soll es grundsätzlich möglich sein, auch alternative *Clients* zu betreiben.

Diese angeführten Anforderungen schließen die Nutzung von *Fat Clients* bereits aus, da Installations- und Konfigurationsbemühungen notwendig wären, ein geräteübergreifendes und plattformunabhängiges System nur schwer umzusetzen ist (z. B. aufgrund von Systemanforderungen der Verarbeitungskomponenten) sowie die Entwicklung von alternativen *Clients* nennenswerten Aufwand bedingen würde. Zwar gelten *Fat Clients* als leichter zu implementieren, allerdings kann eine solche Architektur den Anforderungen dieser Arbeit nicht gerecht werden, sodass *Thin Clients* zu bevorzugen sind. *Thin Clients*, die als reine *Browser*-Anwendung konzipiert

[116] Vor- sowie Nachteile von *Thin/Fat Clients* listen Dunkel und Holitschke (2003, S. 22 ff.) auf.

werden, basieren auf etablierten Webtechnologien (z. B. HTML5, XML, JavaScript). Generell gelten diese *Clients* als komplex in der Umsetzung und können langsamer in der Ausführung sein, da Kommunikation über Netzwerke erforderlich ist (Dunkel und Holitschke, 2003, S. 23 ff.).

7.3.1.2 Thin Client: Benutzerinterkation und Umsetzung

Der *Thin Client* ermöglicht Benutzerinteraktion durch Ein- und Ausgabe von Daten. In den Abschnitten 5.5.1 und 5.5.7 wurde hierfür Fließtext als Eingabe (*Input*) und das Ergebnis, weitere Erläuterungen zur Verarbeitung und Kompensation sowie ein Verarbeitungs- und Kompensationsprotokoll als Ausgaben (*Output*) gewählt. Diese vier Ansichten sind in Abbildung 7.5, ausgehend von der Startseite, dargestellt.

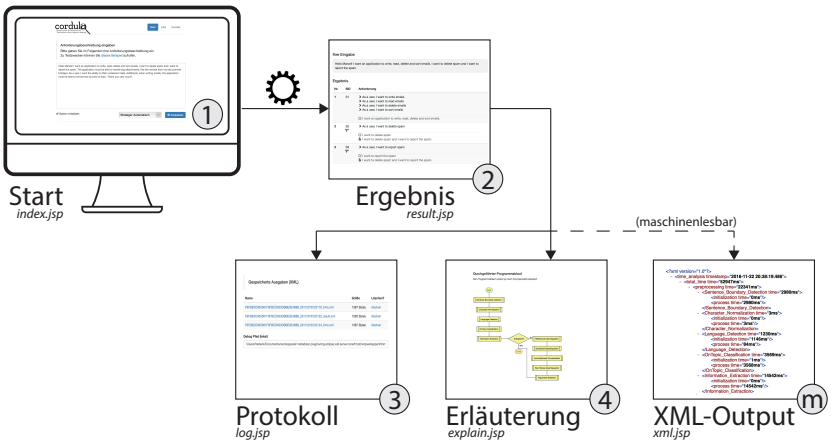


Abbildung 7.5: Flache Systemnavigation als Grundlage niedriger Einstiegsbarrieren

Neben den, für Endanwender verständlichen, Ansichten (1-4) existiert weiterhin der XML-*Output*, der eine maschinelle Weiterverarbeitung ermöglicht (*m*). Damit Endanwender die Ergebnisse strukturiert abspeichern können, ist auch diese Ansicht über die Ergebnisansicht zu erreichen. Um die Benutzerinteraktion für die vier einzelnen Ansichten möglichst intuitiv zu gestalten, wird auf die Empfehlungen von Nielsen und Loranger (2006, S. 169 ff.) zurückgegriffen. Deshalb werden nur wesentliche Bedienelemente angezeigt, einheitlich bezeichnet und die Navigationstiefe auf eine Ebene beschränkt (vgl. Abbildung 7.5). Um die Komplexität der Bedienung weiter zu reduzieren, folgt auf Eingabe des *Inputs* direkt die Ergebnisanzeige, woraufhin Endanwender sich bei Bedarf weiterführende Informationen anzeigen lassen können. Aus der Sicht der Endanwender verhält sich die resultierende Webseite wie eine statische Webseite, da sich die Komplexität im Verborgenen auf Seite des *Servers* abspielt. Die programmiertechnische Umsetzung basiert auf *JavaServer Pages* (JSP)[117]. Hierbei handelt es sich um eine Technologie zur Entwicklung von Webseiten, die auf Webstandards wie HTML basiert und Möglichkeiten zur dynamischen Inhaltsgestaltung bietet. Außerdem werden bekannte *Markup*-Elemente durch spezielle JSP-Elemente ergänzt. Statt HTML-Elemente in den Programmquelltext zu übernehmen, werden spezielle *active elements* in den HTML-Quelltext übernommen (Bergsten, 2004, S. 5).

[117] Einen Einstieg in JSP geben Balzert (2003) sowie Bergsten (2004).

Diese JSP-Elemente werden dabei vom *Server* ausgeführt und die Ergebnisse in die restliche, statische Webseite eingefügt (Bergsten, 2004, S. 4).

7.3.2 Anwendungsschicht

Die Anwendungsschicht stellt den funktionalen Kern des Softwaresystems dar. In ihr „[...] werden sämtliche fachlichen Funktionalitäten der Anwendung realisiert. Dazu gehören datentragende Geschäftsobjekte, aber auch die Realisierung fachlicher Geschäftsprozesse“ (Dunkel und Holitschke, 2003, S. 18). Im Folgenden werden sowohl die genannten datentragenden Klassen besprochen (s. Abschnitt 7.3.2.1) als auch die Funktionalitäten und deren Umsetzungen aufgezeigt (s. Abschnitt 7.3.2.2). Auch finden sich in den Abschnitten 7.3.2.3 (Indikatoren) und 7.3.2.4 (Strategien) Angaben zur programmiertechnischen Umsetzung.

7.3.2.1 Datentragende Klassen

Im Zentrum der Softwareapplikation steht die vom Endanwender eingegebene (reale) Anforderungsbeschreibung, die über mehrere Verarbeitungsschritte hinweg erweitert und transformiert wird. Um dies zu erreichen, existieren elementare Datenklassen, die sowohl die ursprüngliche Anforderungsbeschreibung, einzelne Beschreibungselemente sowie mögliche Zusatzinformationen abbilden. Ausgehend von der Anforderungsbeschreibung (auch: *Description*), werden einzelne Sätze (auch: *Sentence*) erzeugt, die wiederum aus einzelnen *Token* bestehen und im Ergebnis als strukturierte Sätze (auch: *Controlled sentence*) ausgegeben werden können. Zwar existieren noch weitere Klassen (z. B. *Token Groups*, *Chains*), im Folgenden werden aber nur die für das Gesamtverständnis relevanten Klassen dargestellt (vgl. Abbildung 7.6).

Die *Description*-Klasse ist die höchste Klasse in der Klassenhierarchie und stellt den Ausgangspunkt der Verarbeitung dar, da sie initial die unbearbeitete Anforderungsbeschreibung der Endanwender enthält und während der gesamten Weiterverarbeitung speichert. Ausgehend von dieser Anforderungsbeschreibung werden *Sentence*-Objekte erzeugt, die beispielsweise Angaben zur syntaktischen Struktur, Konjunktionen aber auch zur Sprache sowie Relevanz (*On-* und *Off-Topic*) enthält.

Jede *Sentence*-Klasse kann wiederum eine Vielzahl an *Controlled Sentence*-Klassen begründen, da sich die Anzahl der kontrollierten Sätze nach Anzahl der Prozesswörter in einem Ausgangssatz richtet. Zu jedem Prozesswort, welches wie alle Wörter als *Token* repräsentiert wird, werden weitere zugehörige semantische Kategorien (z. B. Rolle, Objekt, Komponente) gesammelt und unter Hinzunahme einer Perspektive und einer syntaktischen Vorgabe gespeichert (s. Abschnitt 5.5.7). Ein *Controlled Sentence* kann somit auch als Menge verbundener *Token* verstanden werden, die auf Grundlage einer definierten Syntax die funktionale Kernaussage eines zugrundeliegenden Satzobjektes wiedergeben. Es wird deutlich, dass die Klasse der *Token* grundlegenden Charakter hat. Sie ist die – hinsichtlich der abgebildeten sprachlichen Einheiten – kleinste datentragende Klasse und enthält dennoch einen Großteil der Informationen, so zum Beispiel die semantischen Kategorien und Lesarten einzelner Wörter.

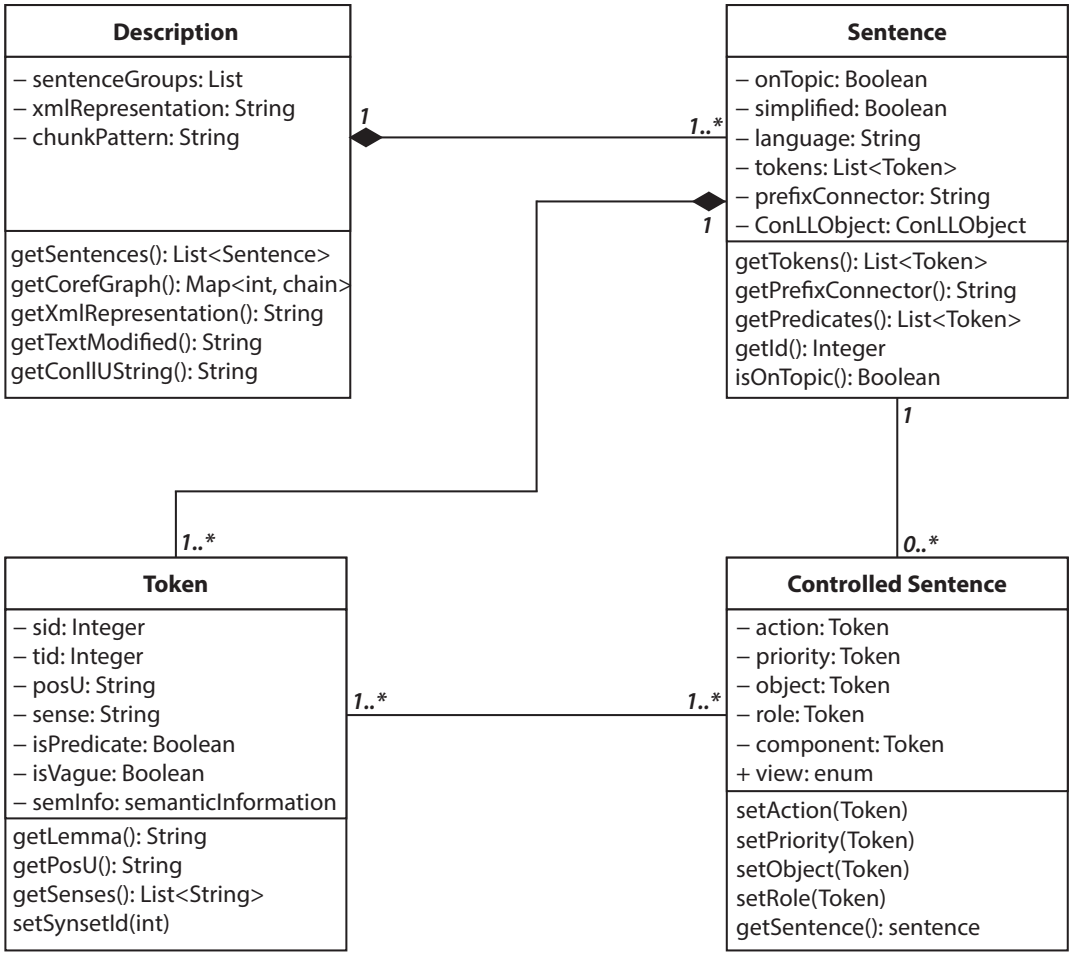


Abbildung 7.6: Datentragende Klassen (kompakte Darstellung)

7.3.2.2 Implementierung der Kompensationskomponenten

Im Folgenden wird die programmiertechnische Einbindung der einzelnen Kompensationskomponenten dargestellt (s. Abschnitt 5.5).

Lexikalische Disambiguierung

Wie bereits in den Abschnitten 2.1.1 und 5.5.4.1 diskutiert, wird lexikalische Ambiguität auf Basis einzelner *Token* und unter Berücksichtigung des jeweiligen Kontextes aufgelöst. In dieser Arbeit wird hierfür Babelfy (s. Abschnitt 3.3.1.1) als Komponente zur Disambiguierung mit der zugrundeliegenden Datenbank BabelNet herangezogen (s. Abschnitt 3.3.1.1). Im Zentrum von Abbildung 7.7 steht die Steuerungskomponente[118], die relevante, englischsprachige Sätze iterativ auf Basis der *Token* durchläuft. Hierbei werden alle *Token*, die zuvor im Rahmen des *Preprocessings* als semantisch relevant für die funktionale Anforderung markiert wurden (s. Abschnitt 5.5.2), einer Disambiguierung unterzogen – es sei denn, es wird zuvor festgestellt, dass sich das *Token* auf der *White-* oder *Blacklist* befindet (s. Abschnitt 5.5.4.1).

[118] Hierbei handelt es sich nicht um den *Controller*, der im Rahmen der Strategieeinführung vorgestellt wurde (s. Abschnitt 5.2), sondern vielmehr um eine eigene Steuerung innerhalb jeder Komponente.

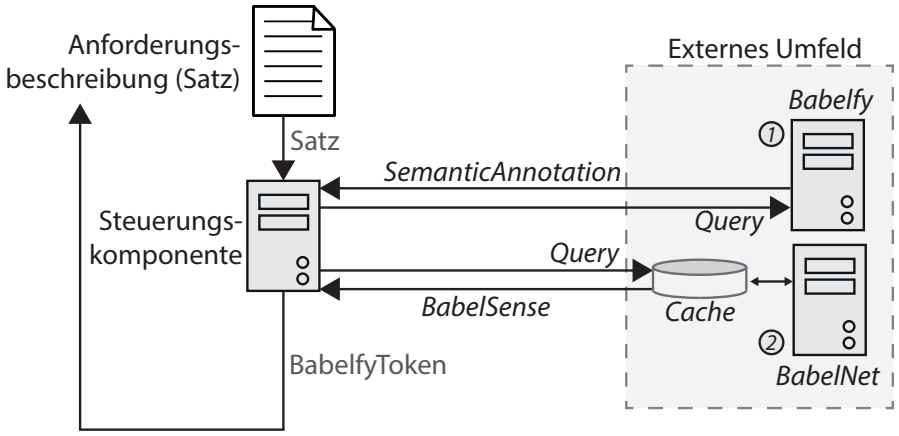


Abbildung 7.7: Integration von Babelfy als Disambiguierungskomponente

Um sowohl die *Token* als auch den Kontext an Babelfy zu übergeben, wird der zu untersuchende Satz zusammen mit Konfigurationsparametern[119] übermittelt, woraufhin Babelfy ein *SemanticAnnotation*-Objekt zurückgibt[120]. Dieses Objekt enthält neben einem *Disambiguation Score* auch Angaben zu genutzten Ressourcen sowie eine sogenannte BabelSynsetID pro *Token* (z. B. „bn:00005095n“), die genutzt werden kann, um weitere Informationen (z. B. Lemma, Lesarten, Bilder) zu jedem *Token* aus BabelNet abzufragen. Diese Informationen werden dem untersuchten *Token*-Objekt angehängt, wie es beispielhaft in Tabelle 7.3 dargestellt ist.

Merkmal	**Ausprägung**	**Quelle**
SID	2	*Preprocess.*
TID	3	
PosX	NN	
Lemma	*application*	
SemInfo	*component*	REaCT
isAmbig	*true*	Babelfy
Sense	*application*	
BabelID	bn:00005095n	
AmbigPictureURL	`.../OpenOffice.org_Writer.png`	BabelNet
AmbigCategory	*Application_software*	
AmbigDomain	*Computing*	
AmbigDescription	*A program that gives a computer instructions that provide the user with tools to accomplish a task*	

Tabelle 7.3: Durch Babelfy erweitertes *Token*-Objekt zu „*application*“

Tabelle 7.3 zeigt die von Babelfy und BabelNet breitgestellten Informationen, die für jedes disambiguierte *Token* vorliegen. Für die maschinelle Weiterverarbeitung ist insbesondere die disambiguierte Lesart und die BabelSynsetID von Bedeutung. Für

[119] Limitation genutzter Ressourcen (z. B. WordNet, Wikipedia); Disambiguierungssgrenzwerte etc.
[120] Siehe weiterführend: `http://babelfy.org/guide` (Stand: 12.01.17).

Endanwender sind darüber hinaus Angaben wie Domäne, Kategorie und Beschreibung relevant, um mehr über das disambiguierte *Token* zu erfahren und gegebenenfalls Verarbeitungsfehler (z. B. falsche Disambiguierung) leichter zu erkennen.

Da es sich bei Babelfy und BabelNet um externe Dienste handelt, ist eine Beeinflussung der Ausführungszeit ohne Weiterentwicklungen nicht möglich. Eine hohe Auslastung im Netzwerk oder eine hohe Auslastung auf den *Servern* kann sich auf die Performanz der gesamten Komponente zur lexikalischen Disambiguierung negativ auswirken. Aus diesem Grund wird ein *Caching*-Verfahren implementiert, das akquirierte Disambiguierungsobjekte speichert, sodass für ein *Token*, für einen definierten Zeitraum[121], nur eine Anfrage an BabelNet zu stellen ist. Die Ergebnisse der Evaluation zeigen einen positiven Effekt (Minimierung der Laufzeit) des *Cachings* auf die Performanz der lexikalischen Disambiguierung (s. Abschnitt 8.3.4). Es ist jedoch festzuhalten, dass die Anfrage an Babelfy durch das *Caching* nicht beschleunigt werden kann, da die Disambiguierung Fall für Fall und unter gegebenem Kontext durchgeführt werden muss.

Syntaktische Disambiguierung

Die syntaktische Disambiguierung wurde bereits in Abschnitt 5.5.4.2 beschrieben und wird in dieser Arbeit durch den *Stanford Parser* durchgeführt. Dazu werden alle relevanten, englischsprachigen Sätze iterativ an diese Komponente übergeben. Als Resultat wird ein *Tree*-Objekt[122] zurückgegeben, welches die produzierten syntaktischen Informationen enthält und somit beispielsweise die Grundlage dafür bildet, über eine *GrammaticalStructureFactory* die *Parsing*-Ausgabe im CoNLL-Format zu erzeugen (beispielsweise relevant aus Kompatibilitätsgründen zwischen Komponenten). Jeder Satz, der die syntaktische Disambiguierung durchläuft, wird um ein *Tree*-Objekt erweitert, auf welches zu jedem Zeitpunkt zugegriffen werden kann.

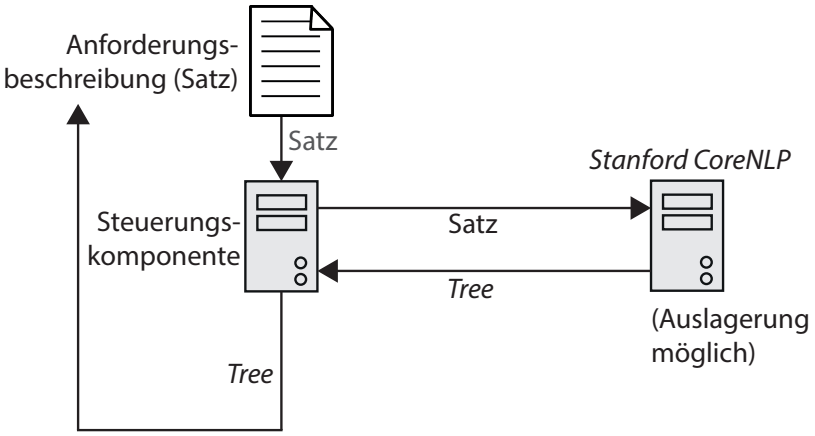


Abbildung 7.8: Integration von *Stanford CoreNLP* zur syntaktischen Disambiguierung

Der *Parser* als Bestandteil von *Stanford CoreNLP* wird als Serveranwendung ausgeführt und entsprechend in das Gesamtsystem eingebunden (vgl. Abbildung 7.8).

[121] Nach 14 Tagen gilt der *Cache* eines *Tokens* als obsolet und wird überschrieben.

[122] Siehe weiterführend: `http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/trees/Tree.html` (Stand: 12.01.17).

Auch hier ist es möglich (s. Abschnitt 7.4.2.3), diese Komponente auszulagern und auf einem Computer mit sehr viel Arbeitsspeicher auszuführen. Dies ist sinnvoll, da Erfahrungswerte zeigen, dass *Parsing* mit zunehmendem Satzumfang erhebliche Ressourcen in Anspruch nehmen kann[123]. Um die Auslagerung der *CoreNLP*-Komponente komfortabel zu ermöglichen, stehen alle erforderlichen Konfigurationsparamter (z. B. URL, *Port*) in einer Konfigurationsdatei zur Bearbeitung zur Verfügung. Eine Änderung am Quelltext ist somit nicht erforderlich.

Das Ergebnis der syntaktischen Disambiguierung zeigt Abbildung 7.9 in der Dependenz- bzw. Konstituentenansicht. Die graphische Darstellung wurde mit *conllu.js* erzeugt, einer frei verfügbaren JavaScript-Programmbibliothek zur Visualisierung des CoNLL-U Ausgabeformats[124].

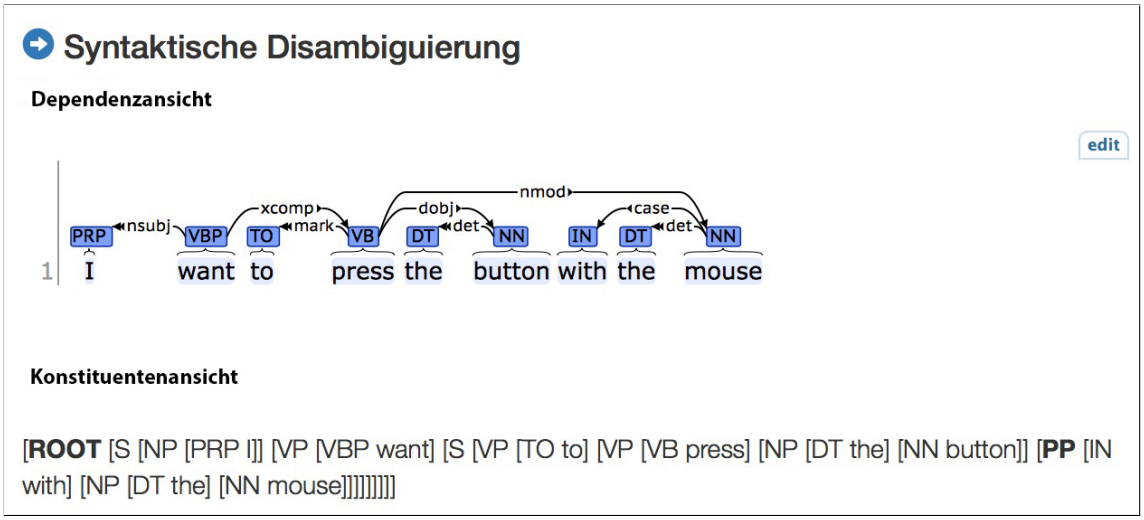


Abbildung 7.9: Dependenz- und Konstituentenansicht

Da allerdings die in Abbildung 7.9 dargestellte Konstituentenansicht für Endanwender, trotz der Hervorhebung relevanter Konstituenten, aufgrund der umfangreichen Klammerung nur schwer nachzuvollziehen ist, existiert zusätzlich eine Ansicht, die die Baumstruktur graphisch darstellt (vgl. Abbildung A.8 im Anhang).

Referentielle Disambiguierung

In Abschnitt 5.5.4.3 wurde die referentielle Disambiguierung konzeptuell dargestellt. Wie auch bei der syntaktischen Disambiguierung kommt hier *Stanford CoreNLP* zum Einsatz. Der Aufbau gestaltet sich daher analog, wenngleich kein Satz sondern die gesamte Anforderungsbeschreibung übergeben und kein *Tree*- sondern ein *CorefChain*-Objekt[125] zurückgegeben wird. Auf eine graphische Darstellung wird daher an dieser Stelle verzichtet und stattdessen auf Abbildung 7.8 verwiesen. Die Angaben zu Möglichkeiten der horizontalen Skalierung gelten analog.

[123] Entsprechende Erfahrungswerte finden sich in den *Stanford FAQs*. Siehe: `http://nlp.stanford.edu/software/parser-faq.shtml` (Stand: 12.01.17).

[124] Siehe weiterführend: `http://spyysalo.github.io/conllu.js/` (Stand: 12.01.17).

[125] Siehe weiterführend: `http://www-nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/hcoref/data/CorefChain.html` (Stand: 12.01.17).

Abbildung 7.10 stellt die mittels *Stanford CoreNLP* erkannten Koreferenzen dar. Beispielsweise ersichtlich ist die aufgelöste referentielle Ambiguität („*They*“) zwischen den Sätzen Nr. 1 und Nr. 2. Während das Personalpronomen sowohl auf „*emails*“ als auch auf „*file attachments*“ referenzieren könnte, wird sich seitens des Systems für das letzte Antezedens („*file attachments*“) entschieden. Auch zu erkennen ist die Koreferenz der Pronomen „*I*“ und „*me*“ sowie die Erkennung von „*spam*“ und „*it*“. In letzterem Fall wird korrekterweise nicht auf „*problem*“ als Antezedens referenziert.

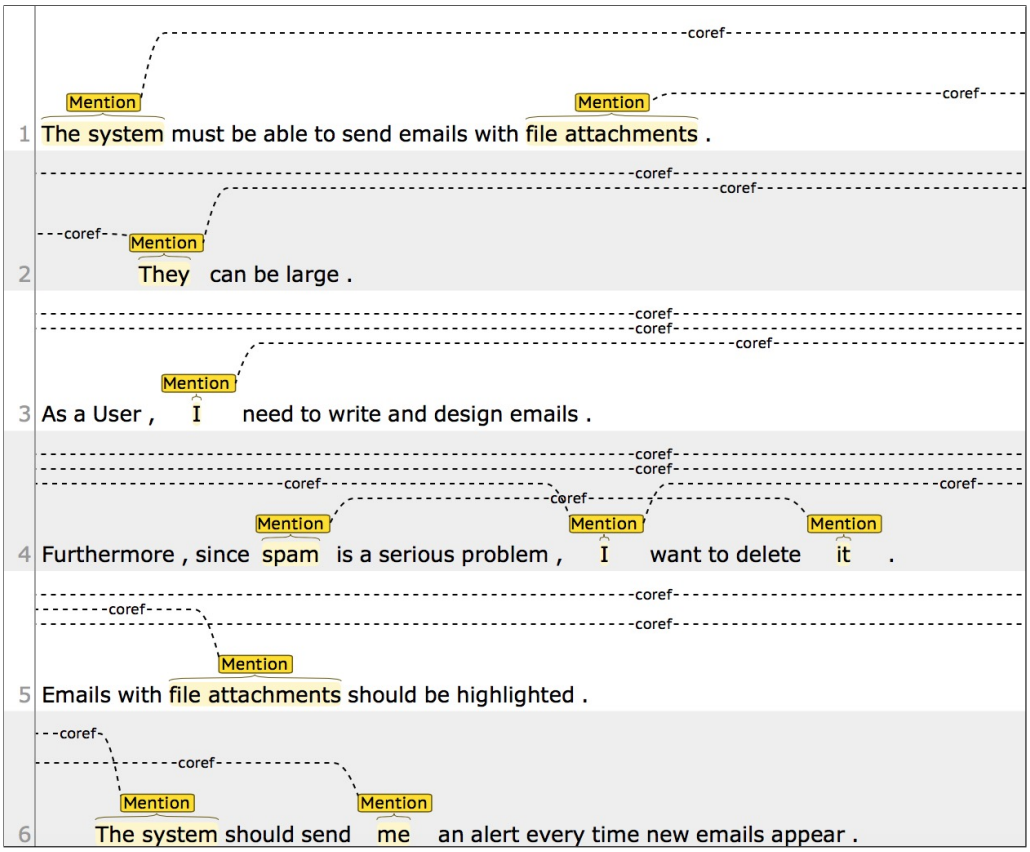


Abbildung 7.10: Mittels *Stanford CoreNLP* erkannte Koreferenzen

Wie in Abschnitt 5.5.4.3 und Abbildung 5.26 dargestellt, folgt nach der Anwendung von *CoreNLP* die erweiterte Ausdruckssuche, die beispielsweise auf die Ergebnisse der lexikalischen Disambiguierung zurückgreifen kann. In diesem Schritt können beispielsweise *Token*, die als weitere Kandidaten für eine Koreferenzkette in Frage kommen, über ihre semantischen Kategorien sowie Lesarten identifiziert werden. Dies ist besonders in Fällen hilfreich, in denen mit dem bisherigen Vorgehen aufgrund verschiedener Schreibweisen (z. B. *Microsoft Sharepoint, MS Sharepoint, Sharepoint*) fälschlicherweise keine Koreferenz festgestellt wurde. Hierfür wird auf die *Token Groups*-Objekte zurückgegriffen, die Angaben zu der Zusammengehörigkeit einzelner *Token* enthalten.

Kompensation von Unvollständigkeit

Das grundsätzliche Vorgehen bei der Kompensation von Unvollständigkeit wurde in den Abschnitten 2.3 und 5.5.5 beschrieben. Im Folgenden werden daran anknüpfend die wesentlichen softwaretechnischen Designentscheidungen erläutert.

Da es sich bei der Erkennung und Kompensation um ein mehrschichtiges Vorgehen handelt, bedürfen insgesamt drei beteiligte Kernkomponenten einer Erläuterung[126]: Im Zentrum steht auch hier die zentrale Komponentensteuerung (1), welche die Erkennung und Kompensation anstößt und resultierende Informationen zusammenführt. Diese Informationen wiederum werden von *MATE Tools*[127] (2) sowie *Apache Solr*[128] (3) bereitgestellt. Abbildung 7.11 zeigt den Informationsaustausch zwischen den Komponenten zur Erkennung und Kompensation von Unvollständigkeit. Eine Besonderheit in Abbildung 7.11 ist die Darstellung der getrennten Server (Hardware), was die hohe Skalierbarkeit der Kompensationskomponente unterstreicht. Denkbar ist allerdings auch, die Steuerungskomponente, *MATE Tools* und *Apache Solr* auf einem einzigen Server auszuführen.

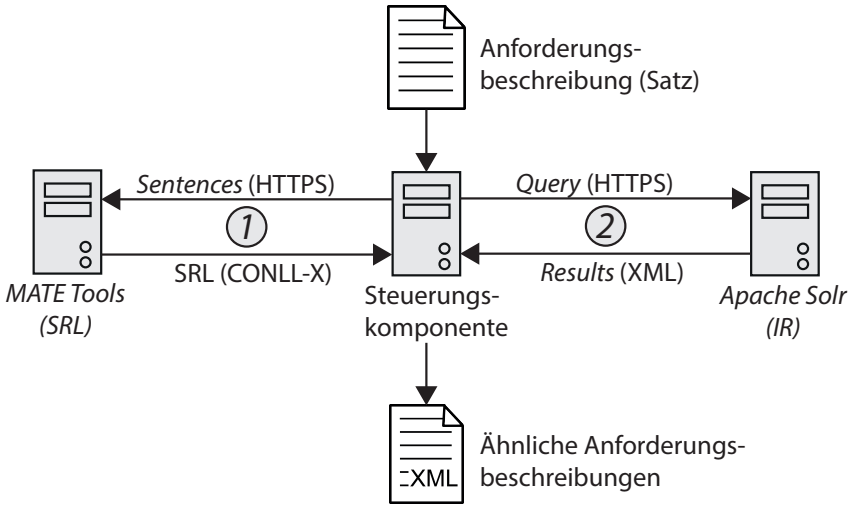


Abbildung 7.11: Komponenteninteraktion zur Kompensation von Unvollständigkeit

Bei den *MATE Tools* des Stuttgarter Instituts für Maschinelle Sprachverarbeitung handelt es sich um eine Sammlung statistischer NLP-*Tools*, wovon der SRL in dieser Arbeit verwendet wird, um einzelne Sätze gegebener Anforderungsbeschreibungen hinsichtlich Prädikaten und deren Valenz zu analysieren. Wird ein Prädikat wie beispielsweise „löschen“ (engl. *delete*) erkannt, wird geprüft, welche Argumente vorhanden sind und welche fehlen. Für fehlende Argumente wurde eine prototypische Wissensbasis auf Grundlage der etablierten *Proposition Bank* von Palmer et al. (2005) erstellt (vgl. Beispiel 7.3.1) und um die domänenspezifische Prädikatverwendung im Softwarekontext aus Abschnitt 6.2 ergänzt.

[126] Weiterführende Angaben zur Auswahl und Konfiguration der jeweiligen Komponenten finden sich in Bäumer und Geierhos (2016) sowie Geierhos und Bäumer (2016).

[127] Siehe: `https://code.google.com/archive/p/mate-tools/` (Stand: 12.01.17). Siehe weiterführend auch Björkelund et al. (2010).

[128] Siehe weiterführend: `http://lucene.apache.org/solr/` (Stand: 12.01.17).

Beispiel 7.3.1 (Modifizierte Prädikatdatenbank, Auszug)

```
<roleset id="delete.01" reqroles="2" roles="3">
   <role f="PAG" descr="entity removing" req="1" n="0"/>
   <role f="PPT" descr="thing being removed" req="1" n="1"/>
   <role f="DIR" descr="removed from" req="0" n="2"/>
</roleset>
```

Auf Grundlage der unvollständigen Prädikate wird eine Kompensationsanfrage (*Query*) erstellt, die an einen Suchmaschinenserver übermittelt wird, um ähnliche und vor allem zwingend vollständige Anforderungsbeschreibungen zurückzuerhalten. An dieser Stelle kommt *Apache Solr* als dritte Komponente – als Kompensations- bzw. Suchserver – zum Einsatz.

Bei *Apache Solr* handelt es sich um einen „*Standalone enterprise search server*“ (Apache Software Foundation, 2016), der auf *Apache Lucene*[129], einer etablierten Programmbibliothek zur Volltextsuche, basiert. Der Kompensationsserver enthält eine Menge von Anforderungsbeschreibungen, die mitsamt annotierten Prädikaten und Argumenten indiziert wurden. Eine Kompensationsanfrage an diesen *Server* liefert relevante Anforderungsbeschreibungen zurück, die ähnlich zu der unvollständigen Anforderungsbeschreibung sind, die es zu kompensieren gilt und mindestens das gesuchte Prädikat in der erkannten Lesart und mindestens das fehlende Argument enthält. Um ähnliche Anforderungsbeschreibungen zu finden, wird der Kontext berücksichtigt – der Satz vor und nach dem unvollständigen Satz. Eine Kompensationsanfrage an *Apache Solr* wird dabei nach folgendem Muster erstellt:

„*Sense*:[Lesart] *AND NOT* [fehlendes Argument]:*NULL AND Context*:[Kontext]“

Bestandteile des Musters sind neben dem fehlenden Argument die Kontextinformationen und auch das unvollständige Prädikat. Auffällig ist, dass bezüglich des Prädikats nicht nach dem spezifischen *Token* gesucht wird (z. B. „*deletes*“), sondern nach der erkannten Lesart (z. B. „delete.01“). Diesem Vorgehen liegt die Notwendigkeit zu Grunde, zwischen den verschiedenen Lesarten bei der Kompensationsanfrage zu unterscheiden, um Anforderungsbeschreibungen zurückzuerhalten, die sowohl im gegebenen Kontext relevant sind, als auch das Prädikat in der identischen Lesart beinhalten. Deutlicher wird dies anhand der Kompensationsanfrage zu folgendem Beispielsatz: „*Emails are the technology of the future. Because of that, I want to* ***send*** *emails and I want to* ***delete****. My friends are using it, too.*“ (vgl. Beispiel 7.3.2).

Beispiel 7.3.2 (Kompensationsanfrage für delete.01)
Sense:delete.01 AND NOT argument_01:NULL AND context:“Emails are the technology of the future. My friends are using it, too”

Um einen effizienten Vergleich des aktuell untersuchen Kontext mit den Kontexten im Suchmaschinenindex zu ermöglichen (s. Abschnitt 5.5.5), wird in *Apache Solr*

[129] Siehe weiterführend: `https://lucene.apache.org` (Stand: 12.01.17).

ein *Preprocessing* vorgenommen, deren Ablauf in Abbildung 7.12 dargestellt ist. Demnach werden nicht Sätze miteinander verglichen, sondern einzelne *Token* (*Bag of Words*), die mittels eines *WhiteSpaceTokenizers* erstellt werden. *Token*, die nicht auf der Stoppwortliste stehen, werden um Synonyme ergänzt, auf Kleinschreibung normalisiert und auf die Stammform reduziert.

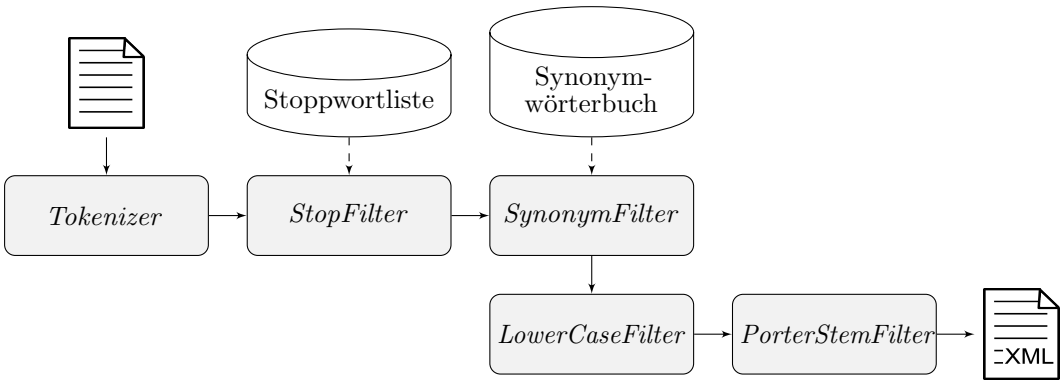


Abbildung 7.12: *Preprocessing* der Kontextinformationen in *Apache Solr*

Das Ergebnis der Kompensationsanfrage für das Prädikat *delete* in der Lesart delete.01 ist in Abbildung 7.13 dargestellt (Ausgabe an der Benutzerschnittstelle). Durch die SRL-Komponente (2) konnte hier bereits das Personalpronomen „*I*“ erkannt werden, während das ebenfalls obligatorische Argument A1 nicht erkannt werden konnte. Dieses Argument konnte allerdings erfolgreich kompensiert werden („*spam emails*”) und wird auf der Benutzeroberfläche durch ein Symbol (**+**) hervorgehoben.

Kompensation von Unvollständigkeit

Im Folgenden sehen Sie erkannte, zulässige Prädikate und deren Argumente.

SID/TID	Lesart (Ist/Soll/Kann)	Argumente
S1T4	send.01 (2/3/3)	A1: "emails" A0: "I" + A2: "[friends]"
S2T4	delete.01 (1/2/3)	A0: "I" + A1: "[spam emails]"

Abbildung 7.13: Beispielhafte Ausgabe der Kompensationskomponente

7.3.2.3 Umsetzung der Indikatoren

Wie in Abschnitt 5.3 dargestellt, basieren die Indikatoren zwar überwiegend auf gemeinsamen semantischen Informationen, die Umsetzung unterscheidet sich aber von Fall zu Fall, sodass im Folgenden jeder Indikator getrennt implementiert wird.

Indikatoren lexikalischer Ambiguität

Der Verdacht auf Ambiguität wird auf Grundlage von *Token* (ausgewählter semantischer Kategorien) getroffen, für die mehrere Lesarten in WordNet (s. Abschnitt 3.3.1.1) existieren. Genauer gesagt, sind das alle semantischen Kategorien mit Ausnahme von „Rolle“ und „Priorität“, die bereits von der IE-Komponente im Vorfeld extrahiert und gespeichert wurden (s. Abschnitt 5.3.2.1). Allerdings sind nicht alle *Token* gleich wichtig für die Entscheidung, ob die Disambiguierungskomponente angewendet werden soll oder nicht. Vielmehr gilt es, im Sinne der Verarbeitungsperformanz, zusätzlich auf Grundlage einer Stoppwortliste[130] und POS-*Tags* zu filtern.

Darüber hinaus ist noch offen, wie ein performanter Zugriff auf WordNet als Ressource sichergestellt werden kann. Ziel muss dabei sein, mit geringem Zeitaufwand an die Lesarten eines *Tokens* zu gelangen, was einen netzwerkbasierten Zugriff ausschließt. In dieser Arbeit wird deshalb auf die *Java WordNet Library* (JWNL)[131] zurückgegriffen. Hierbei handelt es sich um eine JAVA-Implementierung der WordNet API, die umfangreichen Datenzugriff auf WordNet sowie ähnliche Ressourcen ermöglicht. Ein wesentlicher Vorteil dieser Implementierung ist, dass die erforderlichen Dateien allesamt lokal bereitgestellt werden können, womit ein Netzwerkzugriff gänzlich entfällt und ein performanter Zugriff auf WordNet sichergestellt ist.

Indikatoren syntaktischer Ambiguität

Um **PP-Anbindungsambiguität** musterbasiert erkennen zu können, sind Informationen über die Bestandteile eines Satzes erforderlich (engl. *chunks*), wie Nominalphrasen, Verbalphrasen oder eben Präpositionalphrasen. Wie in Abschnitt 5.3.2.2 dargestellt, kann potentielle PP-Anbindungsambiguität anhand eines Musters wie „VP NP PP“ erkannt werden, wobei zu prüfen ist, ob es sich bei der Präposition um „*of*“ handelt, die als unzureichender Ambiguitätsindikator gilt. Um die hierzu erforderlichen Informationen (*Chunks*) zu erhalten, wird in dieser Arbeit der *OpenNLP Chunker* (Apache Software Foundation, 2012) eingesetzt.

Beispiel 7.3.3 (Musterbasierter Indikator syntaktischer Ambiguität)
Satz: „I want to press the button with the mouse“
(Annotierte) Chunks: NP ***VP NP PP*** *NP*

Wie in Beispiel 7.3.3 sichtbar wird, enthält die Sequenz das *a priori* definierte Muster und wird somit als potentiell ambig erkannt. Die Ausführungszeit des *OpenNLP Chunkers* ist dabei als sehr gut zu bezeichnen, da hier beispielsweise lediglich eine Zeit von vernachlässigbaren 0.001 Sekunden in Anspruch genommen wurde.

Im Falle der **Koordinationsambiguität** wird nicht auf die Ergebnisse von *OpenNLP* zurückgegriffen, sondern auf die POS-*Tags*, die bereits seit dem *Preprocessing* vorliegen. Ähnlich zu dem zuvor dargestellten Vorgehen, werden allerdings auch die POS-*Tags* einer Reihe von *Token* eines Satzes als Sequenz dargestellt und

[130] Die Stoppwortliste umfasst 534 Einträge (z. B. „*the*“, „*their*“, „*them*“, „*themselves*“) und basiert zu großen Teilen auf der Stoppwortliste, die von der Apache Foundation bereitgestellt wird. Siehe: `https://github.com/apache/lucene-solr/blob/master/lucene/analysis/common/src/resources/org/apache/lucene/analysis/snowball/english_stop.txt` (Stand: 12.01.17).

[131] Siehe weiterführend: `http://jwordnet.sourceforge.net/handbook.html` (Stand: 12.01.17).

auf Muster untersucht. Ein Beispiel für ein solches Muster ist „JJ NNS CC NNS“[132]. Wird dieses gefunden, ist eine potentielle Koordinationsambiguität gegeben.

Auch das mehrfache Vorkommen von Konjunktoren („*and*“, „*or*“) innerhalb eines Satzes kann auf diese Weise überprüft werden, was als weiteres Indiz für Ambiguität gilt. Dabei kann dieses Vorgehen als sehr performant bezeichnet werden, da nur auf bereits in den Daten direkt vorliegende Informationen zurückgegriffen wird.

Indikatoren referentieller Ambiguität

Die Erkennung potentieller **referentieller Ambiguität** wurde in Abschnitt 5.3.2.3 behandelt und wird, wie auch die syntaktische Ambiguität, durch einen musterbasierten Indikator umgesetzt. Der Musterabgleich basiert dabei ebenfalls auf POS-*Tags*. Eine Besonderheit ist, dass dieses Muster sich nicht zwangsläufig auf nur einen einzigen Satz bezieht, sondern mehrere Sätze umfassen kann. Genauer gesagt können Bestandteile des Musters unterschiedlichen Sätzen zugehörig sein, was eine paarweise Untersuchung der Sätze begründet (s. Abschnitt 5.3.2.3).

Um diese Besonderheit abzudecken, ist es erforderlich, den Indikator in der gesamten Anforderungsbeschreibung bzw. über alle relevanten, englischsprachigen Sätze hinweg zu suchen, wobei nur Sequenzen von aufeinanderfolgenden Sätzen berücksichtigt werden. Zunächst wird hierzu ein gegebener Satz S_i auf das Vorhandensein mindestens zweier Nomina untersucht, wobei beide Nomina deckungsgleich im Plural oder im Singular vorliegen müssen. Wird dieses Teilmuster festgestellt, wird sowohl in S_i als auch im darauffolgenden Satz S_{i+1} das Vorhandensein eines Pronomen geprüft, welches potentiell auf beide gefundenen Nomina in S_i referenzieren könnte und somit referentiell ambig wäre. Da es sich hierbei um ein iteratives Vorgehen handelt, wird Satz S_{i+1} im nächsten Schritt unabhängig von möglicher Fundstellen zum neuen Ausgangspunkt S_i und der vorherige S_{i+2} zum Untersuchungsgegenstand S_{i+1} etc.

Das Thema der **Erkennung potentieller Koreferenz** ist von dem bisher beschriebenen Vorgehen vor allem dadurch abzugrenzen, dass weitere lexikalische Informationen notwendig sind und der Indikator auf die vollständige Anforderungsbeschreibung[133] angewendet wird (s. Abschnitt 5.3.2.3). Genauer gesagt, werden die semantischen Kategorien betrachtet (s. Abschnitt 5.3.2.3), indem alle *Token* einer semantischen Kategorie gruppiert und untereinander verglichen werden. Übereinstimmende *Token* gelten als Indiz für Koreferenz, wobei einzelne Kategorien unterschiedlich verarbeitet werden, wie in Abschnitt 5.3.2.3 dargestellt. Hervorzuheben ist darüber hinaus die in Beispiel 7.3.4 dargestellte Synonymliste, die einbezogen wird, um eine größere Abdeckung zwischen *Token* gleicher semantischer Kategorien zu erhalten.

Beispiel 7.3.4 (Synonymliste, Auszug)

component=application, program, software, system;
object=server,host;
object=certificate,certification,credential,credentials;
object=spam,junk e-mail;
object=electronic mail,e-mail,email;

[132] Zu lesen als: Adjektiv, Nomen (Plural), Konjunktion, Nomen (Plural).
[133] Im Sinne aller relevanten, englischsprachigen Sätze.

In dieser prototypischen Umsetzung ist die Synonymliste manuell erstellt worden[134]. In der weiteren Entwicklung ist vorzusehen, diesen Erstellungsprozess in Teilen zu automatisieren. Hier ist eine Idee, den WSD-*Cache* der lexikalischen Disambiguierung um Synonyme zu erweitern und als Ressource heranzuziehen.

Indikatoren für Unvollständigkeit

Unvollständigkeit wird über das Fehlen der semantischen Kategorien Subjekt (Rolle, Komponente) und Objekt definiert (s. Abschnitt 5.3.2.4). Die hierzu erforderlichen Informationen liegen – wie auch bei den Indikatoren zuvor – bei der Indikatoranwendung bereits vor. So gestaltet sich die eigentliche Indikatoranwendung insofern einfach, als dass es zu prüfen gilt, welche semantischen Kategorien vorhanden sind bzw. welche fehlen. Dies erfolgt auf Satzbasis bzw. unter iterativer Verarbeitung einzelner *Token* eines Satzes. Hierbei ist eine Einschränkung, dass mindestens ein *Token* der Wortart Nomen als Objekt in einem Satz geführt werden muss. Diese zusätzliche Regel wird eingeführt, um Fehler der Anforderungsextraktion durch REaCT abzufangen[135].

In der prototypischen Umsetzung wird nicht berücksichtigt, dass mehrere Prädikate auch mehrere Objekte voraussetzen können. Derzeit ist es ausreichend, wenn ein Objekt im Satz vorhanden ist, auch wenn zwei Prädikate vorkommen und gegebenenfalls sogar ein Prädikat erst nach dem Objekt eingeführt wird. Dieser Umstand ist beim Subjekt weniger entscheidend, da in der Regel nur ein Subjekt angegeben wird.

7.3.2.4 Umsetzung der Strategien

Bei der Umsetzung der in Abschnitt 5.2 erarbeiteten Strategien ist zwischen den vordefinierten Strategien und der *Fallback*-Strategie zu unterscheiden, da sich die Abläufe und somit auch die Umsetzung jeweils anders darstellen.

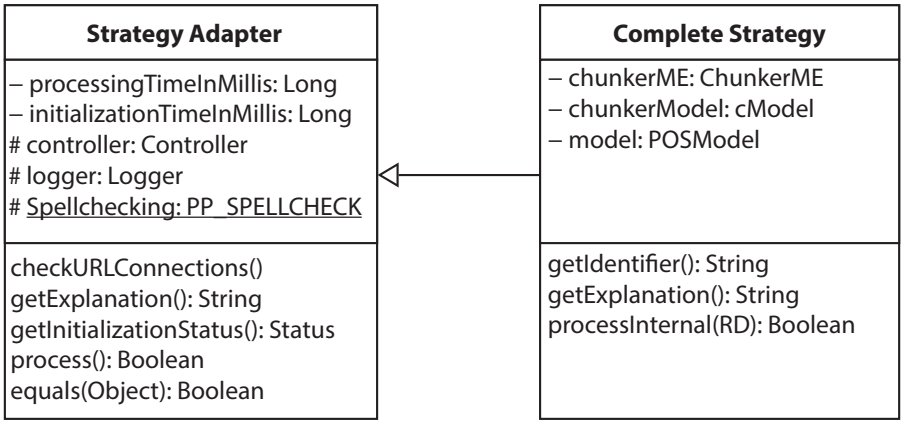


Abbildung 7.14: Vererbung von Struktur-/Verhaltensmerkmalen (kompakte Darstellung)

Das Konzept der Strategien basiert grundsätzlich auf einem Adapter. In diesem Fall handelt es sich um einen *Strategy Adapter*, der als Grundgerüst verschiedener Strategien fungiert (vgl. Abbildung 7.14). Diese Oberklasse vererbt ihre

[134]Unter Einbezug von WordNet als lexikalische Ressource (s. Abschnitt 3.3.1.1).
[135]Die IE-Komponente REaCT klassifiziert z. B. auch Verben oder Adjektive als Objekte.

Struktur-/Verhaltensmerkmale an die Unterklassen (wie z. B. die *Complete*-Strategie). Sie stellt somit einheitliche Metainformationen sicher und sorgt für interstrategische Vergleichbarkeit. Wie in Abbildung 7.14 erkennbar, ist beispielsweise beim *Strategy Adapter* die Methode *getExplanation()* vorgesehen, die eine natürlichsprachliche Erläuterung der Strategie enthält. Da diese Metainformation bei allen Strategien gleichermaßen vorhanden sein soll, ist eine Anordnung auf Ebene der Oberklasse sinnvoll. Weiterhin ermöglicht beispielsweise die Methode *equals()* explizit den Vergleich von zwei Strategieadaptern (auf Ebene der einzelnen Merkmale, die unterstützt werden) und *checkURLConnections()* prüft den Netzwerkverkehr zwischen einzelnen Verarbeitungskomponenten auf Funktionsfähigkeit.

Die einzelnen Strategien enthalten die sequenziell angeordneten Verarbeitungskomponenten sowie deren Anwendungsgegenstand (z. B. satzbasierte oder tokenbasierte Anwendung). Am Beispiel der *Complete*-Strategie sieht dies beispielsweise wie in Tabelle 7.4 aus. Es fällt auf, dass jede Strategie auch die für sich notwendigen mehrsprachigen Programmausgaben bereithält.

Attribut	**Ausprägung**
Identifier	• *Complete Strategy* • Vollständige Verarbeitungsstrategie
Explanation	• *Contains all available processing steps* • Enthält alle verfügbaren Verarbeitungsschritte
Processing Adapter	• Referential_Disambiguation (Beschreibung) • Syntactical_Disambiguation (Satz) • Incompleteness_Compensation (Satz) • Wordsense_Disambiguation (Satz) • Vagueness_Detection (Satz)

Tabelle 7.4: Attribute der *Complete*-Strategie

Der Sonderfall der *Fallback*-Strategie gestaltet sich ein wenig anders: Es handelt sich zwar grundsätzlich ebenfalls um einen *Strategy Adapter*, der zugehörige *Processing Adapter* wird aber erst zum Zeitpunkt der Anwendung gemäß der erkannten Indikatoren erstellt. Hierbei ist darauf hinzuweisen, dass auch bei der bedarfsgerechten Zusammenstellung der Verarbeitungskomponenten auf die Sequenz der Ausführung zu achten ist, da Abhängigkeiten zwischen den Komponenten bestehen oder Synergien genutzt werden können.

7.3.2.5 XML-Schnittstellen

Insgesamt sind zwei XML-Schnittstellen in dem vorliegenden Softwaresystem vorhanden: Die *Output*- und die *Info*-Schnittstelle. Beide Schnittstellen stehen der Weiterverarbeitung nach erfolgreichem Programmdurchlauf zur Verfügung, wobei keine persistente Speicherung der XML-Ausgaben stattfindet, es sei denn, diese Art der Speicherung wird explizit vom Endanwender gewählt (s. Abschnitt 7.3.3.2). Die *Output*-Schnittstelle ist für den Produktiveinsatz vorgesehen, da sie alle Ergebnisse strukturiert zur Verfügung stellt und somit die maschinelle Weiterverarbeitung der Anforderungsbeschreibungen ermöglicht (s. Abschnitt 5.5.7).

Als sehr hilfreich bei der Softwareentwicklung erwies sich darüber hinaus die *Info*-Schnittstelle, die für jede Komponente und somit für jeden Verarbeitungsschritt die in Anspruch genommene Zeit strukturiert wiedergibt (vgl. Beispiel 7.3.5). Diese Angaben sind sowohl für das *Preprocessing* als auch für die Erkennung und Kompensation vorhanden und ermöglichen es Entwicklern, Komponenten zu identifizieren, die nicht performant agieren und das Softwaresystem in der Verarbeitung bremsen.

Beispiel 7.3.5 (*Info*-Schnittstelle, Auszug)

```
<processing time="30573ms">
  <Referential_Disambiguation time="1588ms">
    <initialization time="0ms"/>
    <process time="1588ms"/>
  </Referential_Disambiguation>
  <Incompleteness_Compensation time="4168ms">
    <initialization time="456ms"/>
    <process time="3712ms"/>
  </Incompleteness_Compensation>
  <Word_Sense_Disambiguation time="24802ms">
    <initialization time="277ms"/>
    <process time="24525ms"/>
  </Word_Sense_Disambiguation>
</processing>
```

Wie zu erkennen ist, wird nicht nur die jeweilige Verarbeitungszeit wiedergegeben, sondern auch die Zeit, die gebraucht wird, um Verarbeitungskomponenten zu initialisieren. Dies ist hier von Interesse, da mit umfangreichen Ressourcen gearbeitet wird (z. B. Klassifikationsmodelle), deren Einlesen zeitintensiv ist. Auf diesem Wege lassen sich notwendige Optimierungen feststellen (z. B. Austausch von Ressourcen).

7.3.3 Datenschicht

Klassischerweise sorgt die Datenschicht „[...] dafür, dass die fachlichen Objekte dauerhaft gespeichert und auch wieder geladen werden können“ (Dunkel und Holitschke, 2003, S. 18), wobei sie aufgrund der losen Kopplung möglichst wenig über die anderen Schichten weiß (Dunkel und Holitschke, 2003, S. 18). Im Folgenden wird sowohl der Export interner Datenobjekte (s. Abschnitt 7.3.3.1) als auch die Speicherung von Verarbeitungsresultaten (*Output*) und Verarbeitungszeiten (*Info*) als XML-Dateien besprochen (s. Abschnitt 7.3.3.2).

7.3.3.1 Export instantiierter Datenklassen

Mit der Objektserialisierung[136] wird in dieser Arbeit das Ziel verfolgt, Zwischenergebnisse umfänglich verfügbar, nachvollziehbar und vergleichbar zu machen. Anders

[136] Umfangreiche Informationen zur Serialisierung in Java geben Krüger und Hansen (2014, S. 863 ff.).

als bei der Ergebnisausgabe an der Benutzer- und XML-Schnittstelle, handelt es sich hierbei um Kopien serialisierbarer Datenobjekte (z. B. ein *Token*) mitsamt allen Transformationen und Ergänzungen (z. B. Lesarten). Es sind demnach weit mehr Informationen verfügbar, womit eine maschinelle Weiterverarbeitung ermöglicht wird, die nicht nur auf den finalen Ergebnissen basiert, sondern auch die Zwischenergebnisse miteinbeziehen kann. Denkbar ist beispielsweise eine Applikation zur Ergebnisevaluation oder eine ergänzende Software, die der Fehlerfindung dient. Abbildung 7.15 zeigt hierfür einen ersten Entwurf einer Softwareapplikation, die in der Lage ist, mehrere Anforderungsbeschreibungen (*Description*-Objekte) zu vergleichen und somit gegenüberzustellen.

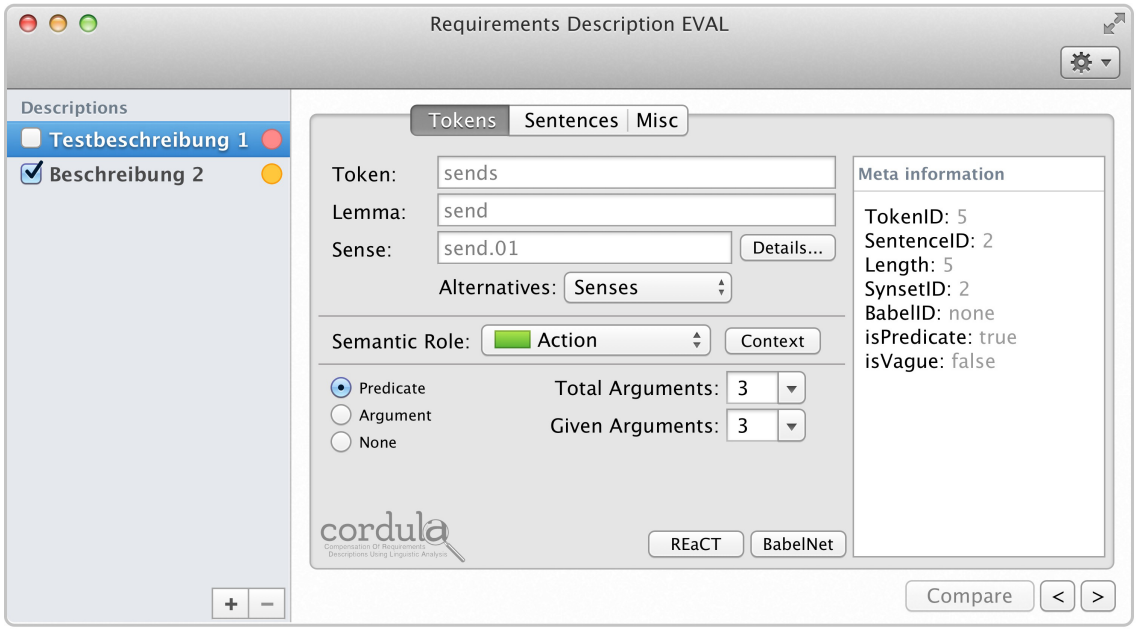


Abbildung 7.15: Gegenüberstellung von *Description*-Objekten

7.3.3.2 Persistente XML-Speicherung

Grundsätzlich ist die strukturierte Ausgabe zur maschinellen Weiterverarbeitung vorgesehen (s. Abschnitt 5.5.7), es besteht aber die Möglichkeit für die Endanwender, die Ergebnisse und Verarbeitungszeiten persistent auf dem Server zu speichern und bei Bedarf zu exportieren (s. Abschnitt 7.3.1.2). Dies kann beispielsweise einem späteren Vergleich verschiedener Ergebnisse dienen. Aus diesem Grund werden die in Abschnitt 7.3.2.5 skizzierten XML-Schnittstellen aufgerufen und die übermittelten Daten mit einem entsprechenden Zeitstempel und einer eindeutigen *Session ID* als Dateinamen gespeichert (vgl. Abbildung 7.16).

In einer späteren programmiertechnischen Umsetzung, die über diesen Prototypen hinausgeht und die beispielsweise auch Benutzerkonten vorsieht, ist eine benutzerspezifische Wahl der Dateinamen als sinnvoll zu erachten.

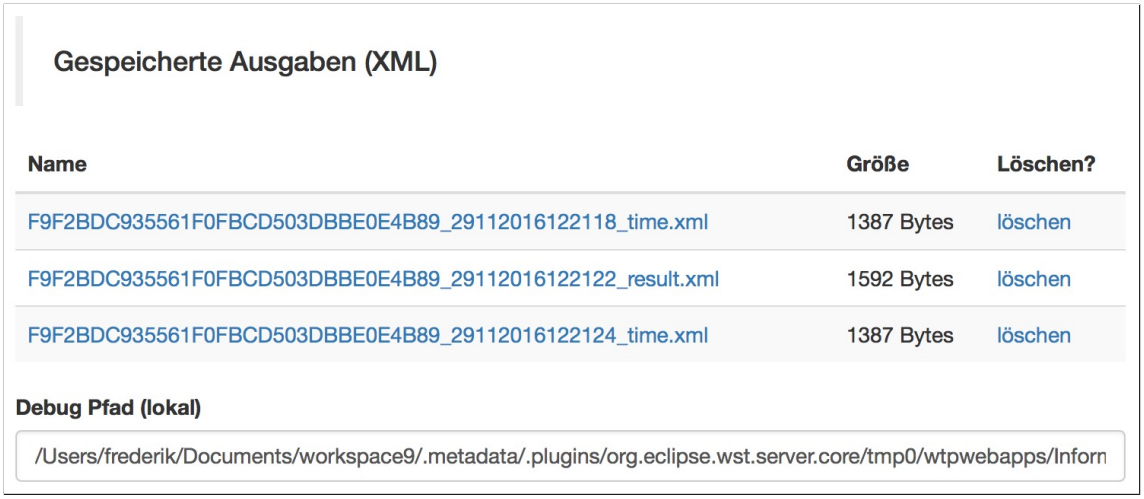


Abbildung 7.16: Protokollarchiv der Verarbeitungszeiten und Ergebnisse

7.4 Anforderungen an die Systemqualität

Zur Sicherstellung der Konzept- bzw. Systemqualität werden im Folgenden ausgewählte, vom *Software Engineering Institute* (SEI) benannte, Maßnahmen diskutiert, die in fünf Oberkategorien unterteilt werden (VSEK Konsortium, 2007c):

- Anforderungserfüllung (*Need Satisfaction Measures*)
- Leistung (*Performance Measures*)
- Wartbarkeit (*Maintenance Measures*)
- Adaptierbarkeit (*Adaptive Measures*)
- Wirtschaftlichkeit (*Organizational Measures*)

Berücksichtigung finden im Folgenden die Maßnahmen in den Kategorien Leistung, Wartbarkeit und Adaptierbarkeit. Dies bedeutet keineswegs, dass die Kategorien der Anforderungserfüllung sowie der Wirtschaftlichkeit nicht bedeutsam wären. Vielmehr handelt es sich hier um ein Systemkonzept mit prototypischer Umsetzung, bei der beispielsweise die Testbarkeit oder die zu erwartenden Betriebskosten nicht im Fokus der Überlegungen stehen.

7.4.1 Leistung

Die Kategorie Leistung beschreibt nach Vogel et al. (2009, S. 114 f.) Laufzeitanforderungen, welche „Qualitäten [umfassen], die die Akzeptanz des Systems beim Auftraggeber oder Benutzer beeinflussen“ (Vogel et al., 2009, S. 114 f.). Im Folgenden finden sich Überlegungen zum Leistungsverhalten (s. Abschnitt 7.4.1.1), zur Sicherheit (s. Abschnitt 7.4.1.2) und zur Bedienbarkeit (s. Abschnitt 7.4.1.3).

7.4.1.1 Leistungsverhalten

Vogel et al. (2009, S. 115) beschreiben Leistungsverhalten als das Leistungsvermögen eines Softwaresystems bei der Reaktion auf äußere Ereignisse, welches „wesentlich durch die Kommunikation an seinen internen und externen Schnittstellen bestimmt [wird]“ (Vogel et al., 2009, S. 115). Dies ist ein wesentlicher Aspekt bei dem vorliegenden Softwaresystem, besteht es doch mehrheitlich aus Einzelkomponenten und betreibt entsprechend intensive Schnittstellenkommunikation.

Im Zentrum der folgenden Überlegungen soll demnach nicht die „durchschnittliche Zeitdauer, die das System zur Bearbeitung eines Ereignisses braucht“ (Vogel et al., 2009, S. 115), stehen, sondern vielmehr die Frage, wie die Kommunikation zwischen Verarbeitungskomponenten hinsichtlich des Leistungsverhaltens zu gestalten ist.

Es existieren Verarbeitungskomponenten wie die Anforderungsextraktion, die in jedem Fall ausgeführt werden müssen (s. Abschnitt 5). Darüber hinaus ist bekannt, dass die Verarbeitungskomponenten unterschiedlich performant sind (vgl. Beispiele in Tabelle 7.5). Um einen Einblick in deren Performanz zu erhalten, wurden 50 zufällig gewählte Anforderungsbeschreibungen aus dem Anforderungsbeschreibungskorpus von Dollmann (2016) herangezogen und jeweils an die Verarbeitungskomponenten[137] zur Anforderungsextraktion, Unvollständigkeitskompensation sowie syntaktischen Disambiguierung übergeben. Tabelle 7.5 zeigt dafür die Antwortzeiten[138]. Wie zu erkennen ist, unterscheiden sich die Antwortzeiten der Komponenten nennenswert. Anhand von ausgewählten Designentscheidungen wird im Folgenden exemplarisch dargestellt, wie versucht wird, die Systemperformanz weiter zu optimieren.

	Min.	*Max.*	∅
Anforderungsextraktion	1.136	3.564	2.122
Unvollständigkeitskompensation	259	1.031	638
Syntaktische Disambiguierung	445	2.532	1.202

Tabelle 7.5: Performanz ausgewählter Verarbeitungskomponenten (in *ms*)

Die Anforderungsextraktion ist von den drei abgebildeten Verarbeitungskomponenten die mit der höchsten Antwortzeit. Sie ist aber elementar für das System und kann deshalb nicht weggelassen werden. Nichtsdestotrotz kann die Performanz verbessert werden, indem beispielsweise die Klassifikation von *On-* und *Off-Topic* Sätzen von der eigentlich Anforderungsextraktion gelöst wird – und somit beide Verarbeitungsschritte bedarfsgerecht einzeln abgerufen werden können. REaCT sieht im Original von Dollmann (2016, S. 57) vor, dass eingehende Anforderungsbeschreibungen zuerst auf nebensächliche Angaben überprüft werden. Dies ist aber nicht immer erforderlich[139]. Eine Trennung von Klassifikation und Extraktion ist demnach performanter.

Darüber hinaus erlaubt es die *Client-Server*-Architektur, Ressourcen der Komponenten (z. B. Klassifikationsmodelle), wie beispielsweise der syntaktischen Disambiguierung, im Arbeitsspeicher zu halten (ausreichende Speichergrößen vorausgesetzt)

[137] Verarbeitungskomponenten in Ausgangskonfiguration.
[138] Informationen zum Testsystem können Abschnitt 7.2 entnommen werden.
[139] Beispielsweise wenn ein Satz erneut der Anforderungsextraktion übergeben werden soll.

und deren Anwendung dadurch zu beschleunigen (abhängig von Hardwarekonfigurationen), da die Modelle nur beim erstmaligen Start geladen werden müssen.
Neben der Optimierung von Komponenten kann auch die Schnittstellenkommunikation optimiert werden, sodass die Kompensationsanfragen performanter werden oder ihre Anzahl minimiert wird. Grundsätzlich wird die Anzahl der Anfragen reduziert, indem bereits das *Preprocessing* nebensächliche Sätze und nicht-englischsprachige Sätze von der weiteren Verarbeitung ausschließt (s. Abschnitt 5.5.2). Performantere Kompensationsanfragen lassen sich darüber hinaus beispielsweise bei der Kompensation von Unvollständigkeit konfigurieren. Die Reduktion der Kompensationskandidaten auf die ähnlichsten Treffer (z. B. fünf Treffer anstatt alle) kann bereits die Performanz der Nachbearbeitung erhöhen. Auch sollte die Antwort nur das Datenfeld der gesuchten Information (Argument), nicht aber den gesamten Treffer enthalten.

7.4.1.2 Sicherheit

„Sicherheit ist eine nicht-funktionale Anforderung mit durchdringendem Charakter“ (Vogel et al., 2009, S. 116). Sie ist facettenreich und von hoher Praxisrelevanz. Vogel et al. (2009, S. 116) unterteilen Sicherheit in Vertraulichkeit (engl. *confidentiality*), Authentifizierung (engl. *authentication*), Integrität (engl. *integrity*), Privatsphäre (engl. *privacy*), Unleugbarkeit (engl. *non-repudiation*) sowie Schutz vor Zerstörung (engl. *intrusion protection*).

Bisher stand Sicherheit nur selten im Fokus der in dieser Arbeit angestellten Überlegungen. Beispielsweise beim Schutz vor fehlerhaften Daten, die zu einem gewollten Systemabsturz führen können, fand sie in Abschnitt 5.5.2 Erwähnung. Auch nicht alle von Vogel et al. (2009) benannten Sicherheitsbereiche sind zum Konzeptionszeitpunkt bereits von Bedeutung. Im Folgenden werden anfängliche Überlegungen zu den Bereichen Authentifizierung und Privatsphäre angestellt.

Authentifizierung

Authentifizierung ist „der Vorgang der Identitätsüberprüfung. Ein Benutzer beweist mithilfe von Berechtigungsnachweisen gegenüber der Authentifizierungsfunktion, dass er der ist, der er vorgibt zu sein“ (Vogel et al., 2009, S. 246). Ein etablierter Typ von Berechtigungsnachweisen ist nach Vogel et al. (2009, S. 246) die Kombination von Benutzername und Passwort, allerdings sind auch Zertifikate, Magnetstreifenkarten, Fingerabdrücke oder das Tippverhalten[140] geeignet.

In dieser Arbeit lässt sich Authentifizierung durch etablierte Techniken im Bereich der Benutzername-Passwort-Kombination umsetzen. Da es sich um ein *Client-Server*-Szenario handelt, ist eine Authentifizierung des Endanwenders gegenüber des *Servers* vor jeglichen weiterführenden Datenübertragung anzustreben (HTTP-Authentifizierung). Hierzu existiert beispielsweise der *defacto* Standard *hypertext access* (auch: .htaccess), welcher serverseitig die Authentifizierungskonfiguration für mehrere Endanwender ermöglicht. Aber auch komplexere sowie komfortablere Umsetzungen sind denkbar[141].

[140] Siehe bzgl. Tippverhalten die vielzitierten Arbeiten von Monrose und Rubin (1997).
[141] Siehe beispielsweise Cook (2002, S. 167 ff.).

Privatsphäre

Privatsphäre bedeutet, „dass Nachrichten, die zwischen zwei Systembausteinen ausgetauscht werden, auf dem Kommunikationspfad selber nicht gelesen, bzw. verstanden werden können“ (Vogel et al., 2009, S. 246). Um Privatsphäre zu erreichen, ist Verschlüsselungstechnologie auf diesem Kommunikationspfad notwendig. In dieser Arbeit wird dies ebenfalls mittels einer *Client-Server*-Architektur realisiert.

Endanwender kommunizieren über einen *Webbrowser* mit dem Softwaresystem. Für diese Kommunikation existieren Verschlüsselungsprotokolle, welche die verschlüsselte Übertragung von Daten über ein (unbekanntes) Netzwerk ermöglichen. Ein etablierter Standard ist die *Transport Layer Security* (TLS, zuvor *Secure Sockets Layer*), die beispielsweise im HTTPS-Kommunikationsprotokoll Anwendung findet. Hierbei baut der *Client* eine Verbindung zum *Server* auf und initiiert die Authentifizierung des *Servers*, welche auf einem Zertifikat basiert, das vom *Client* auf Gültigkeit geprüft wird. Daraufhin wird ein kryptographischer Schlüssel vereinbart, der sowohl zur Verschlüsselung als auch zur Prüfung von Authentizität verwendet wird[142].

7.4.1.3 Bedienbarkeit

Bedienbarkeit (engl. *usability*) umschreibt auf der einen Seite, wie und wie gut Endanwender das Softwaresystem mittels gegebener GUI bedienen können. Es beschreibt auf der anderen Seite aber auch, wie sich das System bezüglich der getroffenen Eingaben verhält (Vogel et al., 2009, S. 115). Zwar geschieht die Umsetzung des beschriebenen Konzepts nur prototypisch (s. Kapitel 7), nichtsdestotrotz ist die Bedienbarkeit eines Softwaresystems essentiell für dessen Akzeptanz.

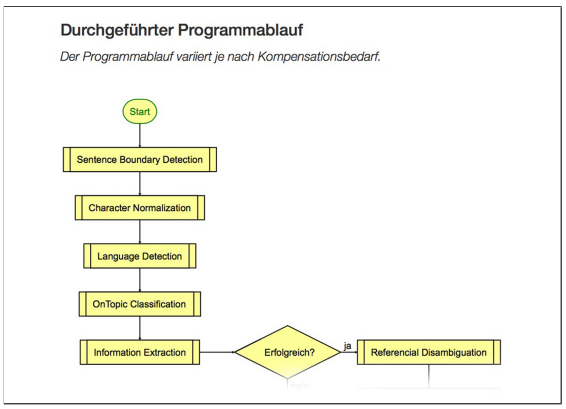


Abbildung 7.17: Programmablauf (GUI)

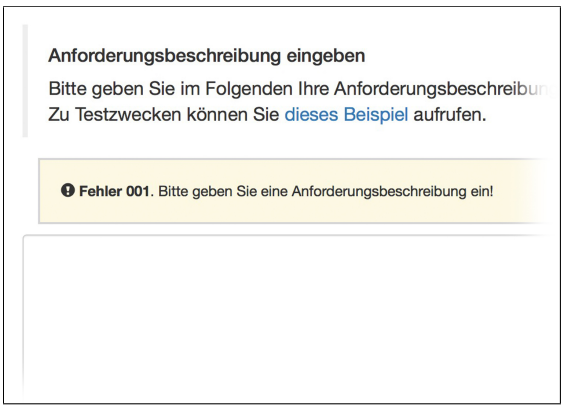


Abbildung 7.18: Fehlermeldung (GUI)

Grundsätzlich verhält sich die Interaktion mit dem beschriebenen System so, dass die Eingabe einer Anforderungsbeschreibung erforderlich ist, um eine kompensierte, strukturierte Anforderungsbeschreibung zurückzuerhalten. Hierzu steht eine einfache Eingabemaske bereit (vgl. Abbildung 7.18). Jede weitere Interaktion mit dem System (über eigene Programmmasken) dient der Ergebnisevaluation. Ziel ist es, die Komplexität des Gesamtsystems auf den Benutzerschnittstellen zu verbergen und eine nachvollziehbare Navigation zu ermöglichen. Als Beispiel hierfür wird die

[142] Stark vereinfachte Darstellung, für mehr Informationen bezüglich TLS siehe Ristić (2014).

Navigation in der erweiterten Ergebnisansicht herangezogen (vgl. Abbildung 7.17), die den individuell durchgeführten Programmablauf darstellt und somit zur Ansicht der Einzelergebnisse des Kompensationsprozesses dient. Die Transparenz der Verarbeitung wird durch die Darstellung des Verarbeitungsprozesses erhöht, indem Endanwender Schritt für Schritt das Ergebnis betrachten und nachvollziehen können.

Bezüglich der Benutzerschnittstellen merken Vogel et al. (2009, S. 115) an, dass barrierefreies Arbeiten es erfordert, alternative Benutzerschnittstellen (z. B. Sprachsteuerung) bereitzustellen, zumindest aber zu ermöglichen. In dieser Arbeit wird dieser Forderung durch Nutzung moderner Webtechnologien (s. Abschnitt 7.4.2.2) und strukturierten Ausgaben nachgekommen. In diesem Zusammenhang ist der fehlende „*Offline*-Modus“ des Systems als mögliche Einschränkung in der Nutzung zu nennen. Aufgrund der *Client-Server*-Architektur ist ein Arbeiten mit dem System nur bei bestehender Netzwerkverbindung möglich. Ein rein lokales Arbeiten ist derzeit nicht vorgesehen. Darüber hinaus sind Überlegungen anzustellen, wie das System auf fehlerhafte Eingaben reagiert. Kommt es zu Fehlern in der Verarbeitung, sind diese dem Endanwender transparent zu präsentieren. Da allerdings kein umfangreiches technisches Vorwissen erwartet werden darf, sind die Fehlermeldungen ohne Fachsprache zu verfassen. Abbildung 7.18 zeigt eine Fehlermeldung, die Endanwendern präsentiert wird, wenn keine Eingabe gemacht, die Weiterverarbeitung aber dennoch gestartet wird.

7.4.2 Adaptierbarkeit

In dieser Arbeit bezieht sich Adaptierbarkeit insbesondere auf die Fähigkeit des Konzepts bzw. des resultierenden Systems, auf einer weiteren Domäne und/oder Sprache angewandt zu werden. Adaptierbarkeit beschreibt dabei nach Hammer (2013) die „Anpassung eines Systems oder einer Applikation an einen Benutzer und/oder eine Aufgabe“ (Hammer, 2013, S. 6). Ergänzt werden kann diese Erklärung um den Aspekt der Systemintegration, wie in Definition 7.4.1 dargestellt wird.

Definition 7.4.1 (Adaptierbarkeit)
„Adaptierbarkeit ist die Eigenschaft von Software, an unterschiedliche funktionale Anforderungen anpassbar zu sein. Dies bezieht sich sowohl auf Anforderungen an die Funktionalität der Komponente als auch auf ihre Fähigkeit, mit unterschiedlichen Systemen zusammenzuarbeiten, d.h. eine Systemintegration zu ermöglichen“ (VSEK Konsortium, 2007a)

Adaptierbarkeit wird im Folgenden unterteilt in Interoperabilität (s. Abschnitt 7.4.2.1), Portabilität (s. Abschnitt 7.4.2.2), Skalierbarkeit (s. Abschnitt 7.4.2.3) und Wiederverwendbarkeit (s. Abschnitt 7.4.2.4).

7.4.2.1 Interoperabilität

Interoperabilität bezeichnet allgemein die „Fähigkeit unterschiedlicher Systeme, möglichst nahtlos zusammenzuarbeiten“ (Dudenredaktion, 2016, S. 934). Im Kontext dieser Arbeit wird darüber hinaus die Definition von Stempfle (1996) hinzugezogen, die den Aspekt der Standardisierung hervorhebt:

Definition 7.4.2 (Interoperabilität)
„Fähigkeit einer Systemkomponente, sich aufgrund genormter Schnittstellen in ein Gesamtsystem in der Weise integrieren zu lassen, daß ein ungehinderter, problemloser Austausch zwischen der eingebundenen Systemkomponente und dem Gesamtsystem stattfinden kann. [...] Interoperabilität ist damit eine Wirkung, ein Ergebnis konsequenter Umsetzung anerkannter Standards“ (Stempfle, 1996)

Ein hoher Grad an Interoperabilität ist dabei weit mehr als ein Zustand, den es in dieser Arbeit zu erreichen gilt. Interoperabilität ist strenggenommen vielmehr die zentrale Herausforderung. Schließlich handelt es sich um ein Softwaresystem, welches eine Vielzahl heterogener Softwarekomponenten im Sinne einer gemeinsamen Aufgabe zusammenführt. Dieser Gedanke wird auch bei Bues (1994) deutlich, der Interoperabilität als gegeben ansieht, wenn „[...] heterogene Systeme mit unterschiedlichen Zweckbestimmungen in einem Verbund zusammenwirken, so daß sie sich dem Benutzer wie ein einziges homogenes Leistungsgefüge darstellen“ (Bues, 1994, S. 27).

Im Bereich des NLPs besteht die Diskussion rund um Interoperabilität bereits seit Langem und erscheint durch Themen wie dynamische Ressourcen, Ressourcenintegration und *Semantic Web* auch weiterhin aktuell (z. B. Witt et al., 2009). Die vorliegende Arbeit greift dabei auf bestehende Errungenschaften zurück, indem die gesamte interne sowie externe Schnittstellengestaltung auf offenen, etablierten (*De-Facto-*)Standards beruht (z. B. CoNLL-U[143]). In dieser Arbeit resultiert dies in drei Designentscheidungen:

1. Die gesamte interne Kommunikation zwischen den Schnittstellen wird auf ein einheitliches, internes Datenmodell normalisiert.

2. Die Gesamtausgabe der Softwareapplikation nutzt für den plattform- und implementationsunabhängigen Austausch eine strukturierte Ausgabe.

3. Die Gesamtausgabe enthält alle vorliegenden Informationen, die zur Weiterverarbeitung benötigt werden könnten und die über die zentralen Verarbeitungsergebnisse hinausgehen können.

Um dies zu erreichen sind Konvertierungsprozesse notwendig. Beispielsweise gibt das REaCT-*Tool* zur Anforderungsextraktion (s. Abschnitt 5.5.3) standardmäßig eine CoNLL-strukturierte Ausgabe und wahlweise XML oder JSON aus. Ein entsprechender Konvertierungsprozess innerhalb des Softwaresystems transformiert die strukturierte Ausgabe in das intern genutzte Datenformat.

7.4.2.2 Portabilität

Portabilität wird im IT-Kontext oftmals nur im Sinne der Plattformunabhängigkeit[144] verwendet (z. B. Vogel et al., 2009, S. 116). Dies geht nach Bues (1994, S. 28) aber insofern nicht weit genug, als dass sich Portabilität auch auf Daten und Benutzeroberflächen beziehen kann. Im NLP-Kontext wird Portabilität insbesondere mit der Fragestellung verknüpft, inwieweit sich ein Softwaresystem auf weitere Domänen oder Sprachen übertragen lässt und welchen Aufwand dies bedarf.

[143] Siehe weiterführend: `http://universaldependencies.org/format.html` (Stand: 11.01.17).
[144] Fähigkeit von Softwareapplikationen, auf verschiedenen Systemplattformen ausgeführt zu werden.

Domäne und Sprache

Die Portabilität im Hinblick auf die **Domäne** (hier: Anforderungsbeschreibungen, RE) ist beim vorgestellten Softwaresystem vor allem auf Basis der genutzten Ressourcen zu diskutieren, da die meisten der eingesetzten NLP-Verfahren grundsätzlich domänenübergreifend anwendbar sind. Eine Ausnahme bildet die Komponente zur Anforderungsextraktion, die speziell für die Domäne der Anforderungsbeschreibungen entwickelt worden ist. Die Extraktion semantischer Kernkomponenten einer FA lässt sich schwer auf Domänen außerhalb natürlichsprachlicher Anforderungen übertragen.

Einen Überblick über Möglichkeiten der Portierung einzelner Komponenten auf Basis einbezogener NLP-Ressourcen gibt Tabelle 7.6. Angegeben sind Verarbeitungskomponenten sowie deren Portabilität auf eine andere Domäne. Darüber hinaus finden sich sowohl Angaben zum geschätzten Portierungsaufwand als auch zur Verfügbarkeit alternativer Ressourcen, die herangezogen werden können. Des Weiteren werden notwendige Verfahrenswechsel angezeigt.

Das *Preprocessing* ist vollumfänglich portabel: Sei es die Satzgrenzenerkennung, die Textbereinigung oder die Sprachenidentifikation – ein Domänenwechsel stellt keinen Aufwand dar, da kein Verfahrens- oder Ressourcenwechsel erforderlich ist. Identisch stellt sich die Situation bei der Kompensation von Vagheit und der syntaktischen Disambiguierung dar, die ebenfalls vollumfänglich portabel sind. In beiden Fällen entsteht daher kein Portierungsaufwand. Anders wiederum stellt sich dies bei den Komponenten der referentiellen und lexikalischen Disambiguierung dar: Beide Verfahren greifen auf domänenspezifische Ressourcen (z. B. *Black-* und *Whitelist*) zurück, um die Ergebnisqualität zu verbessern. Es ist demnach geringer Aufwand notwendig, um entsprechende Ressourcen an eine neue Domäne anzupassen.

	Portabel	Verfahrensw.	Alt. Ressourcen	Aufwand
Preprocessing	●	○	○	–
Anforderungsextraktion	○	●	●	Hoch
Lexikalische Disambiguierung	●	○	●	Gering
Syntaktische Disambiguierung	●	○	○	–
Referentielle Disambiguierung	●	○	●	Gering
Kompensation von Unvollständigkeit	●	○	●	Hoch
Kompensation von Vagheit	●	○	○	–
Strukturierte Ausgabe	○	●	●	Hoch

Tabelle 7.6: Domänenspezifische Portabilität einzelner Systemkomponenten

Im Vergleich dazu ist eine Portierung der Kompensation von Unvollständigkeit aufwändiger. Während das Verfahren grundsätzlich nicht auf die Domäne der Softwareanforderungen beschränkt ist, ist die zentrale Ressource (Suchmaschinenindex mit Kompensationstexten, s. Abschnitt 5.5.5) domänenspezifisch und muss für weitere

Domänen neu konzipiert werden. Dieser Aufwand ist als erheblich einzuschätzen, da die Datenakquise sowie -aufbereitung zeit- und arbeitsintensive Tätigkeiten sind.

Ein Verfahrenswechsel ist sowohl bei der Anforderungsextraktion als auch bei der strukturierten Ausgabe unerlässlich. Wie bereits angeführt, ist die Anforderungsextraktion als Ganzes ein domänenspezifisches Verfahren, welches nicht portiert werden kann. Ebenso verhält es sich mit der strukturierten Ausgabe, die zum einen stark von der Anforderungsextraktion abhängt und zum anderen das Ziel hat, strukturierte FA auszugeben, was ebenfalls schwer in eine andere Domäne zu portieren ist.

Anders als der Aspekt der Domäne ist die **Sprachabhängigkeit** unter zwei Gesichtspunkten zu diskutieren. Zum einen bezieht sie sich auf die Anzeigesprache, also die Sprache, in der das Softwaresystem mit dem Endanwender kommuniziert (z. B. Anleitungstexte, Bedienelemente). Zum anderen bezieht sie sich auf die Verarbeitungssprache, demnach auf die Sprache, die das System als Eingabe verarbeiten kann. Die Verarbeitungssprache ist dabei als ein weitaus komplexerer Diskussiongegenstand zu verstehen, da alle Systemkomponenten davon betroffen sind, während bei der Anzeigesprache lediglich die Benutzerschnittstelle einer Änderung bedarf.

Die derzeitige Anzeigesprache ist Deutsch. Zum jetzigen Zeitpunkt ist das gesamte Softwaresystem auf den Betrieb einer mehrsprachigen Benutzeroberfläche ausgelegt, wobei die Sprachen Englisch und Deutsch bereits vorkonfiguriert sind. Sollte eine Erweiterung um zusätzliche Sprachen erforderlich sein, ist eine Auslagerung dieser Konfigurationsmöglichkeit in externe Konfigurationsdateien sinnvoll, auf welche bei der prototypischen Umsetzung in dieser Arbeit verzichtet wird.

Bezugnehmend auf die Verarbeitungssprache empfiehlt sich eine erste Betrachtung auf Ebene der Systemkomponenten. Tabelle 7.7 listet diesbezüglich die verwendeten Komponenten zusammen mit den unterstützten Sprachen auf. Es zeigt sich, dass die einzelnen Verarbeitungskomponenten allesamt die englische Sprache unterstützen. Darüber hinaus verarbeiten mehrere der gewählten Einzelkomponenten (z. B. die lexikalische und die syntaktische Disambiguierung) bereits jetzt weitere Sprachen (z. B. Deutsch, Chinesisch). Die Fragen, die sich nun stellen, sind:

- Lassen sich die übrigen Komponenten um weitere Sprachen erweitern?
- Und falls nicht, existieren Alternativen, die implementiert werden könnten?

Im Falle des *Preprocessings* (s. Abschnitt 5.5.2), bestehend aus Einzelkomponenten zur Normalisierung, Textbereinigung, Grammatik- / Rechtschreibprüfung, Satzendeerkennung und Sprachenidentifikation, muss die Beantwortung der Fragen auf Basis der Einzelkomponenten erfolgen.

Während die Normalisierung, Textbereinigung und Sprachenidentifikation weitestgehend unabhängig von der zugrundeliegenden Verarbeitungssprache agieren[146], ist beispielsweise die Satzgrenzenerkennung abhängig von der jeweiligen Sprache (s. Anhang C.1.2). Selbiges gilt für die Grammatik- / Rechtschreibprüfung. Zwar unterstützen die verwendeten Komponenten bereits eine Vielzahl an Sprachen, eine Portierung auf eine weitere Sprache kann allerdings einen notwendigen Komponentenwechsel und damit einen Mehraufwand bedeuten.

[145] Die Komponente unterstützt insg. 271 Sprachen. Siehe Abschnitt 3.3.1.1.
[146] Die Sprachenidentifikation unterstützt 71 Sprachen und ist mit geringem Aufwand erweiterbar.

	Englisch	Deutsch	Chinesisch	Französisch	Spanisch	Arabisch
Preprocessing	●	●	○	●	●	○
Anforderungsextraktion	●	○	○	○	○	○
Lexikalische Disambiguierung[145]	●	●	●	●	●	●
Syntaktische Disambiguierung	●	●	●	●	◑	◑
Referentielle Disambiguierung	●	○	●	○	○	○
Kompensation von Unvollständigkeit	●	○	○	○	○	○
Kompensation von Vagheit	●	○	○	○	○	○
Strukturierte Ausgabe	●	○	○	○	○	○

Tabelle 7.7: Unterstützte Verarbeitungssprachen einzelner Komponenten.
● = unterstützt, ◑ = partiell unterstützt, ○ = nicht unterstützt

Ein Mehraufwand entsteht auch, wenn die Komponente zur Anforderungsextraktion an eine weitere Sprache angepasst werden sollte. Zum einen sind die Verarbeitungskomponenten innerhalb von REaCT (Dollmann und Geierhos, 2016) anzupassen (z. B. *Parser*). Zum anderen handelt es sich um ein Klassifikationsverfahren, dass auf Trainingsdaten angewiesen ist. Ein Wechsel der Sprache bedeutet demnach, dass auch eine ausreichende Anzahl annotierter Anforderungsdokumente vorliegen muss. Während dies erstens eine nennenswerte Herausforderung darstellt, ist es zweitens auch mit erheblichem Aufwand verbunden. Die gleiche Problematik mit den zugrundeliegenden Ressourcen ist bei der Portierung der Kompensation von Unvollständigkeit zu erwarten. Wie auch bei der Portierung auf eine weitere Domäne, ist der zugrundeliegende Suchmaschinenindex in Gänze zu ersetzen. Dies bedeutet, dass ein umfangreicher Datenbestand in der Zielsprache akquiriert, aufbereitet und annotiert werden muss, was mit hohem Aufwand verbunden ist.

Die Komponenten zur Disambiguierung unterscheiden sich wesentlich hinsichtlich der Portabilität: Während die lexikalische Disambiguierung eine Vielzahl an Sprachen unterstützt, damit hochgradig portierbar ist und keinen Mehraufwand erzeugt, geht eine Änderung bei der syntaktischen und referentiellen Disambiguierung mit einem wahrscheinlichen Verfahrenswechsel einher. Die syntaktische Disambiguierung unterstützt derzeit Englisch, Deutsch, Chinesisch und Französisch sowie Spanisch und teilweise Arabisch. Die referentielle Disambiguierung unterstützt Englisch und Chinesisch. Ein Verfahrenswechsel ist bei weiteren Sprachen unabdingbar.

Im Kontrast dazu steht die Kompensation von Vagheit, die auf linguistischen Regeln basiert. Gelten dieselben Regeln auch für die zu portierende Sprache, so können sie einfach übernommen werden. Sind sie allerdings nicht anwendbar, sind neue Regeln anzugeben, was etwas Mehraufwand bedeutet.

Ein geringer Mehraufwand ist auch bei der strukturierten Ausgabe zu erwarten, die ebenfalls an die weitere Sprache anzupassen ist. Wobei hier lediglich die Reihenfolge der erkannten semantischen Bausteine neu definiert werden muss, falls sie von der vordefinierten Reihenfolge abweicht (s. Abschnitt 1.3.1.2).

Eine Übersicht über den zu erwartenden Portierungsaufwand ist Tabelle 7.8 für die sechs meistgesprochenen Sprachen der Welt[147] zu entnehmen.

	Chinesisch	Hindi	Spanisch	Französisch	Arabisch	Russisch
Preprocessing	+	+	0	0	+	+
Anforderungsextraktion	+++	+++	+++	+++	+++	+++
Lexikalische Disambiguierung[148]	0	0	0	0	0	0
Syntaktische Disambiguierung	0	+++	++	+++	++	+++
Referentielle Disambiguierung	0	++	++	++	++	++
Kompensation von Unvollständigkeit	++	++	++	++	++	++
Kompensation von Vagheit	+	+	0	0	+	+
Strukturierte Ausgabe	+	+	+	+	+	+

Tabelle 7.8: Geschätzter Portierungsaufwand neuer Verarbeitungssprachen (hoher [+++], mittlerer [++], geringer [+], kein [0] Aufwand)

Systemplattform

Bei Plattformunabhängigkeit, also der Fähigkeit einer Softwareapplikation, ohne weitere Änderung auf einer Vielzahl an Rechnerarchitekturen ausgeführt werden zu können (Vogel et al., 2009, S. 116 f.), stellt sich bereits zu Beginn der Überlegungen eine elementare Frage: Plattformunabhängigkeit für wen? Denn die Anforderungen unterscheiden sich hier hinsichtlich der *Server*- und *Client*-Perspektive wesentlich: Endanwender interagieren über eine Benutzerschnittstelle mit dem Softwaresystem. Dabei liegt das System nicht lokal vor, sondern wird über einen *Server* bereitgestellt und über einen *Webbrowser* aufgerufen (*Client*). Für Endanwender ist Plattformunabhängigkeit daher insofern sichergestellt, als dass der Zugriff unabhängig von Betriebssystem und verwendeter Hardware (z. B. Computer, Mobiltelefon) erfolgt. Aus der Sicht der *Server*-Applikation gestaltet sich dies insofern anders, als dass diese auf einem zentralen Computer ausgeführt wird. Anders als beim Endanwender ist nicht von einer Vielzahl wechselnder Betriebssysteme und insbesondere nicht von mobilen Betriebssystemen (z. B. Android, iOS) auszugehen. Allerdings sind dennoch verschiedene Betriebssysteme (z. B. Windows, Linux) zu erwarten. Im Endeffekt bleibt die Herausforderung der Plattformunabhängigkeit somit grundsätzlich bestehen.

Plattformunabhängigkeit kann hierbei auf verschiedene Weisen erreicht werden[149], im Folgenden liegt der Fokus aber auf der plattformunabhängigen Entwicklung. Dies bedeutet, dass Anwendungen zum einen (überwiegend) unabhängig von der zugrundeliegenden Plattform ausführbar sind (z. B. Hybrid-*Apps*), sich an die Plattform anpassen können (z. B. durch *Fat Binaries*) oder auf Zwischencode (z. B. *Bytecode*) und entsprechenden Laufzeitumgebungen basieren.

[147] Nach Englisch als meistgesprochene Sprache. Siehe Statista (2016) für Details zu den Weltsprachen.
[148] Die Komponente unterstützt 271 Sprachen. Siehe Abschnitt 3.3.1.1.
[149] Weitreichende Überlegungen zur Portabilität finden sich in Hoffmann (2013, S. 107 ff.).

Ein populäres und etabliertes Beispiel für eine entsprechende Programmiersprache ist Java, wobei die Portabilität insbesondere durch die genaue Spezifikation elementarer Datentypen und dem Verzicht maschinennaher Datentypen und Operatoren gewährleistet wird (Krüger und Stark, 2009, S. 50). Die erforderliche Laufzeitumgebung (engl. *Java Runtime Environment, JRE*) ist für alle gängigen Betriebssysteme vorhanden. Kritisch anzumerken ist allerdings, dass es sich hierbei nur um eine begrenzte Plattformunabhängigkeit handelt, da sie im Tausch mit einer Abhängigkeit von der Laufzeitumgebung erzielt wird (Krüger und Stark, 2009, S. 50 f.).

In dieser Arbeit wird Java zur Entwicklung des Softwaresystems unter anderen aufgrund der weiten Verbreitung und Plattformunabhängigkeit herangezogen.

Benutzerschnittstelle

„Portierbarkeit mit Blick auf den Menschen, den Benutzer, bedeutet die Vereinheitlichung der Benutzeroberflächen, so daß ein Wechsel zwischen unterschiedlichen Systemen ohne einen zusätzlichen Lernaufwand und kurzfristige Effizienzverluste vollzogen werden kann“ (Bues, 1994, S. 28).

Diese Anforderung, die Bues (1994) an die Portabilität von Softwaresystemen stellt, ist auch heute von Bedeutung. Mit der zunehmenden Akzeptanz mobiler Endgeräte existiert eine Vielzahl an Softwareapplikationen, die auf verschiedenen Endgeräten und Betriebssystemen ausgeführt werden. Hierbei ist ein nahtloses Benutzungserlebnis von Bedeutung (s. z. B. Kadlec und Fröhlich, 2013, S. 142 f.).

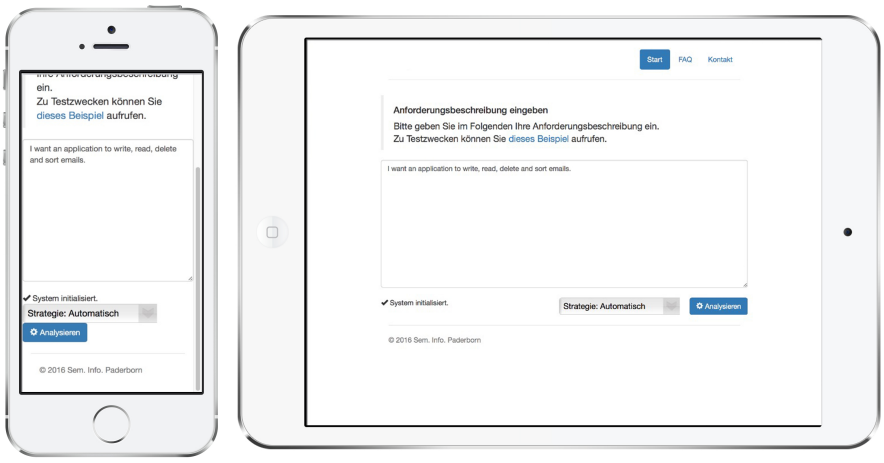


Abbildung 7.19: Softwaresystem mit responsivem Webdesign (GUI)

Moderne Webapplikationen reagieren auf die wechselnden Eigenschaften der Endgeräte (dies betrifft sowohl Eigenschaften wie die Bildschirmauflösung als auch Eingabemöglichkeiten) durch eine flexible Gestaltung, die sich an die gegebenen Umstände anpassen kann (responsives Webdesign). Moderne Webstandards wie HTML5, CSS3 und JavaScript bilden hierfür die Grundlage.

Das in dieser Arbeit konzipierte Softwaresystem nutzt als Benutzerschnittstelle eine Webapplikation, die diese Möglichkeit der flexiblen Darstellung nutzt, indem sich die Bedienelemente an die gegebene Bildschirmauflösung anpassen (s. Abschnitt 5.5).

Abbildung 7.19 zeigt dies exemplarisch anhand eines Mobiltelefons und eines Tabletcomputers. Für Endanwender ist es daher unerheblich, welches Betriebssystem und welches Endgerät verwendet wird, solange die Darstellung von Webinhalten möglich und die Unterstützung von modernen Webstandards gegeben ist.

7.4.2.3 Skalierbarkeit

„Systeme müssen [...] bei zunehmender Last adäquat reagieren, um ihre Dienste in einer definierten Güte anbieten zu können“ (Vogel et al., 2009, S. 117). Skalierbarkeit kann dabei vertikal oder horizontal erfolgen (Vossen et al., 2012, S. 14 f.), wobei beide Vorgehensweisen im Kontext dieser Arbeit vorstellbar und daher im Folgenden zu diskutieren sind. Grundsätzlich bezieht sich Skalierbarkeit hier auf die *Server*-Komponente in der *Client-Server*-Architektur.

Nach derzeitiger Konzeption werden alle Komponenten des Systems serverseitig bereitgestellt (vgl. Abbildung 7.20), was bedeutet, dass auf einem Computer (Knoten) alle notwendigen Komponenten ausgeführt werden[150].

Im Rahmen einer vertikalen Skalierung („*scale up*“) könnte dieser Computer durch einen leistungsstärkeren ersetzt oder durch Hardwareressourcen erweitert werden. Dies hat den Vorteil, dass an der eigentlichen Software keine Veränderung vorgenommen werden muss, da sich nur das Umfeld ändert – dies wäre ohne größere Umstände auch bei dem hier vorgestellten Konzept umzusetzen. Ein Nachteil ist jedoch, dass diese Skalierung einer natürlichen Grenze unterliegt: Irgendwann sind die besten Komponenten verbaut – eine weitere Skalierung ist nicht mehr möglich.

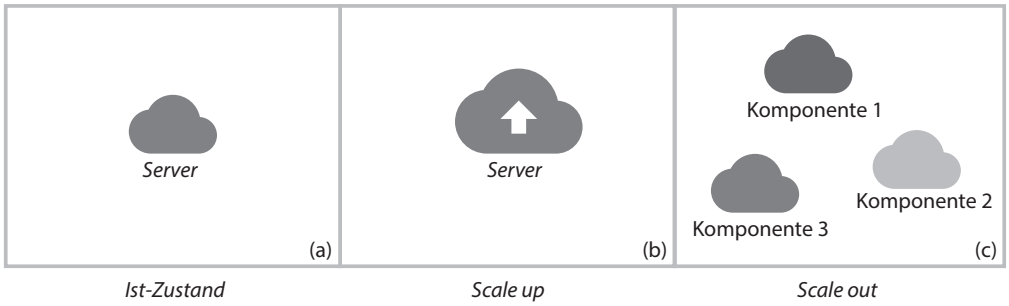


Abbildung 7.20: Möglichkeiten der Skalierbarkeit von Softwaresystemen. In Anlehnung an Vossen et al. (2012, S. 15)

Bei der horizontalen Skalierung („*scale out*“) wird das System erweitert, „[...] indem mehr Knoten hinzugefügt werden. Die Arbeit wird also »auf mehr Schultern« verteilt“ (Vossen et al., 2012, S. 14). Ob ein System für eine horizontale Skalierung in Frage kommt, hängt auch vom Grad möglicher Parallelisierung und Art der programmiertechnischen Umsetzung ab.

Das vorgestellte Konzept ist so aufgebaut, dass alle Komponenten parallel und isoliert ausgeführt werden können. Jede einzelne Komponente könnte so beispielsweise auf einem eigenen Knoten ausgeführt werden. Diese einzelnen Knoten wiederum könnten dann, bis zu einem gewissen Grad, bedarfsgerecht vertikal skaliert werden.

[150] Ausgehend von diesen Komponenten können weitere externe Ressourcen eingebunden werden.

7.4.2.4 Wiederverwendbarkeit und Nachhaltigkeit

Wiederverwendbarkeit bedeutet mit Bezug zu Softwaresystemen, dass sowohl einzelne Systemkomponenten als auch linguistische Ressourcen in diesem oder anderen Softwareprojekten (in gleicher Funktion) erneut zur Anwendung kommen können (Vogel et al., 2009, S. 117). Die hier verwendeten Systemkomponenten sind dabei als wiederverwendbar zu bezeichnen, da sie alle genau für diesen Einsatzzweck entwickelt worden sind (und daher z. B. notwendige Schnittstellen besitzen). Exemplarisch lässt sich dies an der Komponente zur syntaktischen Disambiguierung aufzeigen, die als externe Programmbibliothek für beliebige weitere Softwaresysteme herangezogen werden kann. Auch die Komponente zur Kompensation von Unvollständigkeit ist wiederverwendbar, wenngleich es sich nicht um eine einzige Programmbibliothek, sondern um ein vollständiges Kompensationssystem handelt, welches über Programmierschnittstellen zu steuern ist (Bäumer und Geierhos, 2016). Ähnliches gilt für die lexikalische Disambiguierung, die zwar als Programmbibliothek eingebunden werden kann, jedoch auf externe Dienste angewiesen ist. Hier wird die Wiederverwendbarkeit durch die Verfügbarkeit der externen Ressourcen möglicherweise eingeschränkt.

Hinsichtlich der Wiederverwendbarkeit genutzter linguistischer Ressourcen sind somit weitere Einschränkungen zu bedenken: So stellt sich bei extern eingebundenen Ressourcen die offensichtliche Frage nach der zukünftigen Verfügbarkeit. Im hier beschriebenen Softwaresystem ist die lexikalische Disambiguierung die einzige Komponente, die auf extern bereitgestellte Ressourcen zurückgreift und somit im Zweifel zu ersetzen ist, sollte die Verfügbarkeit nicht mehr gegeben sein (s. Abschnitt 7.1). Darüber hinaus ist zu bedenken, dass Ressourcen oftmals explizit für einen Verwendungszweck erstellt werden und gegebenenfalls für weitere Anwendungsszenarien ungeeignet sind: Modelle können speziell für eine Fragestellung trainiert worden sein oder eine Datenbank könnte aus einer speziellen Komposition von Informationen bestehen. In der Tat ist beides anzutreffen: Die Klassifikationsmodelle, die REaCT zur Anforderungsextraktion heranzieht, lassen sich zu genau diesem Zweck wiederverwenden, eignen sich aber nicht für andere Anwendungsszenarien. So verhält es sich auch mit dem Index, der zur Unvollständigkeitskompensation herangezogen wird: Eine sehr spezielle Ressource, die nur diesem einen Zweck dienen kann. Grundsätzlich können Drittapplikation sie aber einbinden – damit ist sie wiederverwendbar.

Unter **Nachhaltigkeit** (auch: Zukunftsfähigkeit) ist die Anwendbarkeit des beschriebenen Konzepts unter dem zeitlichen Aspekt zu verstehen oder als Frage formuliert: Wie lange ist das hier beschriebene Konzept in dieser Form und unter Einbezug der genannten Komponenten zielführend im Sinne der Verarbeitung von Anforderungsbeschreibungen einsetzbar? Diese Frage ist von besonderer Relevanz, da die natürliche Sprache einer stetigen Entwicklung unterliegt, was mit der Zeit unweigerlich die Anpassung der Methoden verlangt. Im Folgenden wird hierfür eine Lösung gesucht, indem die Komponenten einzelner Methoden unter mehreren (möglichst) objektiven Faktoren hinsichtlich möglicher Modifikation und Weiterentwicklung betrachtet werden.

Tabelle 7.9 listet Methoden zusammen mit ihrer angenommenen Nachhaltigkeit auf (Fazit). Diese ergibt sich dabei aus neun Merkmalen, die sich insbesondere auf die Verfügbarkeit von Quelltext und Ressourcen und deren Modifizierbarkeit beziehen. Ein Beispiel für eine Methode mit hoher Nachhaltigkeit ist die Sprachenidentifizie-

rung, für deren implementierte Komponente sowohl alle Ressourcen als auch der Quelltext frei verfügbar sind. Darüber hinaus sind Schnittstellen verfügbar, um neue Ressourcen zu erstellen. Dagegen ist eine niedrige Nachhaltigkeit der lexikalischen Disambiguierung wegen der Babelfy-Komponente zu attestieren, da, trotz aktiver Weiterentwicklung, eine Modifikation derzeit mangels zugänglichem Quelltext und Ressourcen nicht möglich und auch nicht vorgesehen ist. Dies bedeutet für die zukünftige Entwicklung des Softwaresystems, dass möglicherweise eine alternative WSD-Komponente zu implementieren ist (s. Abschnitt 3.3.1). Grundsätzlich ist das Konzept dabei bewusst modular aufgebaut, um den Austausch von Ressourcen und Komponenten jederzeit zu ermöglichen.

	Aktive Entwicklung	*Community*-getrieben	Quelltext verfügbar	Freie Lizenz	Ressourcen verfügbar	Ressourcen modifizierbar	Keine ext. Abhängigkeiten	Modifikationsaufwand gering	Modifikation vorgesehen	Fazit
Preprocessing										
Sprachenidentifizierung	●	●	●	●	●	●	●	●	●	Hoch
Rechtschreibkorrektur	●	●	●	●	●	●	●	○	●	Hoch
Grammatikprüfung	●	●	●	●	●	●	●	○	●	Hoch
Satzendeerkennung	●	○	●	●	●	●	●	○	●	Hoch
Anforderungsxtraktion										
Identifikation	○	○	○	–	○	●	●	○	○	Niedrig
Extraktion	○	○	○	–	○	●	●	○	○	Niedrig
Kompensation										
Lexikalische Disambiguierung	●	○	○	○	○	○	○	○	○	Niedrig
Syntaktische Disambiguierung	●	○	●	●	●	●	●	○	●	Hoch
Referentielle Disambiguierung	●	○	●	●	●	●	●	○	●	Hoch
Erkennung von Vagheit	○	○	●	●	–	–	●	●	●	Mittel
Unvollständigkeitskompensation	○	○	○	–	●	●	●	●	●	Mittel

Tabelle 7.9: Nachhaltigkeit einzelner Komponenten nach Methoden

7.4.3 Wartbarkeit und Erweiterbarkeit

Ein Softwaresystem zeichnet sich auch durch Wartbarkeit und Erweiterbarkeit aus. Während sich Wartbarkeit vor allem auf die Korrektur von Fehlern und gegebenenfalls Aktualisierung von Ressourcen bezieht, beschreibt Erweiterbarkeit primär das Hinzufügen oder den Austausch von Systemkomponenten aufgrund neuer oder geänderter Systemanforderungen (Vogel et al., 2009, S. 117). Um hohe Wartbarkeit und Erweiterbarkeit zu erreichen, sind nach Vogel et al. (2009, S. 117) die Prinzipien hoher Kohäsion und loser Kopplung zu befolgen. Während Kohäsion den Umstand beschreibt, dass idealtypisch einer Klasse oder Komponente eine definierte

Aufgabe zugeschrieben ist, beschreibt lose Kopplung, dass geringe Abhängigkeiten zwischen den einzelnen Komponenten eines Systems bestehen (s. Tabelle 7.10). Eine weitestgehend lose Kopplung wird in dieser Arbeit erreicht, indem einzelne Verarbeitungskomponenten über standardisierte Formate (gegebenenfalls durch Konverter) kommunizieren und ein zentrales Datenobjekt modifiziert wird. Der Wechsel einer Komponente stellt daher keine Beeinträchtigung anderer Komponenten dar, solange sich alle Komponenten an das definierte Austauschformat halten. Darüber hinaus ist hohe Kohäsion gegeben, da es sich von der grundsätzlichen Systemarchitektur bei den Verarbeitungskomponenten um Experten handelt, die explizit für definierte Aufgaben eingebunden werden. Eine konkrete Aufgabe, wie zum Beispiel die Kompensation von Unvollständigkeit, wird in der dafür eingebundenen Komponente vorgenommen. Auch innerhalb der Verarbeitungskomponenten sind die zugrundeliegenden Ressourcen weitestgehend zugänglich und modifizierbar. Auch hier stellt die lexikalische Disambiguierung eine Ausnahme dar, da die extern eingebundenen Ressourcen nur begrenzt modifizierbar sind. Als positive Beispiele sind die Kompensation von Unvollständigkeit und die Vagheitserkennung zu nennen, deren Ressourcen gänzlich modifizierbar sind.

Werden die zentralen Strategien unter dem Aspekt der Wartbarkeit und Erweiterbarkeit betrachtet, ergibt sich ein vergleichbares Bild wie bei den Komponenten und Ressourcen. Mit geringem Aufwand lassen sich neue Strategien hinzufügen oder bestehende Strategien modifizieren. Sie liegen, wie auch die Komponenten, als getrennte Systemeinheiten vor. Eine Änderung an einer der Strategien erfordert keine Änderungen an den sonstigen Komponenten.

	Methode	(Sub-)Komponenten
Preproc.	Satzgrenzenerkennung	*Stanford CoreNLP*
	Zeichennormalisierung	Java *Normalizer*
	Anforderungsklassifikation	REaCT
	Anforderungsextraktion	REaCT
		CoNLL *Converter*
		OpenNLP *Chunker*
Indikatoren	PP-Anbindung	*Chunker*
	Koordinationsambiguität	Syn. Muster
	PRP-Referenz	Syn. Muster
	Koreferenz	WordNet (JWNL)
		Synonymliste (Koreferenzen)
	Lexikalische Ambiguität	WordNet (JWNL)
		Stoppwortliste
Verarbeitung	Ref. Disambiguierung	*Stanford CoreNLP*
		Synonymliste (Referenzen)
	Syn. Disambiguierung	*Stanford CoreNLP*
	Unvollständigkeits-kompensation	Apache Solr
		Mate *Tools*
		CoNLL *Converter*
		Stanford CoreNLP
		Propbank
	Lex. Disambiguierung	BabelNet
		Babelfy
		WSD-*Caching*
	Vagheitserkennung	Wortliste, Regeln

Tabelle 7.10: Übersicht einzelner Systembestandteile

8 Evaluation

Im Folgenden wird das Gesamtsystem zwecks abschließender Bewertung evaluiert, wobei das Evaluationskonzept (s. Abschnitt 8.1) eine Zweiteilung vorsieht: Zuerst werden die Indikatoren und Strategien hinsichtlich ihrer Anwendbarkeit auf realen Anforderungsbeschreibungen evaluiert und eine Typisierung möglicher Fehler bei der Indikator- und Strategieanwendung vorgenommen (s. Abschnitt 8.2). Es folgt die Evaluation der Performanz des Systems und seiner Bestandteile (s. Abschnitt 8.3).

8.1 Evaluationskonzept

Das Softwaresystem ist mittels der folgenden „sach- und fachgerechte[n] Bewertung“ (Dudenredaktion, 2016, S. 559) sowohl hinsichtlich der Anwendbarkeit der Indikatoren und der Strategien (s. Abschnitt 8.2) als auch der Performanz (s. Abschnitt 8.3) zu untersuchen. Es handelt sich hierbei um eine summative Evaluation, welche auf die abschließende Bewertung des Ist-Zustands in dieser Arbeit abzielt. Thematisch wird bei der Evaluation der Strategieanwendung betrachtet, welche Strategiekonfigurationen unter unterschiedlichen Indikatorkombinationen zur Anwendung kommen und welche/wie häufig/unter welchen Umständen Fehler sowohl bei der Strategieanwendung als auch bei der Indikatorbestimmung auftreten. Neben der generellen Anwendbarkeit des vorgestellten Konzepts wird somit auch die Zuverlässigkeit der Indikatoren und der Strategiekonfiguration in Abhängigkeit variierender Texte untersucht. Bei der Evaluation der Performanz steht das Gesamtsystem sowie die Verarbeitungskomponenten im Fokus. Eine hohe Performanz gilt dabei als Qualitätsmerkmal eines benutzerfreundlichen Softwaresystems.

Während das methodische Vorgehen innerhalb der Evaluationsteile voneinander abweicht, ist die Struktur des zugrundeliegenden Protokolls einheitlich. So sieht dieses die Festlegung von **Evaluationsgegenständen** vor (z. B. das Zusammenwirken von Indikatoren und Strategien), aus denen sich der **Evaluationszweck** ableiten lässt (z. B. Fehleridentifikation). Aufbauend auf dem Zweck werden daraufhin **Evaluationsfragen** festgelegt, deren Beantwortung in einzelnen Abschnitten verteilt erfolgt und in beiden Teilen unter Hinzunahme eines **Evaluationskorpus** und eines geeigneten methodischen Vorgehens geschieht. Im Folgenden wird die Anwendbarkeit von Indikatoren und Strategien (s. Abschnitt 8.2) als erster der beiden Teile durchgeführt.

8.2 Evaluation der Anwendbarkeit von Strategien

Zu Beginn wird das Evaluationsprotokoll zur Strategienanwendbarkeit in Abschnitt 8.2.1 dargestellt. Es folgt die Entscheidungsevaluation der Strategieauswahl (s. Abschnitt 8.2.2), die aufzeigt, welche Strategien auf realen Anforde-

rungsbeschreibungen zur Anwendung kommen. Daraufhin wird die Indikatorzuverlässigkeit, welche die zuvor betrachtete Strategieanwendung maßgeblich beeinflusst (s. Abschnitt 8.2.3) sowie mögliche Fehlertypen (bei Strategien und Indikatoren) untersucht (s. Abschnitt 8.2.4).

8.2.1 Evaluationsprotokoll

Der Untersuchungsfokus liegt im Folgenden auf den in den Abschnitten 5.2 und 5.3 konzipierten Strategien und Indikatoren, wobei vor allem Abhängigkeiten und das Zusammenwirken von Indikatoren, Strategien und Methoden **Evaluationsgegenstände** sind. Ein Zusammenwirken findet dabei vor allem bei der Strategie- und Methodenanwendung statt, während sich Abhängigkeiten bei der Indikator- und Strategieanwendung finden. So sind die Ergebnisse der Indikatoranwendung beispielsweise maßgeblich dafür verantwortlich, welche Strategie vom *Selector* bei der Strategieauswahl herangezogen wird. Fehlerhafte Indikatoren wirken sich somit auf die Strategien aus, die gegebenenfalls notwendige Methoden nicht aktivieren können. Die Indikatorzuverlässigkeit wiederum ergibt sich aus der Zuverlässigkeit der zugrundeliegenden Merkmale, die teilweise von zusätzlichen *Tools* bereitgestellt werden und ebenfalls fehlerhaft sein können. Der **Evaluationszweck** lässt sich daher wie folgt zusammenfassen:

- Identifikation von Fehlerquellen in der Strategiewahl
- Analyse von Verarbeitungsfehlern und deren Auswirkungen auf die Strategieanwendung

Für diesen Evaluationszweck sind im Folgenden entsprechende **Evaluationsfragen** (`F`) dargestellt (s. Abschnitt 4.2.1):

- `F1`: Wie oft werden einzelne Strategien bei der Strategieauswahl herangezogen?
- `F2`: Inwiefern decken die vordefinierten Strategien die Indikatorkombinationen realer Anforderungsbeschreibungen ab?
- `F3`: Wie zuverlässig funktionieren die definierten Indikatoren auf realen Anforderungsbeschreibungen?
- `F4`: Welche Fehler beeinflussen die Indikatorzuverlässigkeit und erschweren somit die Strategieauswahl?
- `F5`: Welche Fehler beeinflussen die Strategieanwendung und verschlechtern somit das Gesamtergebnis?

Die Beantwortung dieser Fragen erfolgt auf einem **Evaluationskorpus**. Hierfür wird das in Abschnitt 6.1 vorgestellte Anforderungsbeschreibungskorpus hinzugezogen. Dabei werden 400 zufällige Anforderungsbeschreibungen aus dem Korpus im Umfang von zwei bis fünf Sätzen (100 Beschreibungen je Umfang) vom Softwaresystem verarbeitet.

Das **Vorgehen** umfasst die manuelle Auswertung der Fragen `F1-5` auf dem Testkorpus. Hierzu werden die 400 Anforderungsbeschreibungen sukzessive an das Softwaresystem übermittelt und die Verarbeitungsergebnisse protokolliert. Dies umfasst zum Beispiel die erfolgte Strategieauswahl oder die erkannten und der Entscheidung zugrunde liegenden Indikatoren. Die Evaluationsergebnisse werden im Folgenden detailliert dargestellt.

8.2.2 Evaluation der Strategieauswahl

In diesem Abschnitt wird untersucht, wie häufig sich der *Selector* bei der Strategieauswahl auf Grundlage der kontextsensitiven Indikatoren aus Abschnitt 5.3 hinsichtlich der einzelner Strategien zur Verarbeitung und Kompensation von Anforderungsbeschreibungen entscheidet. Grundsätzlich kann der *Selector* dabei zwischen vordefinierten Strategien oder einer eigenen Konfigurationsvariante (*Fallback*-Strategie) wählen. Vordefiniert sind die in Abschnitt 5.2 vorgestellten Konfigurationsvarianten **F1** (Evaluationsgegenstand), wobei die *Complete*-Strategie nicht bei der Strategieauswahl berücksichtigt wird, da sowohl die Vagheitserkennung als auch die erweiterte Ergebnisausgabe keine Evaluationsgegenstände sind. Abbildung 8.1 zeigt die Auswahlhäufigkeit der einzelnen Strategien.

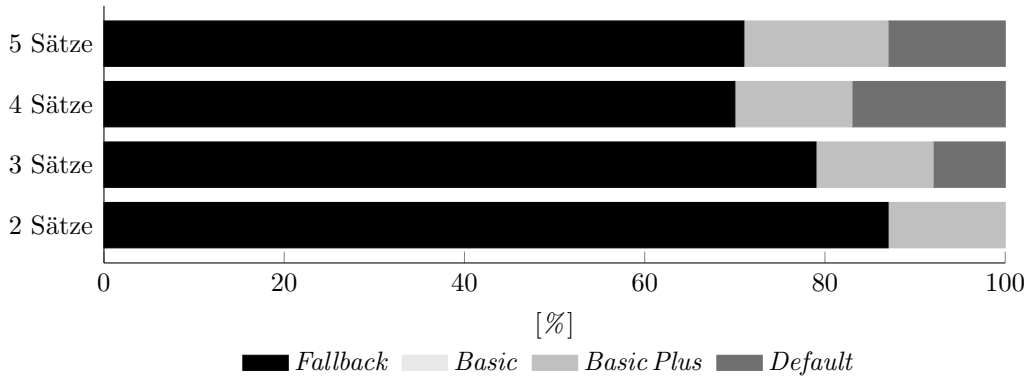


Abbildung 8.1: Auswahlhäufigkeit angewendeter Kompensationsstrategien

Es fällt auf, dass die *Fallback*-Strategie unabhängig vom Anforderungsbeschreibungsumfang im Durchschnitt 77% der Kompensationsdurchläufe abdeckt. Dies erscheint insbesondere angesichts der erwarteten Effizienzgewinne vordefinierter Strategien suboptimal. Andererseits ist nun nachgewiesen, dass die *Fallback*-Strategie **F2** als Rückfall-Strategie, für den Fall, dass keine vordefinierte Strategie hinsichtlich der gefundenen Indikatorkombination geeignet ist, greift. Das heißt aber auch im Umkehrschluss, dass kaum eine der vorher definierten Strategien im Echtfall Anwendung findet, da eine partielle Abdeckung einer Indikatorkombination seitens einer Strategiekonfiguration ausgeschlossen wurde (s. Abschnitt 5.2). Dies ist auf Grund der Möglichkeit, eigene Strategiekonfigurationen zu erstellen, unproblematisch.

Neben der *Fallback*-Strategie werden laut Abbildung 8.1 die *Basic Plus*- (14%) und die *Default*-Strategie (9%) auf die Anforderungsbeschreibungen angewendet, wobei die *Default*-Strategie erst ab einem Anforderungbeschreibungsumfang von drei Sätzen vom *Selector* hinzugezogen wird. Ferner fällt auf, dass die *Basic*-Strategie unabhängig

vom Anforderungsbeschreibungsumfang keine Berücksichtigung findet. Dies bedeutet, dass in keiner der evaluierten Anforderungsbeschreibungen einzig die Indikatorkombination der syntaktischen Disambiguierung und der Unvollständigkeitskompensation (SYN + INC) gefunden wurde. Werden neben der *Basic*-Strategie auch andere Indikatorkombinationen begutachtet (vgl. Tabelle 8.1), fällt auf, dass stattdessen die Kombination der lexikalischen Disambiguierung und der Unvollständigkeitskompensation (WSD + INC) vorkommt (97 Mal). Daneben kommt auch der Indikator für lexikalische Ambiguität (WSD) sehr oft vor (110 Mal), wird aber bislang von keiner vordefinierten Strategie abgedeckt, sodass die *Fallback*-Strategie mit eigenen Konfigurationsvarianten eingreift.

Häufigkeit	Indikatorkombination	Aktiv	Aktion
0	SYN + INC[151]	Ja	Wegfall
56	WSD + SYN + INC[152]	Ja	
37	WSD + SYN + INC + REF[153]	Ja	
33	WSD + SYN + REF	Nein	
38	WSD + SYN	Nein	
97	WSD + INC	Nein	Aufnahme
110	WSD	Nein	Aufnahme

Tabelle 8.1: Indikatorkombinationen und deren Häufigkeiten (Auszug)

Das häufige Vorkommen der Indikatorkombination WSD + INC sowie des Indikators WSD begründen die Aufnahme als neue Konfigurationsvariante, deren Auswirkung auf die einzelnen Strategiehäufigkeiten sich in Abbildung 8.2 ablesen lässt. So übernimmt die *Fallback*-Strategie in der neuen Strategiekonstellation nur noch 25% (zuvor 77%) der Anforderungsbeschreibungen, während die WSD-Strategie 28% und die WSD+INC-Strategie 24% übernehmen. Die *Basic*-Strategie wird vollständig verworfen, was aufgrund der ausbleibenden Anwendung (mangels Indikatorkombination) keine Ergebnisveränderung herbeiführt.

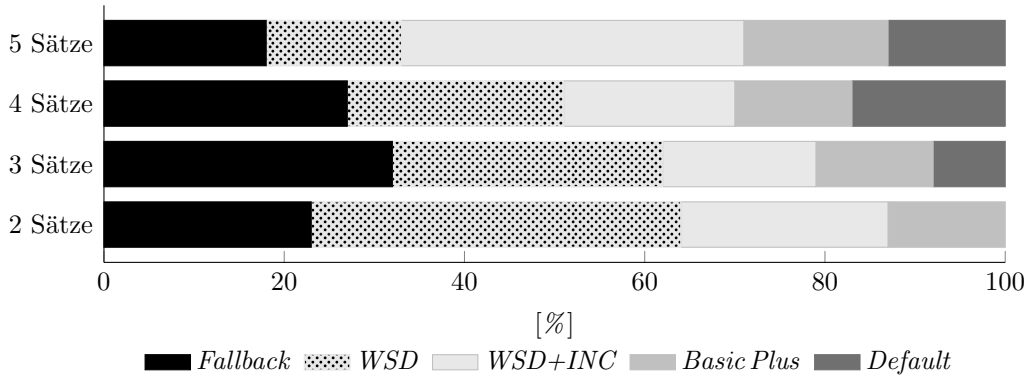


Abbildung 8.2: Aufteilung der Kompensationsstrategien nach Strategierevidierung

[151] Äquivalent zur *Basic*-Strategie (s. Abschnitt 5.2.2).
[152] Äquivalent zur *Basic Plus*-Strategie (s. Abschnitt 5.2.3).
[153] Äquivalent zur *Default*-Strategie (s. Abschnitt 5.2.4).

Bislang wurde nur die Aufteilung der angewendeten Strategien betrachtet, nicht aber, ob die zugrundeliegenden Entscheidungen auf Indikatorbasis auch korrekt waren. Dieser Frage wird im Folgenden Abschnitt nachgegangen.

8.2.3 Evaluation der Indikatorzuverlässigkeit

Die in Abschnitt 5.3 beschriebenen Indikatoren basieren auf linguistischen Merkmalen und Merkmalsmustern, die mittels verschiedener *Tools* ermittelt werden (z. B. *Chunking*, WordNet-Abgleich). Hierbei können Fehler auftreten, die zu falschen oder nicht erkannten Indikatoren führen. Eine hohe Indikatorzuverlässigkeit ist erreicht, wenn die erkannte Notwendigkeit einer Methodenanwendung überwiegend korrekt ist – die zugrundeliegenden *Tools* demnach zusammen mit definierten Merkmalen und Mustern eine geringe Fehlerquote aufweisen (s. Abschnitt 5.3.2.5). Abbildung 8.3 zeigt eine Erweiterung von Abbildung 5.13, welche die unterschiedlichen Indikatoren zusammen mit ihren zugrundeliegenden Merkmalsquellen darstellt.

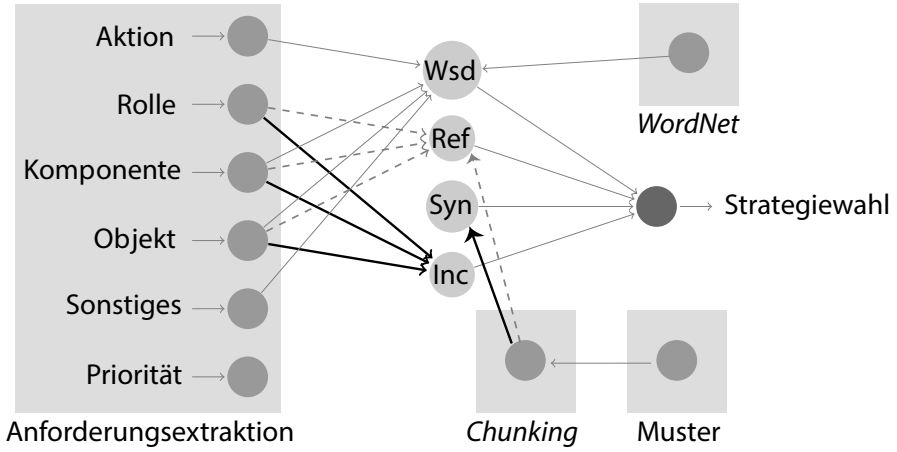


Abbildung 8.3: Indikatoren und ihre zugrundeliegenden Merkmalsquellen

Im Folgenden gilt es zu evaluieren, wie oft Fehler bei der Indikatorerkennung auftreten. Dies lässt sich dabei als binäres Problem verstehen: Entweder liegen Merkmale (z. B. für lexikalische Ambiguität) in der Anforderungsbeschreibung vor, oder sie liegen nicht vor. Die Akkuratheit (engl. *Accuracy*) der Indikatoren ließe sich dann darüber bestimmen, wie viele Anforderungsbeschreibungen in Relation zu allen Beschreibungen korrekt klassifiziert wurden:

$$a = \frac{\text{korrekt klassifizierte Beschreibungen}}{\text{alle betrachteten Beschreibungen}} \tag{8.1}$$

Im Umkehrschluss bedeutet das, dass die Fehlerrate f sich durch $f = 1 - a$ beschreiben lässt. Dieses Maß der Akkuratheit bzw. Fehlerrate ist allerdings als oberflächlich anzusehen, da vier mögliche Ergebniskombinationen existieren, wovon bisher nur zwei berücksichtigt werden. Diese vier Ergebniskombinationen sind:

- *True Positive* (TP): Merkmal liegt vor, der Indikator schlägt aus

- *False Positive* (FP): Merkmal liegt nicht vor, der Indikator schlägt aus
- *False Negative* (FN): Merkmal liegt vor, der Indikator schlägt nicht aus
- *True Negative* (TN): Merkmal liegt nicht vor, der Indikator schlägt nicht aus

Allerdings ist es auch für menschliche Leser nicht immer einfach, Merkmale zuverlässig zu identifizieren, sodass in dieser Arbeit drei (angeleitete) Evaluatoren die Anforderungsbeschreibungen evaluieren. Das Ergebnis des Softwaresystems wird dann mit dem gemeinsamen Ergebnis (Mehrheitsentscheid) der Evaluatoren verglichen. Bezogen auf die in Abschnitt 8.2.2 herangezogenen 400 Anforderungsbeschreibungen
F3 stellt Tabelle 8.2 die Ergebniskombinationen pro Indikator dar.

	TP	***TN***	***FP***	***FN***
INC	171	120	34	75
REF	156	170	11	63
SYN	142	179	22	57
WSD	400	0	0	0

Tabelle 8.2: Häufigkeit der Ergebniskombinationen

Wie angemerkt, ist ein „Globalwert für die [...] Genauigkeit nicht ausreichend“ (Carstensen et al., 2010, S. 155), da zum Beispiel auch von Interesse ist, wie zuverlässig das angewandte Verfahren arbeitet. An dieser Stelle wird auf zwei etablierte Evaluationsmaße, die vor allem im IR genutzt werden, zurückgegriffen (Carstensen et al., 2010, S. 155): *Recall* (r) und *Precision* (p).

$$r = \frac{TP}{TP + FN} \tag{8.2}$$

Der *Recall* gibt an, wie viele der Anforderungsbeschreibungen, die kompensationsbedürftig sind (d. h. in denen Merkmale einzelner Indikatoren vorliegen) gefunden werden. Ein niedriger *Recall* bedeutet demnach, dass viele defizitäre Anforderungsbeschreibungen übersehen werden (Carstensen et al., 2010, S. 586).

$$p = \frac{TP}{TP + FP} \tag{8.3}$$

Die *Precision* gibt an, wie häufig die als kompensationsbedürftig erkannten Anforderungsbeschreibungen auch wirklich kompensationsbedürftig sind; d. h. wie häufig ein vermeintlich erkannter Indikator wirklich vorliegt (Carstensen et al., 2010, S. 155).

Wie Carstensen et al. (2010, S. 155) zu bedenken geben, können „*Precision* und *Recall* [...] stark voneinander abweichen. Als einheitliches Gütemaß wird daher oft das als F-Maß (engl. F-*Score*) bekannte harmonische Mittel angegeben“ (Carstensen et al., 2010, S. 155). Um die Aussagekraft der erhobenen Evaluationswerte zu erhöhen, wird im Folgenden das harmonische Mittel herangezogen ($\beta = 1$).

$$F_\beta = (1 + \beta^2) \cdot \frac{p \cdot r}{(\beta^2 \cdot p) + r} \tag{8.4}$$

$$F_1 = \frac{2 \cdot p \cdot r}{p + r} \tag{8.5}$$

An dieser Stelle ist zu bedenken, dass die Hinzunahme einer (vermeintlich) nicht zwingend notwendigen Kompensationsmethode (gleichzusetzen mit *False Positive*) lediglich einen negativen Einfluss auf die Gesamtlaufzeit hat (s. Abschnitt 8.3), während das Nichtberücksichtigen eines Indikators zu einer Ergebnisverschlechterung führt (*False Negatives*), da die notwendige Kompensation ausbleibt. Wie aufgezeigt wurde, betrifft dies derzeit gleich mehrere Indikatoren. Es bietet sich an dieser Stelle demnach an, den F-*Score* hinsichtlich der höheren Gewichtung des *Recalls* zu modifizieren, um den Einfluss der *FN* stärker zu berücksichtigen ($\beta = 2$).

$$F_2 = \frac{5 \cdot p \cdot r}{4 \cdot p + r} \tag{8.6}$$

Tabelle 8.3 stellt die Ergebnisse der verschiedenen Evaluationsmaße dar.

	Accuracy	***Recall***	***Precision***	***F_1-Score***	***F_2-Score***
INC	0,73	0,70	0,83	0,76	**0,72**
REF	0,82	0,71	0,93	0,81	**0,75**
SYN	0,80	0,71	0,87	0,78	**0,74**
WSD	1,00	1,00	1,00	1,00	**1,00**

Tabelle 8.3: Ergebnisse der Indikatorevaluation

Wie Tabelle 8.3 zeigt, weist die Kompensation von Unvollständigkeit (INC) den niedrigsten F_2-*Score* auf, was insbesondere auf einen niedrigen *Recall*-Wert zurückzuführen ist. Im Vergleich dazu ist die referentielle Disambiguierung (REF) hinsichtlich des *Recalls* als gleichwertig zur Unvollständigkeit einzustufen, kann jedoch eine wesentlich höhere *Precision* aufweisen (Δ 0, 1), was sich allerdings nur gering auf den F_2-*Score* auswirkt (Δ 0, 03). An dieser Stelle ist wiederum ein deutlicher Unterschied in der *Accuracy* zu erkennen (Δ 0, 09).

Die syntaktische Disambiguierung (SYN) weist eine hohe *Precision* und einen guten *Recall* auf. Demgegenüber scheint nur die lexikalische Disambiguierung noch fehlerfreier zu arbeiten, was allerdings nur bedingt stimmt: Lexikalische Ambiguität ist auf Grundlage einzelner *Token* festzustellen, die Indikatoren attestieren jedoch für die gesamte Anforderungsbeschreibung Ambiguität und Unvollständigkeit. Es ist daher sehr wahrscheinlich, dass in allen Beschreibungen mindestens ein ambiges *Token* korrekt als ambig erkannt wird und somit sowohl der *Recall* als auch die *Precision* sehr gut sind. Nichtsdestotrotz kommen auch hier Fehler in der Indikatoranwendung vor[154], die derzeit nur nicht sichtbar sind. Der WSD-Indikator stellt somit einen Sonderfall dar, der in Abschnitt 8.2.4.1 besprochen wird.

[154]Die Fehler entstehen bspw. durch fehlende Einträge in den zugrundeliegenden Ressourcen. Dass Einträge in WordNet fehlen können, wurde dabei bereits in Abschnitt 3.3.1.2 angemerkt.

Bei der Interpretation dieser Ergebnisse ist zu berücksichtigen, dass die Ergebnisgüte unmittelbar von der Text- und Evaluationskomplexität abhängt, was bedeutet, dass für einfachere Evaluationsgegenstände beispielsweise ein F_1-*Score* von 100% als sehr gut gilt, während bei komplexen Gegenständen bereits 60–70% als sehr gut bezeichnet werden können (Carstensen et al., 2010, S. 586). Hierzu merken Carstensen et al. (2010, S. 586) an, dass „auch Menschen [...] nicht in der Lage [sind], bei der Analyse von komplexen Texten sowohl 100% Vollständigkeit als auch 100% Präzision zu erreichen“ (Carstensen et al., 2010, S. 586). Um die aufgetretenen Fehler sowie die Evaluationsergebnisse besser nachzuvollziehen, wird in Abschnitt 8.2.4 eine Evaluation der Fehlertypen vorgenommen.

8.2.4 Evaluation möglicher Fehlertypen

Die Zuverlässigkeit von Indikatoren und Strategien wird von unterschiedlichen Fehlern negativ beeinflusst, die im Folgenden besprochen werden. Dabei ist das Ziel, bestimmte Fehlertypen auszumachen und die jeweilige Auswirkung auf die Zuverlässigkeit der Indikatoren (s. Abschnitt 8.2.4.1) und Strategien (s. Abschnitt 8.2.4.2) abzuschätzen.

8.2.4.1 Indikatoranwendung

F4

Wie sich in Abschnitt 8.2.3 zeigt, treten bei der Indikatoranwendung Fehler auf. Die Frage, die nun beantwortet werden muss, ist: Welche Fehlerarten beeinflussen die Indikatorzuverlässigkeit und erschweren somit die Strategieauswahl? Ausgehend von diesen Fehlertypen ist in der Weiterentwicklung des hier beschriebenen Softwaresystems beispielsweise über zusätzliche Schritte der Qualitätssicherung nachzudenken.

Indikatoren für Unvollständigkeit

Partielle Unvollständigkeit wird über fehlende Details (semantische Kategorien) erkannt (s. Abschnitt 5.3.2.4). Dabei gilt Unvollständigkeit als gegeben, wenn die semantischen Kategorien der Rolle oder Komponente (Subjekt) fehlen oder aber das Objekt einer FA fehlt. Während die Kompensation von Unvollständigkeit auf SRL-Verfahren zurückgreift, um die Argumente eines Prädikats als Kompensationskandidaten zu extrahieren, bedient sich der Indikator demnach der, von der Anforderungsextraktion bereitgestellten, semantischen Kategorien (bzw. der Information über deren Fehlen). Hierbei kann es zu unterschiedlichen Fehlern kommen. Diese treten allerdings allesamt ursprünglich nicht beim Indikator, sondern bereits bei der Anforderungsextraktion auf. Zum einen können semantische Kategorien erkannt werden, die allerdings in der Anforderungsbeschreibung so nicht existieren (fehlerhafte Erkennung). Dies ist beispielsweise bei „*Export to adobe pdf format would be nice*“ der Fall, wo fälschlicherweise ein Subjekt erkannt wird („*Export*“). Zum anderen können einzelne semantische Kategorien nicht erkannt werden (ausbleibende Erkennung), wie beispielsweise die Rolle „*We*“ in „*We would like to add a column for external links [...]*“. Im Hinblick auf die Indikatorzuverlässigkeit ist die ausbleibende Erkennung semantischer Kategorien weniger schädlich für das Gesamtergebnis, da somit im schlimmsten Fall die Kompensation von Unvollständigkeit initiiert werden würde, was lediglich eine schlechte Performanz (Gesamtlaufzeit) bedeuten würde.

Die genauere Betrachtung der Evaluationsergebnisse zeigt dabei, dass beide Fehlertypen bei der Anwendung auf realen Daten vorzufinden sind. Der hohe Freiheitsgrad in den Formulierungen führt dabei oftmals zu Unvollständigkeit, beispielsweise wenn Auslassungen bestehen wie in „*Would like to see an option to Open and Close archive*“ (fehlende Rolle). Hierbei fällt auf, dass sehr oft das Subjekt in Anforderungsbeschreibungen ausgelassen wird und der Indikator dadurch aktiviert wird.

Wie in Abschnitt 5.3.2.4 beschrieben, ist das Objekt nicht immer erforderlich um einen wohlgeformten Satz zu bilden und dennoch wird es vom Indikator als notwendig angesehen. Dies ermöglicht es, in grammatikalisch fehlerhaften Sätzen wie „*I need a software which can be only play mp3 files on Android but the user not allow to copy or send with bluetooth*“ zu erkennen, dass kein Zusammenhang zwischen dem zu Beginn eingeführten Objekt „*mp3 files*“ und den Aktionen „*copy*“ und „*send*“ festgestellt werden kann. Dies führt allerdings auch dazu, dass Prädikate als unvollständig erkannt werden, die kein Objekt benötigen: „*App should be able to run at startup*“. Wie die Evaluation zeigt, sind diese im Anforderungskontext aber die Ausnahme. Ein weiteres Beispiel ist der Satz: „$As\ an\ administrator_{Rolle}\ when\ I_{Rolle}\ start\ a\ game\ I_{Rolle}\ should\ be\ able\ to\ put\ the\ maximum_{Objekt}\ duration_{Objekt}\ of\ the\ game$“. Interessant an diesem Beispiel ist zum einen, dass keine Unvollständigkeit erkannt wird, was korrekt ist. Zum anderen, dass syntaktische Ambiguität aufgrund der PP-Anbindung vorliegt: Die PP „*of the game*“ wird nicht an die NP „*maximum duration*“ gebunden (und ist somit nicht Teil des Objekts), sondern an die VP. In diesem Fall hat diese syntaktische Ambiguität keinen negativen Einfluss auf den Unvollständigkeitsindikator, es zeigt aber, dass die Indikatoren für bestimmte linguistische Phänomene von eben diesen negativ beeinflusst werden können.

Indikatoren referentieller Ambiguität und Koreferenz

Während der Evaluation haben sich beim Indikator für referentielle Ambiguität und Koreferenz drei Fehlertypen herauskristallisiert: Fehlerhafte POS-*Tags*, fehlerhafte semantische Kategorien und fehlende Synonyme.

Die Erkennung potentiell ambiger Referenzen basiert in dieser Arbeit maßgeblich auf POS-*Tags*, die als Muster (z. B. „NN+NN+PRP“) satzübergreifende Anwendung finden (s. Abschnitt 5.3.2.3). Allerdings zeigt sich, dass der zugrundeliegende POS-*Tagger* nicht fehlerfrei arbeitet. Dies ist nicht verwunderlich, führt jedoch zu Fehlern, da die (in der Anzahl limitierten) definierten Muster des Indikators nicht zuverlässig greifen. Ein Wechsel auf einen anderen POS-*Tagger* wäre zwar möglich, allerdings weisen alle aktuellen *Tagger* eine gewisse Fehlerquote auf (ACL Wiki, 2016).

Zur Erkennung von Koreferenz werden über die POS-*Tags* hinaus die semantischen Kategorien und Ähnlichkeitswerte (der semantisch annotierten Wörter) herangezogen (s. Abschnitt 5.3.2.3). Hierbei zeigt sich, dass zwei Fehlertypen gehäuft auftreten und zu falschen Ergebnissen führen können: Zum einen sorgen fehlerhafte semantische Kategorien dafür, dass ein Vergleich zweier potentiell koreferenter Wörter ausbleibt, da sie nicht in derselben semantischen Kategorie auftreten (z. B. wird „*Application*“ als Komponente und einmal fälschlicherweise als Objekt erkannt). Da die semantischen Kategorien von der Expertenkomponente zur Anforderungsextraktion (REaCT) bereitgestellt werden, ist auf deren Qualität zu diesem Zeitpunkt zu vertrauen. Maßnahmen zur Qualitätsverbesserung sind somit bei der Anforderungsex-

traktion vorzunehmen. Zum anderen ist die zugrundeliegende Synonymliste (derzeit manuell) zu erweitern, da zwar zum Beispiel „*System*“ auf „*Application*“ verweist, in den Beschreibungen aber oftmals von „*Apps*“ gesprochen wird und daher keine Übereinstimmung gefunden wird. Im Vergleich zu den fehlerhaft erkannten semantischen Kategorien ist dieses Defizit einfach zu beheben (Aufnahme weiterer Synonyme in die Liste). Das ergänzende Einbinden externer Ressourcen (z. B. WordNet) kann eine höhere Abdeckung ermöglichen, hätte allerdings in diesem Beispiel („*Apps*“) nicht geholfen, da kein Eintrag hierzu vorliegt. Eine kombinierte Lösung aus Synonymliste und WordNet erscheint dennoch auf Grund der höheren Abdeckung grundsätzlich zielführend, wenngleich eine manuelle Ergänzung unvermeidbar ist. Bestehen bleiben Rechtschreibfehler als Fehlerquelle, die derzeit einen Abgleich mit der Synonymliste und WordNet behindern.

Indikatoren syntaktischer Ambiguität

Die bereits beim Indikator für referentielle Ambiguität und Koreferenz genannte Fehlerquote des POS-*Taggers* kann auch beim Indikator für syntaktische Ambiguität zu Fehlern führen, da auch dieser auf POS-*Tags* beim Musterabgleich zurückgreift. Zusätzlich werden *Chunks* im Musterabgleich hinzugezogen, die ebenfalls fehlerhaft sein können (vgl. Abbildung 8.3). Somit ist sowohl die Erkennung von Koordinationsambiguität (POS-*Tags*) als auch die Erkennung von Präpositionalphrasen (mittels *Chunks*) grundsätzlich fehleranfällig. Im Falle der Erkennung von Koordinationsambiguität sind POS-*Tags* die einzigen Anhaltspunkte, die berücksichtigt werden. Fehlerhafte *Tags* führen daher zu falschen Entscheidungen. Insbesondere die Erkennung von Konjunktionen ist für einen performanten Indikator bedeutsam, da diese als Vorauswahlkriterium herangezogen werden. Hierzu zeigt die Evaluation, dass die Wörter „*and*“ und „*or*“ alle richtigerweise als Konjunktionen annotiert wurden. Andersherum wurden auch alle annotierten Konjunktionen korrekt erkannt. Die drei häufigsten Konjunktionen sind dabei „*and*“ (310 Treffer), „*or*“ (294 Treffer) und „*but*“ (218 Treffer). Ein vom Indikator aufgrund von Konjunktionen vorselektierter Satz ist: „*This software must be able to create* $\boldsymbol{and}_{CC}$ *deliver highly targeted* $\boldsymbol{and}_{CC}$ *personalized messages*“.

Fehlerhaft erkannte Sätze sind vor allem in der Kombination von Modifikatoren und Konjunktionen vorzufinden. So wurde der Satz „*Please add an option to clean search window text on* $\boldsymbol{safe}_{JJ}$ ***lock or application*** *minimize*“ vom Indikator als potentiell ambig eingestuft, da „*safe*“ als Adjektiv (Modifikator) und nicht als Nomen erkannt wird. Und auch der Satz „*App should be able to play songs from my favorite artists via* $\boldsymbol{amazon}_{JJ}$ ***music and winamp***“ gilt fälschlicherweise als potentiell ambig, da „*amazon*“ als Adjektiv (Modifikator) annotiert wurde.

Im Hinblick auf potentielle Ambiguität durch PP-Anbindungen führen fehlerhafte *Chunks* (z. B. NP statt VP) zu Fehlern, da die definierten Muster fälschlicherweise oder nicht mehr angewendet werden. Ein Beispiel ist der Satz „*The message displayed could be* $\boldsymbol{turned}_{VP}$ $\boldsymbol{on/off}_{NP}$ $\boldsymbol{in}_{PP}$ *preferences if desired*“, in dem „on/off“ fälschlicherweise als NP erkannt wurde. Dabei erzielt der OpenNLP *Chunker* in Performanzgegenüberstellungen bereits überzeugende Ergebnisse (z. B. Pinto et al., 2016), sodass ein Wechsel auf alternative *Chunker* nicht unbedingt eine Verbesserung vermuten lässt.

Während der Evaluation haben sich auch Fälle aufgetan, in denen Indikatoren mehrfach greifen würden, wie zum Beispiel bei: „*It would be nice to be able to* $\boldsymbol{select}_{VP}$ $\boldsymbol{multiple}_{JJ}$ $\boldsymbol{files}_{NNS}$ $\boldsymbol{and\ folders}_{NNS/NP}$ $\boldsymbol{in}_{PP}$ *the folder tree, and to be able to drag these selected files to the playlist*“ (Koordinations- und Anbindungsambiguität).

Indikatoren lexikalischer Ambiguität

Auch der Indikator für lexikalische Ambiguität ist einer weiteren Evaluation zu unterziehen. Denn wird die gesamte Anforderungsbeschreibung betrachtet, liegt in allen evaluierten Anforderungsbeschreibungen lexikalische Ambiguität vor, da immer mindestens ein *Token* (in einer berücksichtigten semantischen Funktion) vorliegt, das potentiell ambig ist. Allerdings wird bisher nicht betrachtet, wie oft auf dem Weg zu dieser Entscheidung Fehler passieren. Wie viele *Token* können beispielsweise hinsichtlich potentieller Ambiguität nicht bewertet werden, da die zugrundeliegende Ressource (WordNet) keinen entsprechenden Eintrag enthält? Zur weiteren Erläuterung der Problematik ist der Beispielsatz „*I want to unsubscribe from html newsletters with one click*“ in Abbildung 8.4 dargestellt.

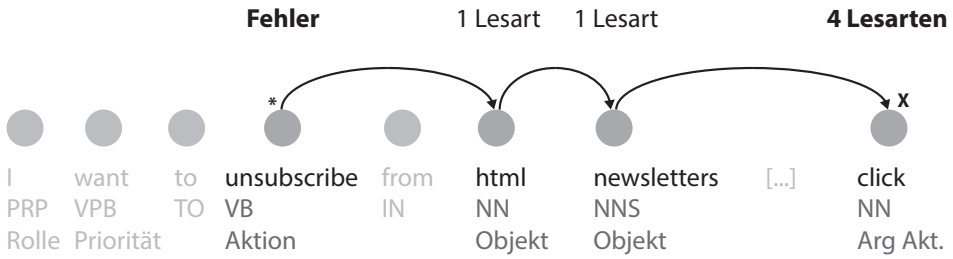


Abbildung 8.4: Fehler bei der tokenbasierten Indikatorbestimmung (WSD)

Der Indikator für lexikalische Ambguität markiert den dargestellten Satz richtigerweise als ambig. In dieser Entscheidung werden die ersten drei *Token* „*I*“, „*want*“ und „*to*“ ignoriert, da sie entweder keiner oder einer nicht berücksichtigten semantischen Kategorie angehören (s. Abschnitt 5.3.2.1). Das vierte untersuchte *Token* „*unsubscribe*“ führt zu einem Fehler, da es in WordNet nicht gefunden werden kann. Es bleibt an dieser Stelle demnach unsicher, ob lexikalische Ambiguität vorliegt und ob der Indikator zu aktivieren ist. Die folgenden beiden relevanten *Token* „*html*“ und „*newsletters*“ haben jeweils nur eine Lesart in WordNet und sind daher als nicht ambig einzustufen. Erst das zehnte *Token* („*click*“) weist Ambiguität auf und ist in WordNet vertreten, sodass der Indikator greift. Hierbei handelt es sich nicht um ein Problem der Genauigkeit, da es sehr unwahrscheinlich (wenn auch nicht ausgschlossen) ist, dass alle ambigen *Token* einer Anforderungsbeschreibung nicht von WordNet abgedeckt werden und der Indikator daher fälschlicherweise deaktiviert bleiben würde. Jedoch ist es aus Performanzgründen problematisch, da in diesem Fall drei Anfragen gestellt wurden, die eigentlich, da „*unsubscribe*“ als Ambiguitätshinweis genügt hätte[155], nicht erforderlich gewesen wären.

Diese Art von Fehler kommt bei der Verarbeitung der 400 Anforderungsbeschreibungen aus Abschnitt 8.2.2 insgesamt 47 Mal (12%) vor und erzeugt 55 zusätzliche

[155] *Unsubscribe* kann beispielweise als „sich abmelden“ oder „abbestellen“ gelesen werden.

Anfragen. Diese Angabe zeigt allerdings nur die Fälle auf, die Einfluss auf die Indikatorbestimmung haben. Werden alle *Token* der 400 Anforderungsbeschreibungen betrachtet, die keine Stoppwörter sind (13.353 *Token*), so sind von diesen 82% in WordNet abgebildet. Hiervon wiederum sind 87% ambig. Das bedeutet, dass potentiell 2.366 *Token* (18%) im Evaluationskorpus die Performanz dieses Indikators schädigen können, da sie nicht von WordNet abgebildet werden und zusätzliche Anfragen begründen. Unter genauerer Betrachtung ist dabei festzustellen, dass sich diese Menge an *Token* vor allem aus falsch geschriebenen Wörtern[156], Fachvokabular, Produktnamen (z. B. „Spotify“) und Dateiendungen (z. B. „xml“) zusammensetzt. Dieser Performanzaspekt wird in Abschnitt 8.3 auf Systemebene untersucht.

8.2.4.2 Strategieanwendung

F5

Die Strategieanwendung kann zu fehlerhaften Ergebnissen führen. Die Fehler reichen dabei von fehlenden Satzteilen, falsch klassifizierten Anforderungen bis hin zu unleserlichen Ergebnissen. Dabei können die Fehlerquellen vielfältig sein. Im Folgenden sind insbesondere die Fehler von Interesse, die sich durch eine Verkettung falscher Entscheidungen und fehlerhafter Informationen ergeben. Abbildung 8.5 zeigt beispielsweise ein fehlerhaftes Kompensationsergebnis (lückenhafte Koreferenzkette, keine Unvollständigkeitskompensation).

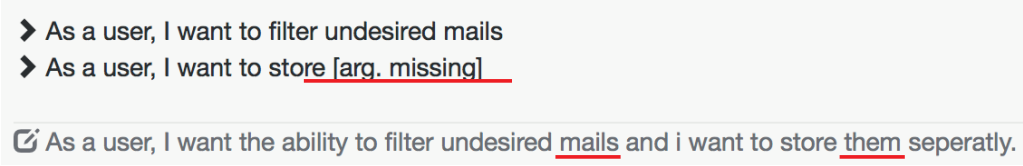


Abbildung 8.5: Fehlerhafte Kompensation: Argument wurde nicht zugeordnet

In diesem Beispiel wird korrekt erkannt, dass „*them*“ als Objekt nicht isoliert interpretiert werden kann, sodass es auf „*mails*” bezogen werden muss. Allerdings ist die Referenzierung misslungen, da das Personalpronomen „*i*“ fälschlicherweise (evtl. aufgrund der Schreibweise) seitens der Anforderungsextraktion als Objekt erkannt wurde, dem intern sowohl das POS-*Tag* NN als auch PRON zugeordnet wurde. Die Strategie musste in diesem Fall den Kompensationsvorgang abbrechen, da keine Entscheidung über das korrekte Objekt getroffen werden konnte. Im Folgenden werden weitere Fehlertypen aufgezeigt, die während der Evaluation auftraten.

Auswirkungen falsch oder nicht erkannter semantischer Kategorien

Fehler in semantischen Kategorien sind gravierend, da diese vielfältige Verwendung in den Indikatoren, Strategien und Methoden finden. Abbildung 8.6 zeigt die Verwendung der IE-Ergebnisse in der strategiebasierten Weiterverarbeitung und zeigt auch, ob diese Komponenten anfällig für falsche IE-Ergebnisse sind (schwarz markiert).

[156] Keine Rechtschreibkorrektur durchzuführen ist weiterhin sinnvoll, da eine Rechtschreibkorrektur zwar einen Abgleich mit WordNet potentiell ermöglicht, jedoch ggf. falsche *Token* (Ergebnis falscher Korrektur) abgleicht und damit insgesamt die Indikatorzuverlässigkeit mindert.

Die **Vagheitserkennung** bezieht die erkannten semantischen Kategorien lediglich in der Form ein, als dass eine Filterung der zu untersuchenden *Token* erfolgen kann. Demnach werden nur die *Token* auf Vagheit geprüft, die einer semantischen Kategorie zugeordnet werden konnten. Aufgrund der Performanz der Vagheitserkennung ist eine solche Filterung aber nicht notwendig, sodass in der derzeitigen Umsetzung kein negativer Effekt durch falsche oder nicht erkannte semantische Kategorien auftritt.

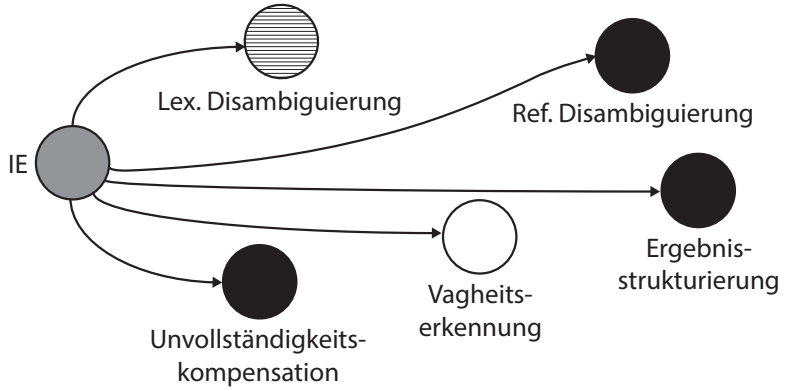


Abbildung 8.6: Komponenten mit Einbezug der IE-Ergebnisse.
• = Fehleranfällig; ∘ = Nicht fehleranfällig; ⊖ = Begrenzt fehleranfällig

Eine vergleichbare Filterung findet bei der **lexikalischen Disambiguierung** statt. Hier trägt das Vorgehen allerdings wesentlich zur Performanz des Verfahrens bei, da nur *Token*, die einer semantischen Kategorie angehören, mit Zusatzinformationen aus BabelNet angereichert werden. Die Auswirkungen möglicher fehlerhafter semantischer Kategorien sind dabei zu vernachlässigen, da die *Token* in jedem Fall disambiguiert werden (BabelID) und lediglich Zusatzinformationen fehlen (z. B. Domäne). Darüber hinaus werden die semantischen Kategorien auch als Hinweis auf Mehrwortlexeme herangezogen, was allerdings ebenfalls nicht zu Fehlern in der Disambiguierung sondern maximal zu Performanzeinbußen führen kann, weshalb die lexikalische Disambiguierung in Abbildung 8.6 schraffiert dargestellt ist.

Ebenfalls der Filterung durch die semantischen Kategorien unterliegen Prädikate, die im Rahmen der **Unvollständigkeitskompensation** auf Vollständigkeit zu prüfen sind. Hierbei werden nur diejenigen Prädikate geprüft, die in der semantischen Kategorie „Aktion“ vorliegen. Ein fälschlicherweise nicht als Aktion erkanntes Prädikat wird daher nicht auf Vollständigkeit geprüft und verbleibt im Zweifel unvollständig. Umgekehrt werden (vermeintliche) Prädikate bzw. Aktionen kompensiert, die nicht verarbeitet werden müssten (vgl. Abbildung 8.7). Die Auswirkung von Fehlern in der Anforderungsextraktion ist demnach höher als beispielsweise bei der Vagheitserkennung, auch wenn es prinzipiell das gleiche Vorgehen ist.

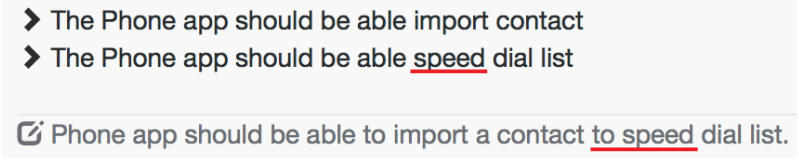


Abbildung 8.7: Fehlerhaftes Gesamtergebnis

Abbildung 8.7 zeigt die FA „*[Phone app]*$_{Komponente}$ *should be able to import*$_{Aktion}$ *a contact to speed*$_{Aktion}$ *dial list*“. Fälschlicherweise wurde „*speed*“ nicht als Teil des Objekts („*speed dial list*“), sondern als Aktion („*to speed*“) erkannt. Aus diesem Grund wurde eine weitere FA extrahiert, welcher „*dial list*“ als Objekt zugeordnet wurde. In der Funktion der Aktion ist „*speed*“ Gegenstand der Unvollständigkeitskompensation und wäre entsprechend weiterverarbeitet worden, ohne das dies notwendig gewesen wäre. Allerdings greift in diesem Fall die Unvollständigkeitskompensation nicht, da der entsprechende Indikator (korrekterweise) keine Unvollständigkeit feststellen konnte. Dieses Beispiel zeigt auf, dass ein solcher Fehler nicht immer einfach zu erkennen ist. Auf Grund der falschen Zuordnung der semantischen Kategorie wurde aus dem Nomen ein Verb, dem in der Funktion der Aktion alle Argumente zugeordnet werden konnten („*phone app*“, „*dial list*“) und das auch durch die lexikalische Disambiguierung als korrekte Lesart bestätigt wurde.

Für die **referentielle Disambiguierung** sind die semantischen Kategorien von Bedeutung, da sie zur Bildung von Koreferenzketten herangezogen werden können. Liegen allerdings Fehler in der Kategoriezuweisung vor, können falsche Koreferenzketten erstellt bzw. bestehende Ketten fälschlicherweise erweitert werden. Dies wiederum kann auch die Kompensation von Unvollständigkeit betreffen, wenn Kompensationsanfragen durch die referentielle Disambiguierung falsch modifiziert werden.

Gravierenden Einfluss auf die Ergebnisqualität im Sinne der **Ergebnisstrukturierung** in kontrollierter Sprache (Satzstruktur) hingegen haben die semantischen Kategorien. Da die erkannten Kategorien (z. B. Rolle) entsprechend der definierten Satzmuster (s. Abschnitt 5.5.7) auf der Benutzeroberfläche positioniert werden, führen falsch erkannte Kategorien zu unleserlichen oder unvollständigen Sätzen und damit mindestens zu einem schlechten Ergebnis (vgl. Abbildung 8.8).

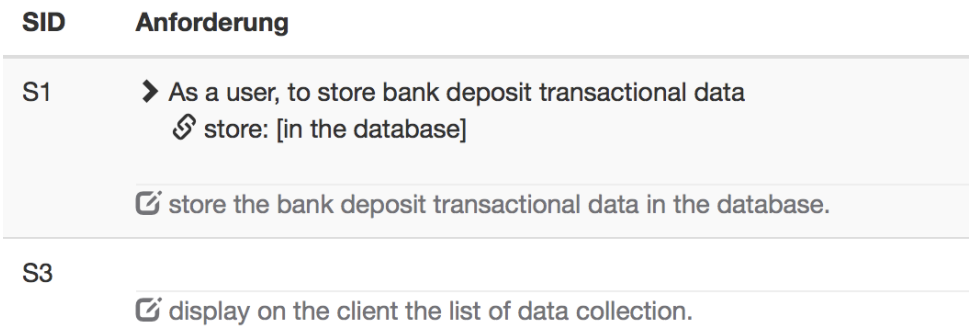


Abbildung 8.8: Fehlerhafte Ergebnisdarstellung in kontrollierter Sprache

Abbildung 8.8 zeigt gleich zwei fehlerhafte Ergebnisse. In der ersten FA (S1) ist in der Anforderungsbeschreibung keine Priorität (z. B. „*want*“) und keine Rolle (z. B. „*I*“) angegeben, die somit beide kompensiert werden müssten, was hier fehlschlägt. Hingegen wird das Satzmuster (vermeintlich) korrekt gewählt und durch „*As a user*“ ergänzt. In der zweiten FA (S3) wird die Ausgabe in kontrollierter Sprache abgebrochen, da keine Aktion erkannt werden konnte. Dies lässt sich auf den Umstand zurückführen, dass alle Komponenten „*display*“ fälschlicherweise als Nomen bzw. in der semantischen Kategorie „Komponente“ erkennen, statt es als Verb bzw. Aktion zu führen.

Fehler in der Unvollständigkeitskompensation

Fehler in der Unvollständigkeitskompensation können insbesondere durch ein schlechtes SRL-Ergebnis (z. B. falsch erkannte Argumente eines Prädikats) und fehlerhafte syntaktische Korrekturmaßnahmen entstehen, die wiederum zu fehlerhaften Kompensationsanfragen und -ergebnissen führen.

Ein Beispiel für ein fehlerhaftes SRL-Ergebnis ist: „*Please include*$_{Aktion}$ *[the file timestamp of the show in the log file]*$_{Argument}$“. Hier wird die PP an die NP angebunden, wodurch das Argument einen größeren Informationsumfang als vorgesehen hat. Diese Fehlerart kann durch die Hinzunahme der syntaktischen Disambiguierung korrigiert werden. Es kann allerdings auch der Fall auftreten, dass die Expertenkomponente für syntaktische Disambiguierung die Ergebnisse verschlechtert. Dies lässt sich an folgendem Beispielsatz illustrieren: „*For sending simple posts it's OK, but if* i_{Rolle} *want to* $send_{Aktion}$ *[some attachments (jpg, pdf....)]*$_{Objekt}$. *It would be nice to have that feature*“. Die SRL-Komponente erkennt das Prädikat (*send.01*) sowie das Argument A1 („*i*“) korrekt (vgl. Abbildung 8.9). Darüber hinaus wird Argument A1 partiell erkannt („*some attachments*“).

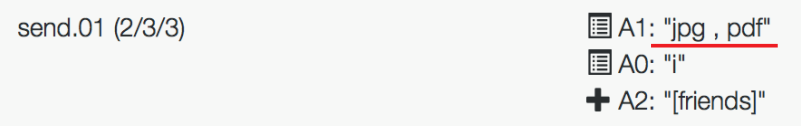


Abbildung 8.9: Fehlerhafte Kompensation: Argument nicht korrekt erkannt

Das Ergebnis in Abbildung 8.9 sieht allerdings insofern anders als das SRL-Ergebnis aus, als dass Argument A1 nun „*jpg, pdf*“ enthält. Dies ist darauf zurückzuführen, dass die Expertenkomponente für syntaktische Disambiguierung korrigierend eingreift und damit leider das Ergebnis verschlechtert („*jpg, pdf*“ statt „*some attachments*“). Hier zeigt sich, dass in den Strategien weitere Regeln zur Qualitätssicherung notwendig sind, da das Aufrufen der Expertenkomponente in diesem Fall nicht zwangsläufig notwendig gewesen wäre, wenngleich beispielhafte Dateiformate die vage Angabe „*some attachments*“ genauer spezifizieren. Als positiv zu bezeichnen ist, dass die Instanziierung des Arguments A2 („*friends*“) erfolgte.

Ein weiteres Beispiel ist der Satz „$Edit_{NN}$ $section_{NN}$ $needs_{VBZ}$ *to be orginized like add section to show how the breakers look. It would be great if it was possible to change the size of characters*“. Während die SRL-Komponente „*Edit section*“ als NP identifiziert, erkennt die syntaktische Disambiguierung „*Edit*“ als Verb und „*needs*“ als Nomen, sodass das korrekt erkannte Argument „$Edit_{NN}$ $section_{NN}$“ fälschlicherweise zu „$section_{NN}$ $needs_{NNS}$“ korrigiert wird.

Fehler durch fehlerhafte Koreferenzketten

Die referentielle Disambiguierung wird auch dazu genutzt, die Ergebnisse der Anforderungsextraktion zu verbessern (s. Abschnitt 5.2). Beispielsweise werden Personalpronomen in der semantischen Kategorie Objekt mit ihren Referenten verknüpft. Treten hierbei Fehler auf, können falsche Zusammenhänge hergestellt werden.

Die folgende FA führt gleich zu mehreren Verarbeitungsfehlern: „*[Because these data ara rather confidential]$_{Bedingung}$, it should be very important to encript$_{Aktion}$ them$_{Objekt}$ before saving*“. Als Objekt dieser FA wird das Personalpronomen „*them*“ erkannt, welches zusammen mit „*it*“ und „*these data*“ eine Koreferenzkette bildet (vgl. Abbildung 8.10). Die Koreferenzkette ist aber insofern fehlerhaft, als dass „*it*“ fälschlicherweise Bestandteil der Kette ist. Das Verknüpfen von „*it*“ mit „*these data*“ ist demnach falsch. Zur Qualitätsverbesserung werden semantische Kategorien als weiteres Indiz hinzugezogen, was in diesem Fall jedoch ebenfalls zu Fehlern führt. Das Objekt „*them*“ referenziert auf „*these data*“ im ersten Halbsatz, der wiederum allerdings seitens der Anforderungsklassifikation als Kategorie „Bedingung“ annotiert wird und somit einer anderen semantischen Kategorie angehört als das Personalpronomen. An dieser Stelle kann „*them*“ nicht korrekt ersetzt werden.

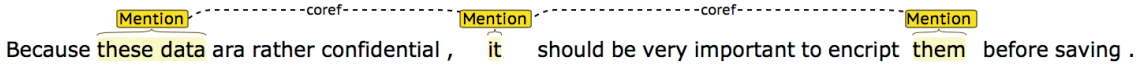


Abbildung 8.10: Beispiel für eine fehlerhafte Koreferenzkette

Die referentielle Disambiguierung kann darüber hinaus auch auf die Kompensationsanfragen der Unvollständigkeitskompensation Einfluss nehmen. Stellt beispielsweise ein Personalpronomen (z. B. „*them*“) in einer FA das alleinige Objekt dar, erlaubt dies keine zielführende Kompensation. Das Ersetzen des Objekts durch ein zuvor annotiertes Objekt (kein Personalpronomen) schafft hier Abhilfe. Allerdings kann eine fehlerhafte Ersetzung auch genau das Gegenteil bewirken: Wird beispielsweise in der FA „*I want to [create emails (large files)]$_{Objekt}$. I must be able to send them$_{Objekt}$*“ das Personalpronomen fälschlicherweise durch „*files*“ statt durch „*emails*“ ersetzt (z. B. aufgrund von Ambiguität), würde das Ergebnis der Unvollständigkeitskompensation insofern beeinflusst, als dass ein wichtiger Suchbestandteil (Argument) verfälscht wird.

Fehler durch fehlerhaftes Preprocessing

Sowohl die Indikatoren als auch die Strategien basieren auf den Ergebnissen, die durch das *Preprocessing* (s. Abschnitt 5.5.2) erzeugt werden. Hierbei können Fehler auftreten, die, wie die Evaluation zeigt, zu fehlerhaften Ergebnissen führen können. Neben der Anforderungsextraktion, die zuvor aufgrund ihrer Relevanz für das gesamte Softwaresystem für sich betrachtet wurde, sind vor allem noch die Anforderungsklassifikation (Bestandteil von REaCT) und die Satzvereinfachung (Bestandteil von *Stanford CoreNLP*) zu betrachten.

Die regelbasierte Satzvereinfachung kann die Ergebnisqualität direkt negativ beeinflussen, indem Sätze fälschlicherweise getrennt werden. Dies passiert, wenn die zugrundeliegende Satzstruktur zuvor falsch erkannt wurde. Beispielsweise führt die FA „*I need a software which can be only play mp3 files on Android but the user not allow to copy or send with bluetooth*“ zu einer falschen Satzvereinfachung, da drei Teilsätze erkannt werden, die in dieser Form von den zugrundeliegenden Regeln nicht abgedeckt werden, sodass eine ungeeignete Regel greift.

Wie Abbildung 8.11 darüber hinaus zeigt, kann die Satzvereinfachung die Verarbeitung auch indirekt negativ beeinflussen. Die beispielhafte Anforderungsbe-

schreibung ist sprachlich von geringer Qualität, prinzipiell aber verarbeitbar. Die Satzvereinfachung erkennt, dass eine komplexe Satzkonstruktion vorliegt und trennt den komplexen Satz in zwei einfache Sätze auf.

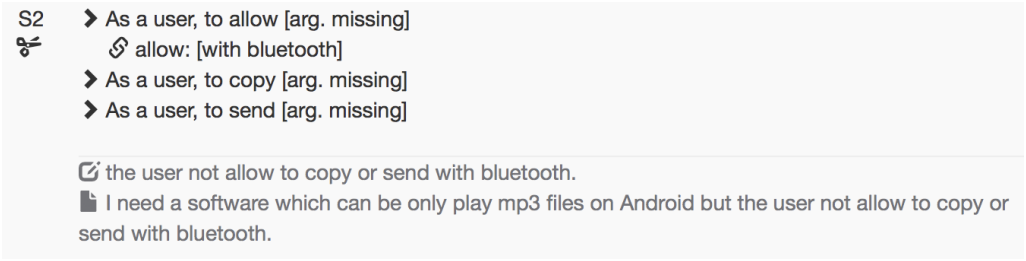


Abbildung 8.11: Beispiel für fehlerhafte Satzvereinfachung und deren Folgefehler

In Abbildung 8.11 ist nun eine Verkettung von Fehlentscheidungen erkennbar. Durch die Satzvereinfachung hat die Anforderungsklassifikation den ersten Satzteil als *Off-Topic* klassifiziert und gelöscht. Der zweite Satzteil enthält keine wesentlichen semantischen Informationen, was wiederum die Anforderungsextraktion erschwert. Es fehlen semantische Kategorien (z. B. Objekt „*mp3 files*“), die nur im ersten Satzteil vorliegen. Das verschlechtert die Ergebnisstrukturierung.

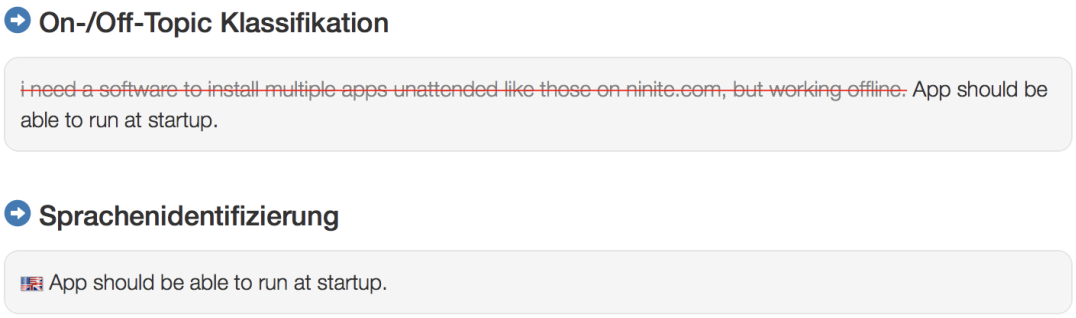


Abbildung 8.12: Beispiel für fehlerhafte Anforderungsklassifikation

Die Anforderungsklassifikation kann durch eine falsche Klassifikation das Gesamtergebnis erheblich schädigen, da Sätze der Weiterverarbeitung vorenthalten werden. Wie in Abbildung 8.12 ersichtlich wird, verarbeitet die Folgekomponente (Sprachendientifizierung) nur noch eine der beiden FA aufgrund der falschen Klassifikation.

8.3 Evaluation der Systemperformanz

Hohe Systemperformanz gilt als etabliertes Qualitätsmerkmal von Softwaresystemen und ist im Sinne der Systemakzeptanz vor allem dort sicherzustellen, wo Nutzerinteraktion stattfindet (Knott, 2016, S. 2, 69 ff.). Aus diesem Grund wird im Folgenden eine Analyse der Gesamtlaufzeiten vorgenommen, wobei zwischen Gesamtlaufzeit des Softwaresystems (s. Abschnitt 8.3.2) und der Gesamtlaufzeit von Verarbeitungskomponenten (s. Abschnitt 8.3.3) sowie von Strategien (s. Abschnitt 8.3.5) unterschieden wird. Darüber hinaus wird das *Caching*-Verfahren der lexikalischen Disambiguierung, als ein Beispiel möglicher Performanzsteigerung, näher betrachtet (s. Abschnitt 8.3.4).

8.3.1 Evaluationsprotokoll

Die Evaluation widmet sich dem in Kapitel 5 konzipierten und in Kapitel 7 implementierten, Softwaresystem, wobei die Messung und Analyse der Performanz einzelner Komponenten sowie des Gesamtsystems **Evaluationsgegenstand** ist. Grundsätzlich gilt Performanz als ein etabliertes Qualitätsmerkmal von Softwaresystemen (Knott, 2016, S. 2, 69 ff.), hat jedoch besonderen Stellenwert im OTF-*Computing*. Diesbezüglich führt beispielsweise Vogel et al. (2009, S. 114 f.) an, dass eine hohe Performanz erheblich zur Anwenderzufriedenheit beiträgt (s. Abschnitt 7.4.1.1).

Nach Knott (2016, S. 69 ff.) bezieht sich die Performanzevaluation bei *Client-Server*-basierten Softwaresystemen, wie es in dieser Arbeit beschrieben wird, insbesondere auf die Performanz der Nutzerschnittstelle (1) und die des *Servers* (2). In dieser Arbeit liegt der Fokus auf dem *Server*, da es sich bei der Nutzerschnittstelle um eine reine Präsentationsschicht und nicht um eine Anwendungsschicht handelt (s. Abschnitt 7.3). Daneben ist laut Knott (2016, S. 69 ff.) noch die Performanz des zugrundeliegenden Netzwerks (3) zu beachten, auf welches aber kaum Einfluss genommen werden kann. Im Falle des vorliegenden Systems betrifft externe Netzwerkkommunikation nur die Komponente der lexikalischen Disambiguierung (s. Abschnitt 7.1) und wird gesondert in Abschnitt 8.3.4 hinsichtlich Netzwerkschwankungen und Performanz evaluiert. Nach Knott (2016, S. 69 ff.) können daher als **Evaluationszweck** folgende Punkte ausgemacht werden:

- Identifikation von Leistungsengpässen auf Seiten des *Servers*
- Steigerung der Nutzerzufriedenheit durch Performanzverbesserung

Im Folgenden bezieht sich Performanz in erster Linie auf die Gesamtlaufzeit bzw. auf den gesamten Ausführungs- und Verarbeitungszeitraum, ausgehend vom Verarbeitungsstart durch den Endanwender bis zur abgeschlossenen Ergebnisstrukturierung (vgl. Definition 8.3.3). Hierbei wird weiter unterschieden[157] zwischen der Initialisierungszeit und der Ausführungszeit einer Komponente oder des Systems. Während die Initialisierungszeit (vgl. Definition 8.3.1) den Ladevorgang des Systems bzw. einer Systemkomponente beschreibt, bezieht sich die Ausführungszeit einzig auf den Verarbeitungsprozess (vgl. Definition 8.3.2).

[157] In Anlehnung an den Microsoft API- und Referenzkatalog zum Thema „Anpassen von Timeoutwerten für Prozesse“. Siehe: `https://msdn.microsoft.com/library/bb750236` (Stand: 23.02.17).

Definition 8.3.1 (Initialisierungszeit)
Die Initialisierungszeit ist die Zeit (in Millisekunden), die eine Komponente oder ein System für die vollumfängliche Initialisierung in Anspruch nimmt.

Definition 8.3.2 (Ausführungszeit)
Die Ausführungszeit ist die Zeit (in Millisekunden), die eine Komponente oder ein System für die vollumfängliche Verarbeitung in Anspruch nimmt.

Definition 8.3.3 (Gesamtlaufzeit)
Die Gesamtlaufzeit ist die Summe der Initialisierungszeit und Ausführungszeit (in Millisekunden), bezogen auf das Gesamtsystem oder einzelne Komponenten.

Die Erkenntnisse, die aus der Evaluation hinsichtlich der Performanz einzelner Komponenten, Indikatoren und Strategien sowie des Gesamtsystems gezogen werden, bilden die Grundlage für die Weiterentwicklung und können zum Beispiel die Auswechslung von Komponenten begründen, sollte keine Abhilfe für festgestellte Performanzprobleme existieren. Um dies jedoch zu erreichen, müssen zu dem genannten Evaluationszweck geeignete **Evaluationsfragen** (`Q`) formuliert werden.

- `Q1`: Wie entwickelt sich die Ausführungszeit unter steigender Last?
- `Q2`: In welchem Wertebereich schwankt die Initialisierungszeit?
- `Q3`: Inwiefern beeinflussen nebensächliche Angaben die Systemlaufzeit?
- `Q4`: Welche Verarbeitungskomponenten haben den größten Laufzeitanteil?
- `Q5`: In welchem Intervall schwankt die Komponenteninitialisierungszeit?
- `Q6`: Welchen Anteil haben Komponentenbestandteile an der Laufzeit?
- `Q7`: Welcher Performanzgewinn kann durch die Anwendung des WSD-*Cachings* in der lexikalischen Disambiguierung erreicht werden?
- `Q8`: Ist eine Grenze auszumachen, ab der keine weiteren *Token* bzw. Lesarten in den WSD-*Cache* aufgenommen werden und somit alle Anfragen per WSD-*Cache* beantwortet werden können?
- `Q9`: Ist die Anfragenverteilung im WSD-*Cache* innerhalb einer Domäne anders ausgeprägt, als bei domänenübergreifenden Anforderungsbeschreibungen?
- `Q10`: Wie entwickelt sich die jeweilige Strategielaufzeit unter steigender Last?

Hinsichtlich der Durchführung der Evaluation müssen folgend geeignete **Methoden** herangezogen werden. Da die Systemperformanz im Fokus steht, ist ein automatisiertes Evaluationsvorgehen notwendig, dass „Verarbeitungsgeschwindigkeit und Antwortzeit im Hinblick auf steigende Last [misst]“ (Schulz, 2012). Hierfür wurde der sogenannte *Evaluator* entwickelt, der automatisiert Benutzereingaben tätigen und strategiebasierte Verarbeitungsprozesse starten kann.

Zur Durchführung bedarf es auch eines **Evaluationskorpuses**, welches Anforderungsbeschreibungen enthält, die an das Softwaresystem übermittelt werden können. Hierzu wird auf die in Abschnitt 6.1 beschriebene Ressource zurückgegriffen, die gleichermaßen FA (*On-Topic*) sowie Nebensächliches (*Off-Topic*) enthält. Wie Reisner (2011) allerdings mit der Forderung nach „realistischen Testdaten“ anmerkt, ist nicht nur auf die Auswahl der Datenbasis zu achten, sondern auch auf deren Anwendung auf das System. So wird empfohlen, „die Testdaten in zufälliger Reihenfolge [zu] verwenden“ (Reisner, 2011), da sonst die Gefahr einer Verfälschung der Evaluation besteht – zum Beispiel durch wiederholte Datenabfrage aus einer Domäne oder einer stets ähnlichen Anforderungsbeschreibungs- bzw. Satzlänge. In dieser Arbeit betrifft dies auch das Verhältnis von FA und nebensächlichen Angaben innerhalb einer Anforderungsbeschreibung.

Abbildung 8.13 stellt das Vorgehen mittels *Evaluator* dar. Ausgehend vom Evaluationskorpus, in dem Sätze hinsichtlich ihrer Zugehörigkeit zu FA oder nebensächlichen Angaben markiert sind, stellt die Evaluationsanwendung zufällig Anforderungsbeschreibungen zusammen (unter gegebenen Rahmenparametern) und übermittelt sie iterativ an das Softwaresystem.

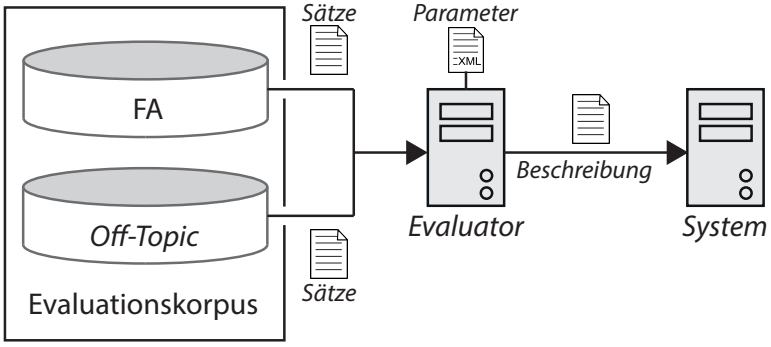


Abbildung 8.13: Generierung der Testdaten mittels *Evaluator*

Dieses Vorgehen hat entscheidende Vorteile: Zum einen wird der Forderung von Reisner (2011) nachgekommen, indem Anforderungsbeschreibungen in zufälliger Reihenfolge ausgewählt sowie kombiniert werden und somit kein Datenmuster die Ergebnisse verfälscht. Zum anderen erlauben es die Konfigurationsparameter von *Evaluator*, genau zu definieren, wie die Merkmale der zu erzeugenden Anforderungsbeschreibungen für Testzwecke ausgeprägt sein sollen (z. B. Mindestlänge, Anzahl nebensächlicher Angaben).

8.3.2 Laufzeitanalysen des Gesamtsystems

In diesem Abschnitt geht es um die Beantwortung der Evaluationsfragen `Q1`, `Q2` und `Q3`, die alle auf das Gesamtsystem als Evaluationsgegenstand abzielen. Um die zur Beantwortung erforderlichen Laufzeiten zu erheben, werden 500 Anforderungsbeschreibungen[158] auf Basis des Evaluationskorpus zufällig generiert, die jeweils an das System

[158] Jede Anforderungsbeschreibung besteht aus 1–5 Sätzen, wobei die konfigurierte Wahrscheinlichkeit, dass ein Satz *Off-Topic* ist, bei 20% liegt. Insgesamt 1500 zufällig gewählte FA.

übertragen werden, sodass insgesamt 500 Anfragen seitens des Systems sequenziell zu bearbeiten sind. Damit sichergestellt werden kann, dass alle Verarbeitungskomponenten ausgeführt werden, ist die *Complete*-Strategie zur Messung der Laufzeiten voreingestellt[159]. Tabelle 8.4 zeigt die durchschnittlichen Ausführungszeiten über unterschiedliche Anforderungsbeschreibungslängen (Satzumfang). **Q1**

Es fällt auf, dass die durchschnittliche Ausführungszeit abhängig vom Anforderungsbeschreibungsumfang zunimmt, was angesichts des erforderlichen Mehraufwands in der Verarbeitung nicht weiter überrascht. Vielmehr ist hervorzuheben, dass die Ausführungszeit merklich schwankt. Dies ist beispielsweise bei der Anforderungsbeschreibung mit einem Umfang von fünf Sätzen (1.105 *ms*) im Vergleich zur Verarbeitung von drei Sätzen (1.566 *ms*) erkennbar.

Anzahl Sätze	**1**	**2**	**3**	**4**	**5**
∅ [*Token*]	11	19	34	49	65
Min [*ms*]	189	449	1.524	1.566	1.105
Max [*ms*]	16.184	29.224	112.346	26.674	26.655
∅ [*ms*]	4.010	5.488	6.976	10.011	11.742

Tabelle 8.4: Durchschnittliche Ausführungszeiten des Softwaresystems unter Last

Hervorzuheben ist darüber hinaus ein Ausreißer[160] mit 112.346 *ms* bei drei Sätzen. Dieser ist aufgrund einer überdurchschnittlichen Verarbeitungszeit der Anforderungsklassifikation (107.263 *ms*) entstanden. Wird dieser Ausreißer entfernt, liegt das Maximum bei 17.121 *ms* und der Durchschnitt bei 5.912 *ms*. Dies ist nur ein weiteres Beispiel dafür, welchen Schwankungen die Ausführungszeit unterliegt und wie einzelne Verarbeitungskomponenten die Laufzeiten beeinflussen, was in Abschnitt 8.3.3 genauer evaluiert wird. Allerdings lässt sich bereits jetzt eine Zweiteilung der Laufzeiten hinsichtlich der Verarbeitungsschritte des *Preprocessings* und der Strategieanwendung vornehmen (vgl. Tabelle 8.5).

Anzahl Sätze	**1**	**2**	**3**	**4**	**5**
∅ ***Preprocessing*** [*ms*]	1.028	2.072	3.058	2.230	2.498
∅ ***Complete*-Strategie** [*ms*]	2.981	3.416	3.917	7.781	9.243

Tabelle 8.5: Durchschnittliche Ausführungszeiten

Im Durchschnitt entfallen 30% der Ausführungszeit auf das *Preprocessing*, während die übrigen 70% der Erkennung- und Kompensation innerhalb der Strategieanwendung zuzuschreiben sind. Hier stellt sich die Frage, ob gegebenenfalls einzelne Verarbeitungskomponenten für diese Verteilung verantwortlich sind oder ob tatsächlich die Aufgabenkomplexität in der Erkennung- und Kompensation für die höhere Laufzeit verantwortlich ist. Dieser Frage wird in Abschnitt 8.3.3 nachgegangen. **Q2**

Neben der Ausführungszeit ist auch die Initialisierungszeit des Gesamtsystems von Interesse. Wobei das an dieser Stelle nicht die Initialisierungzeit der Verarbeitungskomponenten (s. hierzu Abschnitt 8.3.3) oder des *Webservers*, sondern ausschließlich

[159] Eine Evaluation unter Anwendung unterschiedlicher Strategien findet in Abschnitt 8.3.5 statt.
[160] Eindeutige Evaluationskennung: B7E33EFD042791BBD3242922E28CFAC7_04032017120129.

die Initialisierung des Kernsystems betreffen soll. Das umfasst beispielsweise den Verbindungsaufbau[161] zu allen Verarbeitungskomponenten oder das Initialisieren des Strategie-*Controllers* (s. Abschnitt 5.2). Da dieser Initialisierungsschritt nur zum Systemstart durchgeführt wird, wird zur Evaluation das Softwaresystem wiederholt neu gestartet und die jeweiligen Initialisierungszeiten gemessen. Bei insgesamt 120 Messungen[162] kann dabei eine durchschnittliche Initialisierungszeit von 4.386 *ms* ermittelt werden, wobei das Minimum bei 2.167 *ms* und das Maximum bei 19.790 *ms* liegt. Die Initialisierungszeit ist unabhängig vom Anforderungsbeschreibungsumfang
Q3 sowie vom Anteil nebensächlicher Angaben.

Bezüglich nebensächlicher Angaben ist bisher die Frage offengeblieben, wie diese sich auf die Ausführungszeit auswirken. Bekannt ist, dass *Off-Topic*-Angaben für die Performanz schädlich sind und gefiltert werden müssen, da ihre Verarbeitung Zeit in Anspruch nimmt aber keinen Mehrwert schafft. Dabei ist bisher nicht thematisiert worden, inwiefern auch das Filtern, also die Anforderungsklassifikation mittels REaCT (Dollmann und Geierhos, 2016) den Verarbeitungsprozess verlangsamt. Eine Überlegung hierzu ist, dass nebensächliche Angaben erst die Notwendigkeit des Filterns begründen und daher die Verarbeitungszeit der Anforderungsklassifikation vollumfänglich als *Schaden* geltend zu machen ist. Angesichts der Tatsache, dass die Klassifikation vergleichsweise viel Verarbeitungszeit bedarf, wäre der Einfluss von nebensächlichen Angaben auf die Systemlaufzeit groß (vgl. Tabelle 8.6). Jedoch kann auch argumentiert werden, dass die Anforderungsklassifikation elementarer Bestandteil eines maschinellen Textanalysesystems ist. In diesem Fall wäre demnach nur die Verarbeitungszeit durch die nebensächlichen Angaben verschuldet, die durch ihre zusätzliche Klassifikation entstehen. Diesbezüglich ist mit Blick auf die untersuchten Anforderungsbeschreibungen anzumerken, dass die Klassifikation von *Off-* und *On-Topic* die gleiche Verarbeitungszeit in Anspruch nimmt und auch die Länge der Sätze nur marginale Auswirkung auf die Laufzeit hat. Handelt es sich demnach bei einem von zwei Sätzen um eine nebensächliche Angabe, so sind 50% der Verarbeitungszeit als *Schaden* auszumachen. Im Vergleich zu der eingesparten Verarbeitungszeit bei den Folgekomponenten dürfte dies zu vernachlässigen sein.

Aufbauend auf den bisherigen Erkenntnissen wird im Folgenden die Laufzeitanalyse der Verarbeitungskomponenten vorgenommen.

8.3.3 Laufzeitanalyse der Verarbeitungskomponenten

Dieser Abschnitt gibt Antworten auf die Fragen `Q4`, `Q5` und `Q6` aus Abschnitt 8.3.1. Im Fokus stehen dabei die Laufzeiten der Verarbeitungskomponenten und ausgewählter Bestandteile (z. B. Verfahren). Die durchschnittlichen Ausführungs- und Initialisierungszeiten, die ebenfalls auf der in Abschnitt 8.3.2 durchgeführten Evaluation
Q4 beruhen, stellt Tabelle 8.6 dar.

Hinsichtlich der Ausführungszeiten der Komponenten fallen deutliche Unterschiede auf. So ist im Bereich des *Preprocessings* die Satzendeerkennung auffallend zeitintensiv (durchschnittlich 1.120 *ms*). Dies ist allerdings mit einer Designentscheidung zu begründen, die in dieser Arbeit getroffen wurde. So handelt es sich hierbei um eine

[161] Hiermit ist die Herstellung der HTTPS-Verbindungen zu ausgelagerten Komponenten gemeint.
[162] 120 Messungen verteilt über 12 Stunden (10 Messungen / Stunde).

Preprocessing-Komponente, die auch Informationen zur referentiellen und syntaktischen Disambiguierung beitragen kann (Synergieeffekt). Somit sind die gemessenen Zeiten nicht allein auf die Satzendeerkennung zurückzuführen. Da sie allerdings bei diesem ersten Schritt anfallen, sind sie auch hier zu messen und zu protokollieren. Daneben sind die Zeiten der Anforderungsklassifikation sowie der -extraktion zu sehen, beides Bestandteile des REaCT-*Tools*, die im Schnitt eine halbe Sekunde Ausführungszeit beanspruchen. Darüber hinaus existieren noch die Zeichennormalisierung und die Sprachenidentifizierung, die hinsichtlich der Ausführungszeiten von einer Millisekunde zu vernachlässigen sind. Zusammenfassend lässt sich feststellen, dass die Ausführungszeiten absolut akzeptabel erscheinen (30% der Gesamtausführungszeit).

Ein ähnliches Bild ergibt sich bei den Verarbeitungskomponenten im Bereich der Erkennung und Kompensation, wenngleich auch eine Komponente merklich hervorsticht: So ist beispielsweise die Vagheitserkennung im Durchschnitt bereits nach nur einer Millisekunde abgeschlossen, während die lexikalische Disambiguierung im Schnitt 5.204 *ms* zur Verarbeitung benötigt. Dies ist soweit nicht verwunderlich, da die Komponentenkomplexität bei der Disambiguierung wesentlich höher ist und darüber hinaus eine Kompensation stattfindet, während diese bei der Vagheitserkennung ausbleibt. Allerdings erscheint die lexikalische Disambiguierung auch im Vergleich zu anderen Kompensationskomponenten (insb. der Unvollständigkeitskompensation) bedeutend zeitintensiver und bedingt maßgeblich die Ausführungszeit der Erkennung und Kompensation.

	Verarbeitung						**Init.**
Anzahl Sätze:	**1**	**2**	**3**	**4**	**5**	∅	
Preprocessing							
Satzendeerkennung	657	1.443	1.297	1.077	1.126	1.120	1
Zeichennormalisierung	1	1	1	1	1	1	1
Sprachenidentifizierung	1	1	1	1	1	1	757–1.119
Anforderungsklassifikation	180	359	1.341	453	487	564	1
Anforderungsextraktion	187	266	415	697	877	489	1
Erkennung und Kompensation							
Lex. Disambiguierung	2.858	3.155	3.675	7.497	8.835	5.204	2–257
Ref. Disambiguierung	1	1	1	1	1	1	1
Syn. Disambiguierung	13	6	11	10	10	10	1
Unvollständigkeitskompen.	106	249	224	263	384	245	27–395
Vagheitserkennung	1	1	1	1	1	1	1–2

Tabelle 8.6: Durchschnittliche Laufzeiten der Komponenten [*ms*].
Jeweils auf Basis von 100 Anforderungsbeschreibungen

Wie ersichtlich wird, sind auch die Initialisierungszeiten sehr unterschiedlich. Auf der einen Seite existieren Komponenten mit einer sehr geringen Initialisierungszeit, wie beispielsweise die vollständig extern ausgelagerten Verarbeitungskomponenten (z. B. Satzendeerkennung), deren Initialisierung entfällt[163]. Auf der anderen Seite existieren Komponenten wie die Sprachenidentifizierung, die lokal ausgeführt werden und daher zu initialisieren sind (z. B. indem Klassifikationsmodelle geladen werden). **Q5**

[163]Entsprechender HTTPS-Verbindungsaufbau geschieht bereits im *Controller* (s. Abschnitt 8.3.2).

Werden alle Initialisierungszeiten der Verarbeitungskomponenten zusammen über den vollständigen Evaluationsverlauf betrachtet, so ergibt sich ein Intervall zwischen 1.100 *ms* und 1.800 *ms*, wobei der Durchschnitt bei 1.352 *ms* liegt. Wie auch bei den Ausführungszeiten gibt es auch hier Komponenten wie die Zeichennormalisierung sowie die Vagheitserkennung, die in den Zeiten nur marginal abweichen. Demgegenüber stehen Komponenten wie die Sprachenidentifizierung sowie die Un-
Q6 vollständigkeitskompensation, die stark schwanken.

Um der Frage nach dem Einfluss einzelner Komponentenbestanteile nachgehen zu können, werden ausgewählte Verarbeitungskomponenten hinsichtlich des Zeitaufwands einzelner Funktionsaufrufe untersucht. Hierbei liegt der Fokus auf den Komponenten mit dem größten Anteil an der Gesamtausführungszeit (Satzendeerkennung und Lexikalische Disambiguierung).

Die Satzendeerkennung setzt sich aus mehreren Unterfunktionen zusammen, von denen nur eine Funktion eine messbare Ausführungszeit erzeugt. So ist die Einbindung des *Stanford Parsers* für 99% der Ausführungszeit verantwortlich, während alle Nachverarbeitungsschritte hinsichtlich der Laufzeiten zu vernachlässigen sind.

Eine vergleichbare Situation herrscht bei der Verarbeitungskomponente zur lexikalischen Disambiguierung. Diese lässt sich ebenfalls in mehrere Unterfunktionen aufteilen, von denen nur drei einen sichtbaren Anteil an der Ausführungszeit haben: (1) die Funktion zur Einbindung von Babelfy als Disambiguierungskomponente, (2) die Funktion zum Hinzuziehen der BabelNet-Ressource und (3) die Funktion des WSD-*Caches*. Während Babelfy (1) nur 5-6% der Ausführungszeit begründet, sind BabelNet (2) und der *Cache* (3) zusammen für über 93% der Ausführungszeit verantwortlich. Es zeigt sich somit, dass insgesamt zwei Verarbeitungskomponenten den Großteil der Gesamtausführungszeit der Komponenten begründen und dass hierbei von diesen Komponenten wiederum nur wenige Bestandteile involviert sind. Dies ermöglicht in der weiteren Arbeit eine zielgerichtetere Optimierung der Systemperformanz.

8.3.4 Entwicklung und Nutzen des WSD-Cachings

Wie in Abschnitt 7.3.2.2 beschrieben, kann bei der lexikalischen Disambiguierung eine Performanzsteigerung durch eine temporäre Zwischenspeicherung (*Cache*) auf Basis disambiguierter *Token* erreicht werden. Dies ist möglich, da die Anzahl an BabelNet-Abrufen, die notwendig sind um lexikalisches Wissen über die *Token* zu
Q7 akquirieren, minimiert werden kann. Um die Funktionsweise, das Verhalten und den Nutzen des *Cachings* besser zu verstehen, wird im Folgenden sowohl eine Evaluation in Hinblick auf die Zeitersparnis als auch auf die Ressourcenverteilung durchgeführt. Diesbezüglich wird zuerst auf Basis von 550 ambigen *Token* die Abrufzeit mit und ohne Anwendung des *Caching*-Verfahrens untersucht. Es wird deutlich, dass sich die Abrufzeiten durch das *Caching*-Verfahren erheblich verringern. Werden beispielsweise 90 ambige *Token* abgerufen, steigt die Abrufzeit ohne *Caching* auf 15,38 *s*, was im Gegensatz zur Abrufzeit mit *Caching* (0,360 *s*) die Gesamtlaufzeit erheblich negativ beeinflusst (s. Tabelle 8.7). Werden alle Durchläufe hinsichtlich der durchschnittlichen Abrufzeit pro *Token* betrachtet, ergibt sich ein Durchschnitt von 4,2 *ms* mit *Caching* und 183,1 *ms* ohne *Caching*.

	Alle *Token* [*ms*]		∅ pro *Token* [*ms*]	
# *Token*	Mit Caching	Ohne Caching	Mit Caching	Ohne Caching
10	48	2.032	4,8	203,2
20	88	3.388	4,4	169,4
30	124	6.323	4,1	210,8
40	173	**7.863**	4,3	196,6
50	199	**6.149**	3,9	123,0
60	244	12.096	4,1	201,6
70	287	12.744	4,1	182,1
80	318	15.019	4,0	187,7
90	360	15.389	4,0	171,0
100	392	18.538	3,9	185,4
			∅ 4,2	∅ 183,1

Tabelle 8.7: Stichproben zur Laufzeitevaluation der lex. Disambiguierung (BabelNet).

In diesem Beispiel wird jeweils nur eine Anfrage durchgeführt und somit eine nicht-repräsentative Momentaufnahme erzeugt. Dies ermöglicht es, den Einfluss der Netzwerkauslastung darzustellen. In Tabelle 8.7 ist zu erkennen, dass das Abrufen lexikalischer Informationen für 50 *Token* im Vergleich zu 40 *Token* weniger Zeit in Anspruch nimmt. Ein Effekt, der sehr wahrscheinlich auf (lokale) Netzwerkauslastung bzw. Auslastung der BabelNet-*Server* zurückzuführen ist. Um diesen Einflussfaktor besser nachvollziehen zu können, bedarf es einer längerfristigen Messung. Hierzu wird stündlich, über einen Zeitraum von zehn Tagen und unter Verwendung von zehn zufällig gewählten Messstellen (begrenzt auf Europa, u. a. Großbritannien, Deutschland, Österreich) die Serververfügbarkeit und generelle -antwortzeit von BabelNet gemessen[164] (siehe Messergebnisse in Anhang B). Abbildung B.1 zeigt die Ergebnisse der verschiedenen Messstellen, wobei ersichtlich wird, dass die Antwortzeiten stark abhängig von den Messstandorten sind. So liegt der zeitliche Durchschnitt in der Tschechischen Republik (CZ) deutlich über dem Durchschnitt der Antwortzeiten in Großbritannien (GB) und Deutschland (vgl. Abbildung B.2). Andererseits fällt auf, dass in CZ relativ konstante Zeiten erreicht werden, während beispielsweise in Deutschland auffällige Schwankungen (30-80 *ms*) zu verzeichnen sind. Zusammenfassend lassen sich somit die Vermutungen bezüglich der Netzwerkschwankungen bestätigen, da insbesondere für die Messstellen in Deutschland deutliche Schwankungen in den Antwortzeiten zu verzeichnen sind (vgl. Abbildung B.2).

Für ein besseres Verständnis, wie die Abrufzeit für *Token* über längere Zeit schwankt, wird eine vordefinierte Menge von zehn *Token*[165] herangezogen. Die Abfrage erfolgt alle zwei Stunden über zehn Tage, wobei im Folgenden zusätzlich davon ausgegangen wird, dass alle Anfragen von einem Standort unter gleichbleibenden Hardwarebedingungen (s. Abschnitt 7.2) gestellt werden.

[164] Eine Messung/Stunde pro Messstelle mit jeweils 10 Paketen.

[165] *Token*: „E-Mail“ (*bn:00029345n*), „Computer“ (*bn:00021464n*), „send“ (*bn:00090548v*), „write“ (*bn:00085489v*), „delete“ (*bn:00084456v*), „SPAM“ (*bn:00048634n*), „provider“ (*bn:00064912n*), „large“ (*bn:00116076r*), „GMX“ (*bn:00726178n*) und „Sharepoint“ (*bn:15927858n*).

Tag	Abrufzeiten [*ms*]												
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	∅
1	203	183	178	214	292	**320**	142	187	271	167	184	188	211
2	260	233	243	304	282	187	254	259	194	155	208	201	232
3	283	164	310	152	259	298	223	297	253	175	300	294	251
4	171	202	299	183	268	172	152	170	140	267	293	250	214
5	168	247	257	309	189	292	249	232	157	227	269	261	238
6	175	259	150	208	272	146	189	308	154	190	237	260	212
7	270	299	239	254	263	273	165	183	244	254	166	125	228
8	191	293	232	185	266	189	147	246	**127**	236	301	231	220
9	169	221	283	171	196	188	296	198	154	283	157	196	209
10	147	187	237	185	157	149	265	128	263	245	191	308	205

Tabelle 8.8: Durchschnittliche Abrufzeit vordefinierter *Token* über 10 Tage (BabelNet)

Tabelle 8.8 zeigt die ermittelten Durchschnittswerte der BabelNet-Anfragen, protokolliert über zehn Tage. Diese liegen im Bereich 127-320 *ms* und damit im Durchschnitt (222 *ms*) über dem Ergebnis der Stichprobe (183 *ms*, vgl. Tabelle 8.7). Im Vergleich zu den Abrufzeiten des *Cachings* sind diese Werte, ungeachtet der Schwankungen, allesamt erheblich höher. Dies überrascht wenig, da es sich um einen Vergleich zwischen lokalen und externen Anfragen handelt. Allerdings ist die erreichte Zeitersparnis, insbesondere für den Anwendungsfall des OTF-*Computings*, von erheblicher Bedeutung. Für die Performanz ist auch die Entwicklung der Anfragenverteilung nach Ressourcen von Interesse, welche maßgeblich vom verwendeten Vokabular (und Lesarten) abhängt. Diesbezüglich ergeben sich die Evaluationsfragen `Q8` und `Q9`, die
Q8 es im Folgenden zu beantworten gilt:

Die Frage `Q8` (Ist ein Sättigungszustand auszumachen, bei dessen Erreichung keine weiteren Token bzw. Lesarten in den WSD-*Cache* aufgenommen werden und somit alle Anfragen per WSD-*Cache* beantwortet werden können?) kann sowohl auf Basis von *Token* als auch von Lesarten beantwortet werden. Ein Sättigungszustand wäre erreicht, wenn keine *Token* bzw. Lesarten mehr in den *Cache* aufgenommen und alle Anfragen aus diesem Zwischenspeicher beantwortet werden können. Dies kann allerdings *per se* nicht eintreten, da die natürliche Sprache von Eigennamen und Neologismen geprägt ist, welche bestehende Ressourcen nicht (vollumfänglich) abdecken und somit Anfragen an BabelNet gestellt werden müssen, da der Zwischenspeicher keine geeigneten Einträge enthält, wenngleich diese auch dort (überwiegend) zu keinem Treffer führen. Allerdings könnte ein Sättigungszustand für *Token* bzw. Lesarten eintreten, die in lexikalischen Ressourcen abgebildet sind bzw. innerhalb einer Domäne verwendet werden. Deshalb wird im Folgenden die Untersuchung des kompletten Korpus (FA und NFA) von Dollmann (2016) hinsichtlich des Vokabularzuwachses über alle 1881 Sätze hinweg vorgenommen. Iterativ werden hierfür einzelne Sätze in *Token* aufgeteilt. Diese *Token* werden lemmatisiert und in ein – zu Beginn leeres – Lexikon aufgenommen. So kann beispielsweise der erste Satz vollständig (14 *Token*) aufgenommen werden, während vom zweiten Satz nur sechs von zehn *Token* aufgenommen werden, da die übrigen vier *Token* schon im Lexikon enthalten sind. Es zeigt sich, dass 921 Sätze (48,9%) kein *Token* zum Lexikon beitragen, da

sie bereits in Gänze durch das darin befindliche Vokabular abgedeckt werden. Nun bedeutet dies aber nicht, dass die Sätze, die zu Beginn tokenisiert und inventarisiert werden, die alleinige Grundlage für das Lexikon bilden und die darauffolgenden 921 Sätze durch das Vokabular abgedeckt sind. Es kommen immer wieder neue *Token* in den Sätzen vor. Das heißt, dass ein Sättigungszustand zwar theoretisch erreicht werden kann, jedoch auf Grund der Varianz im Vokabular schwer vorherzusehen ist. Vielmehr ist davon auszugehen, dass nach einer gewissen Anzahl von Sätzen der Vokabularzuwachs abnimmt. Für den untersuchten Datensatz lässt sich dies insofern nachweisen, als dass 70% des Vokabulars mit 50% der Sätze inventarisiert wird, während die letzten 25% der Sätze nur noch 14% der *Token* im Lexikon beitragen. **Q9**

Zur Beantwortung von Frage Q9 (Ist die Anfragenverteilung im WSD-*Cache* innerhalb einer Domäne anders ausgeprägt, als bei domänenübergreifenden Anforderungsbeschreibungen?) wird zum einen das Korpus von Dollmann (2016) als Datensatz für domänenübergreifende FA herangezogen (s. Abschnitt 6.1). Zum anderen wird die Stichprobe aus dem PAS-Korpus als domänenspezifischer Datensatz[166] hinzugenommen (s. Abschnitt 6.2). Beide Testdatensätze werden iterativ an das Softwaresystem übertragen, während gleichzeitig die Ressourcenverteilung in den Anfragen[167] überwacht wird. Abbildung B.3 im Anhang zeigt die Ressourcenanfragen, unterteilt in diejenigen, die der Zwischenspeicher entgegennimmt und jene, die an BabelNet übertragen werden. Auf den ersten Blick fällt auf, dass sich die Anfragenverteilungen in Abbildung B.3 (im Anhang) stark ähneln. Sowohl beim domänenspezifischen (vgl. Abbildung B.3, A) als auch beim domänenübergreifenden Datensatz (vgl. Abbildung B.3, B) wird zu Beginn ein Großteil der Anfragen durch BabelNet beantwortet (85% bzw. 90%), was damit zu erklären ist, dass der Zwischenspeicher erst gefüllt werden muss. Nichtsdestotrotz ist es überraschend, dass mehr Anfragen beim domänenübergreifenden Datensatz durch den Zwischenspeicher beantwortet werden. Bisher war davon auszugehen, dass innerhalb einer Domäne eher zu einem gemeinsamen Vokabular tendiert wird. Somit sollte sich beim domänenspezifischen Datensatz eine hohe Abdeckung durch den Zwischenspeicher einstellen. Diese Annahme scheint sich auch nach der Hälfte der Anfragen (300 *Token*) zu bestätigen, da nun bereits 28% beim domänenspezifischen und 22% beim domänenübergreifenden Datensatz durch den Zwischenspeicher beantwortet werden. Auch zum Ende hin (600 *Token*) wird ein größerer Anteil der Anfragen beim domänenspezifischen Datensatz vom Zwischenspeicher beantwortet (37,5% gegenüber 27,5%). Anders formuliert bedeutet das, dass in der domänenspezifischen Durchführung bereits wenige *Token* im *Cache* ausreichen, um viele Anfragen zu beantworten. Dies ist darauf zurückzuführen, dass das verwendete Vokabular innerhalb einer Domäne kleiner ist und daher bereits nach wenigen Anfragen ein nennenswerter Anteil der folgenden Anfragen aus dem *Cache* beantwortet werden kann.

8.3.5 Laufzeitanalyse der Strategien

Q10

Die Laufzeit der Strategien wird maßgeblich durch die bereits in Abschnitt 8.3.3 evaluierte Komponentenlaufzeit beeinflusst. Da sich die Strategiekonfigurationen

[166] Entnommen der Kategorie „Kommunikation“, Unterkategorie „E-Mail“.
[167] Zur besseren Vergleichbarkeit wird eine gemeinsame Basis von 600 *Token* herangezogen.

allerdings von einander unterscheiden, wird im Folgenden der direkte Vergleich der Gesamtlaufzeiten über alle Strategien hinweg durchgeführt. Hierzu werden für jede Strategie jeweils 100 Anforderungsbeschreibungen im Umfang von zwei, drei und vier Sätzen[168] an das Softwaresystem übermittelt (vgl. Abbildung 8.14).

Es ist aufgrund der Strategiekonfigurationen wenig verwunderlich, dass die *Light*-Strategie die geringste Gesamtlaufzeit aufweist (⌀ 3.182 *ms*), während die *Complete*-Strategie die meiste Zeit in Anspruch nimmt (⌀ 9.308 *ms*).

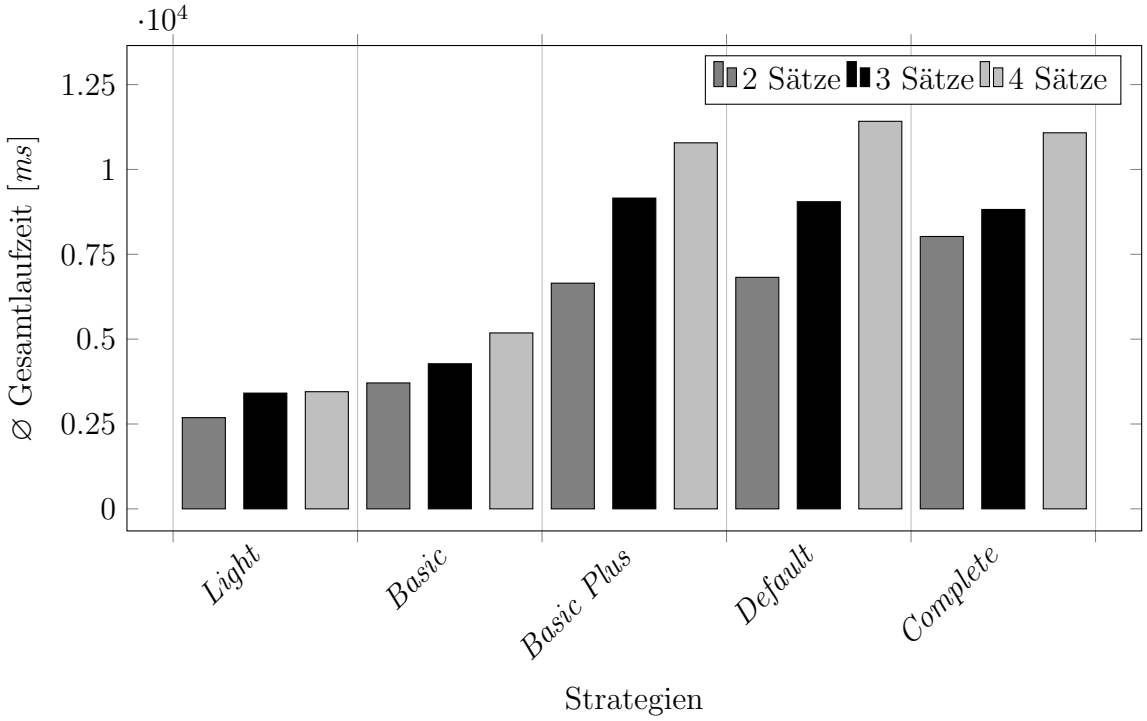


Abbildung 8.14: Gesamtlaufzeit der Strategien nach Beschreibungsumfang. Jeweils 100 Durchläufe mit aktiviertem WSD-*Cache*

Es fällt auf, dass die *Basic Plus*- (⌀ 8.863 *ms*), *Default*- (⌀ 9.096 *ms*) und *Complete*-Strategie (⌀ 9.308 *ms*) zeitlich sehr nah beieinander liegen, was auf die gewählten Kompensationskomponenten zurückzuführen ist. Damit ist sowohl die *Light*- als auch die *Basic*-Strategie (⌀ 4.389 *ms*) hinsichtlich einer performanten Ausführung zu wählen, während die *Basic Plus*-, *Default*- und *Complete*-Strategie zwar eine höhere Abdeckung in Erkennung und Kompensation aufweisen, dadurch aber auch mehr Zeit in Anspruch nehmen.

8.4 Evaluationsfazit

Die durch die Evaluation erhaltenen Einblicke in das Systemverhalten sowie die -performanz werden im Folgenden hinsichtlich ihrer Aussagekraft und Relevanz für die Weiterentwicklung des Systems diskutiert. Während sich der erste Evaluationsteil

[168] Durchschnittliche Anzahl *Token*: 29 (2 Sätze), 49 (3 Sätze), 61 (4 Sätze).

der Zuverlässigkeit von Indikatoren widmete, befasste sich der zweite Teil mit der Performanz einzelner Systembestandteile.

Zuverlässigkeit. Gegenstand des ersten Evaluationsteils waren die definierten Indikatoren aus Abschnitt 5.3, wobei insbesondere die Identifikation von Fehlerquellen sowie die Analyse von Verarbeitungsfehlern und deren Auswirkungen auf die Strategieanwendung im Mittelpunkt der Evaluation standen (s. Abschnitt 8.2). Hierbei zeigte sich, dass die vorgegebenen Strategien hinsichtlich der Indikatorabdeckung suboptimal definiert sind, da im Evaluationskorpus vermehrt bisher nicht berücksichtige Indikatorkombinationen auftreten und die Strategien hier nicht angewendet werden. Dieser Umstand ist grundsätzlich bedacht, da genau für diesen Fall die *Fallback*-Strategie konzipiert wurde (s. Abschnitt 5.2.6). Die vermehrte Anwendung der *Fallback*-Strategie unterstreicht jedoch die Notwendigkeit, langfristig auf vordefinierte Strategien zu verzichten und den Ansatz einer automatischen Strategie zu verfolgen (s. Abschnitt 5.2.6). Insbesondere, um flexibler und zuverlässiger auf Qualitätsschwankungen (unbekannte Indikatorkombinationen) in den Daten zu reagieren.

Die Indikatorzuverlässigkeit unterliegt verschiedenen Einflüssen. So zum Beispiel der Zuverlässigkeit im Sinne der Ergebnisqualität von zugrundeliegenden *Tools* sowie der (Fehler-)Toleranz der angewendeten Regeln und Muster. Hierbei ist insbesondere das Tool REaCT (Dollmann und Geierhos, 2016) für Fehler verantwortlich. Nicht, weil es besonders unzuverlässig ist, sondern weil die Ergebnisse (semantische Kategorien) in mehreren Indikatoren herangezogen werden und Fehler in der Anforderungsklassifikation und -extraktion somit besonders zum Tragen kommen, während beispielsweise *Stanford coref* nur bei dem Indikator für referentielle Ambiguität angewendet wird – Verarbeitungsfehler dieser Komponente haben demnach nur begrenzte Auswirkungen auf die generelle Indikatorzuverlässigkeit.

Um die Zuverlässigkeit der Indikatoren beurteilen zu können, wurden pro Indikator die Qualitätsmaße *Recall* und *Precision* bestimmt (s. Abschnitt 8.2.3). Wie auch in anderen Arbeiten zur Verbesserung natürlichsprachlicher Anforderungen (insb. Tjong, 2008, S. 2) wird dabei der *Recall* stärker gewichtet als die *Precision*, da es wichtig ist, möglichst alle potentiell ambigen und unvollständigen Anforderungsbeschreibungen zu identifizieren. Hierbei stellte sich insbesondere der Indikator referentieller Ambiguität als zuverlässig heraus (F_2-*Score* von 0,75), während die Kompensation von Unvollständigkeit auf Grund eines niedrigen *Recalls* nur 0,72 als F_2-*Score* erreicht.

Neben den Indikatoren können auch bei der Strategieanwendung Fehler auftreten, nämlich dann, wenn falsche Komponenten zusammenarbeiten oder die Ergebnisse verfälschen. Besonders ärgerlich ist das, wenn korrekte Ergebnisse (z. B. syntaktische Strukturen) durch Hinzunahme von Expertenkomponenten verschlechtert werden (z. B. syntaktische Disambiguierung), da diesen Komponenten in der vorliegenden Arbeit eigentlich eine Konfliktlösungskompetenz zugesprochen wird (s. Abschnitt 5.2).

Bei der Fehleranalyse zeigte sich zum einen (s. Abschnitt 8.2), dass Verarbeitungsfehler auftreten, die unterschiedliche Systemabschnitte betreffen und verschiedene Ursachen haben können. So können fehlerhafte semantische Informationen sowohl die Indikatoranwendung (erster Systemabschnitt) negativ beeinflussen als auch die strukturierte Ausgabe schädigen (letzter Systemabschnitt). Jedoch begründen Fehler

in den ersten Systemabschnitten (insb. *Preprocessing*) oftmals gravierendere Schäden (z. B. falsche Satzgrenzen und semantische Kategorien, fehlerhafte Klassifikation).

Darüber hinaus kann beispielsweise bei der Unvollständigkeitskompensation sowohl eine falsche Kompensationsanfrage auf Grund einer fehlerhaften syntaktischen Struktur als auch eine nicht erkannte Koreferenz oder Anapher (referentielle Disambiguierung) zu einem schlechten Kompensationsergebnis führen.

Zum anderen zeigte sich aber auch, dass einige Komponenten als Fehlerquellen besonders in Erscheinung treten. Dabei wurden falsch oder nicht erkannte semantische Kategorien (Informationsextraktion) auf Grund des weitreichenden und gravierenden Einflusses hervorgehoben. Aber auch die fehlerhafte Erkennung syntaktischer Strukturen führt zu weitreichenden Folgen. Auf technischer Ebene ist damit der Einfluss des *Stanford CoreNLP-Tools* in dieser Arbeit als sehr groß zu bezeichnen, findet es doch sowohl bei der Satzgrenzenerkennung, Satzvereinfachung, syntaktischen und referentiellen Disambiguierung als auch als Expertenkomponente in der Unvollständigkeitskompensation Anwendung.

Performanz. Neben der Ergebnisverschlechterung, die Fehler in der Indikator- und Strategieanwendungen bedingen können, ist auch der potentielle negative Einfluss auf die Systemperformanz zu nennen. Insbesondere dann, wenn Komponenten ausgeführt werden, die nicht ausgeführt werden müssten. Um mehr über die Performanz des Gesamtsystems und darüber hinaus auch über die Laufzeiten der einzelnen Komponenten und Strategien zu erfahren, wurde der zweite Evaluationsteil der Systemperformanz gewidmet (s. Abschnitt 8.3). Hierbei standen insbesondere die Identifikation von Leistungsengpässen auf Seiten des Servers sowie die Steigerung der Nutzerzufriedenheit durch Performanzverbesserung im Zentrum der Überlegungen.

Die Evaluation der Verarbeitungskomponenten zeigte, dass die Satzgrenzenerkennung, die Anforderungsklassifikation sowie -extraktion und die lexikalische Disambiguierung zeitintensive Komponenten sind, wobei die lexikalische Disambiguierung die mit Abstand höchste (und dazu unberechenbare[169]) Laufzeit aufweist. Dies ist problematisch, da es für REaCT (Anforderungsklassifikation und -extraktion) derzeit keinen äquivalenten Ersatz gibt (s. Abschnitt 3.2) und es sich bei Babelfy um die (aktuell) beste Softwarelösung zur WSD handelt und beide Komponenten daher nicht ersetzt werden können. Vielmehr gilt es andere Wege zu finden, die Performanz zu steigern: So wurde exemplarisch ein *Caching*-Verfahren für die Disambiguierung mittels Babelfy und BabelNet eingeführt und evaluiert, welches als Zwischenspeicher fungiert und somit einzelne Abfragen wesentlich performanter beantworten kann. Wie sich während der Evaluation zeigte, ist die Zeiteinsparung dabei erheblich (vgl. Tabelle 8.7), was vor allem der Nutzerzufriedenheit (geringere Wartezeit) zugute kommt. Jedoch lässt sich ein solches *Caching* nur für statische Informationen (wie lexikalische Informationen zu Lesarten aus BabelNet) realisieren und ist demnach für die meisten Komponenten im untersuchten Softwaresystem ungeeignet. Ferner ist es vor allem dann effizient, wenn es innerhalb einer Domäne zum Einsatz kommt und dadurch das Vokabular im Umfang begrenzt ist (schnellere Abdeckung).

Die Evaluation der Strategien (s. Abschnitt 8.3.5) bestätigt die offensichtliche Annahme, dass umfangreichere Strategien (höhere Indikatorabdeckung) mit einer

[169] Unberechenbar im Sinne hoher Netzwerkschwankungen / schwankender Abrufzeiten.

längeren Laufzeit einhergehen. Allerdings ist festzustellen, dass auch hier einzelne Komponenten (insb. die lexikalische Disambiguierung) einen größeren Anteil an der Gesamtlaufzeit bedingen als andere. So unterscheidet sich die *Default*-Strategie von der *Complete*-Strategie vor allem im *Output* und in der Vagheitserkennung, was sich in den Gesamtlaufzeiten nicht bemerkbar macht. Zeitlich nur minimal schneller als die *Default*-Strategie ist die *Basic Plus*-Strategie, obwohl diese noch die referentielle Disambiguierung ausführt. Diese ist allerdings (in der jetzigen Form) eine sehr performante Komponente (vgl. Tabelle 8.6) und fällt daher hinsichtlich der Laufzeit nicht ins Gewicht.

Teil IV

Fazit und Ausblick

9 Zusammenfassung und Reflexion

Mehrdeutigkeiten, Vagheit und Unvollständigkeit sind als Herausforderungen der natürlichsprachlichen Anforderungsbeschreibung in der Wissenschaft und Praxis seit langer Zeit bekannt. Die Erkennung und/oder Kompensation sprachlicher Ungenauigkeiten und Unvollständigkeit ist Gegenstand vieler wissenschaftlicher Arbeiten und praxisnaher Handlungsempfehlungen (s. Kapitel 3), deren resultierende Handlungsvorschläge von der Nutzung unterschiedlicher Lesetechniken, *Checklisten* und kontrollierter Sprachen bis hin zur Anwendung spezieller Software reichen. Sie alle vereint die Annahme, dass *Stakeholder* sich zum einen des Handlungsbedarfs zur Erstellung aussagekräftiger Anforderungsbeschreibungen bewusst sind sowie zum anderen gewillt und fähig sind, potentielle Defizite in ihren Anforderungsbeschreibungen zu beheben (s. Abschnitt 1.1). Diese Vorstellung bewahrheitet sich im OTF-*Computing* nicht, da weder die Möglichkeit einer umfassenden Benutzerinteraktion besteht, noch angenommen werden kann, dass Endanwender die erforderlichen Spezifikationskenntnisse aufweisen.

Deshalb leistet die vorliegende Arbeit einen Beitrag zur vereinfachten (disambiguierten, syntaktisch vollständigen) Kommunikation zwischen Endanwendern, die ihre individuellen Anforderungen an eine geplante Software beschreiben und Entwicklern, welche diese final umsetzen müssen. Zwar existieren bereits hochspezialisierte Softwarelösungen, die einzelne Defizite in natürlichsprachlichen Texten automatisiert erkennen und korrigieren können, jedoch sind diese auf die Domäne nicht adaptierbar. Sie zielen nicht auf Anforderungsbeschreibungen ab und übersteigen aufgrund ihrer Bedienungskomplexität (z. B. Ein- und Ausgabeformate, Schnittstellen) meist die Anwenderkompetenz.

Mit dieser Positionierung widmete sich diese Dissertation der Erkennung und Kompensation struktureller, referentieller und lexikalischer Ambiguität sowie Unvollständigkeit in Anforderungsbeschreibungen. Außerdem war es das Ziel, die Auswahl, Steuerung und Abstimmung der notwendigen Kompensationskomponenten zu automatisieren sowie mit CORDULA ein endanwenderfreundliches Softwaresystem zu Testzwecken zu implementieren (s. Kapitel 4). Eine bedarfsgerechte Analyse und Kompensation qualitativ stark schwankender Anforderungsbeschreibungen hinsichtlich mehrerer möglicher Defizite und ohne Benutzerinteraktion ist dabei ein Novum.

Die geringe Qualität der Anforderungsbeschreibungen (s. Abschnitt 1.4) ist dabei eine Herausforderung für die maschinelle Textverarbeitung. Als wirksame Gegenmaßnahmen haben sich dabei die Anforderungsklassifikation sowie die Satzvereinfachung erwiesen. Nichtsdestotrotz erschwert die geringe Textqualität die Anwendung von elementaren Verarbeitungsschritten, wie beispielsweise der Anforderungsextraktion im *Preprocessing*. Hierauf wird in dieser Arbeit mittels zahlreicher Regeln und Testverfahren bzw. dem Abgleich von Informationen (z. B. im Falle widersprüchlicher POS-*Tags*) und der Benennung von Expertenkomponenten reagiert.

Zur bedarfsgerechten Analyse und Kompensation wurden kontextspezifische Indikatoren definiert, die Anforderungsbeschreibungen der Endanwender analysieren und über erkannte Merkmale bzw. Merkmalsmuster notwendige Verarbeitungs- und Kompensationskomponenten auswählen. Indikatoren können dabei auf unterschiedliche linguistische Merkmale zurückgreifen, die in dieser Arbeit definiert wurden. Die semantischen Kategorien als Kernkomponenten einer FA haben dabei wesentliche Bedeutung, da sie von fast allen Indikatoren einbezogen werden. Semantische Kategorien sind das Ergebnis der Anforderungsextraktion und robuster gegenüber schlechter Textqualität, verglichen mit klassischen linguistischen Merkmalen wie POS-*Tags*, *Chunks* oder Lexikonabfragen. Die Kombination semantischer Kategorien und klassischen linguistischen Merkmalen zur bedarfsgerechten Anwendung verschiedener Kompensationsverfahren erfolgt erstmalig in dieser Arbeit und ist wesentlich für die Zusammenführung von Anforderungsbeschreibungen und Softwarekomponenten. Die stark schwankende Textqualität der Anforderungsbeschreibungen begründet darüber hinaus auch, dass die Komponentensteuerung und die Ergebnisabstimmung bedarfsgerecht über flexible Strategien erfolgen muss (s. Abschnitt 5.2). Hierbei bezieht sich Flexibilität sowohl auf die interne Informationsverarbeitung einer Strategie als auch auf die Möglichkeit, Strategien neu aufzunehmen oder zu entfernen. Strategien unterscheiden sich primär hinsichtlich der unterstützten Verarbeitungs- und Kompensationkomponenten aber auch im Umfang der internen Weiterverarbeitung. Neben vorkonfigurierten Strategien (z. B. *Light*, *Basic*, *Default*) ist eine *Fallback*-Strategie vorgesehen, die immer dann greift, wenn eine erkannte Indikatorkombination durch die bestehenden Strategien nicht abgedeckt wird (s. Abschnitt 5.2.6).

Das konzipierte Softwaresystem (CORDULA) formt aus den Indikatoren, Strategien sowie Verarbeitungs- und Kompensationskomponenten eine Anwendungseinheit, was insbesondere bedeutet, dass die Kommunikation zwischen den einzelnen Systembestandteilen, im Sinne einheitlicher Ein- und Ausgabeformate, hergestellt sowie zwischen Endanwendern und dem Gesamtsystem, mittels Benutzerschnittstellen, ermöglicht wird (s. Abschnitt 5.5). Die Konzeption orientiert sich dabei, wie auch die Implementierung, an den identifizierten Qualitätsmerkmalen moderner Softwaresysteme, so zum Beispiel der Forderung nach Interoperabilität, Portabilität und guter Wartbarkeit (s. Abschnitt 7.4). Der daraufhin entwickelte Prototyp von CORDULA umfasst alle wesentlichen Bestandteile des Konzepts, insbesondere das Zusammenwirken von Indikatoren, Strategien und Komponenten (s. Kapitel 7).

Gegenstand der Implementierung ist auch die Entwicklung von bisher nicht existenten Komponenten, wie es zum Beispiel bei der Unvollständigkeitskompensation der Fall ist. Zwar wird in der Literatur bereits die Erkennung von unvollständigen Prädikaten durchgeführt (z. B. Körner, 2014), eine automatische Kompensation von Anforderungsbeschreibungen auf Basis der Prädikat-Argument-Struktur einzelner FA ist aber ein Novum (Bäumer und Geierhos, 2016; Geierhos und Bäumer, 2016) und fester Bestandteil dieser Arbeit. Besonders hervorzuheben ist dabei die bedarfsgerechte Hinzunahme weiterer Komponenten (z. B. syntaktische Disambiguierung), sollte sich durch die Unvollständigkeitskompensation erneut potentielle Ambiguität ergeben. Diese Berücksichtigung möglicher negativer Folgen der eigenen Kompensationsaktivität ist ebenfalls bisher nicht Gegenstand der Forschung.

Allerdings umfasst ein großer Teil der Implementierung neben der reinen Softwareentwicklung auch die Erstellung von linguistischen Ressourcen, die im Bereich der natürlichsprachlichen Softwareanforderungen in erforderlichem Umfang kaum existieren (s. Kapitel 6). Hier sind in dieser Arbeit Anforderungsbeschreibungen herangezogen worden, die die qualitativen Merkmale der Anforderungen enthalten, die im OTF-*Computing* zu erwarten sind. Die resultierenden Ressourcen (insb. Anforderungsbeschreibungs- und PAS-Korpus) sind als Ergebnisse dieser Arbeit für weitere Arbeiten im Bereich der Anforderungsanalyse und -kompensation eine Hilfestellung, lösen sie in Teilen doch das ressourcenbedingte Kaltstartproblem vieler Ansätze. Allerdings ist der Umfang der erstellten Ressourcen immer noch begrenzt, sodass die Ressourcenerweiterung weiterhin Forschungsgegenstand sein muss. Dies bezieht sich dabei nicht nur auf die beiden Korpora, sondern auch auf unscheinbarere Ressourcen wie Synonymliste oder *Blacklists*.

Durch die Evaluation der Indikatoren auf realen Anforderungsbeschreibungen konnte aufgezeigt werden, dass die definierten Indikatoren eine zufriedenstellende Zuverlässigkeit aufweisen (∅ F_2-*Score* von 0,80). Jedoch wurde andererseits auch festgestellt (s. Abschnitt 5.2), dass bestimmte Strategiekonfigurationen nie Anwendung finden (z. B. Strategien ohne lexikalische Disambiguierung), während andere Indikatorkombinationen von bestehenden Strategiekonfigurationen nicht abgedeckt werden (s. Abschnitt 8.2.2). Wie die Evaluation der Indikatorkombinationen und der Strategien aufzeigte (s. Abschnitt 8.4), trägt die *Fallback*-Strategie durch ihre datengetriebene *ad hoc*-Konfiguration wesentlich zur Indikatorabdeckung bei.

Im Sinne einer höheren Flexibilität erscheint es langfristig grundsätzlich sinnvoll, gänzlich auf eine automatische Kompensationsstrategie auszuweichen. Jedoch ist dies derzeit mangels umfangreicher Ressourcen (insb. Anforderungsbeschreibungskorpora) noch nicht umsetzbar. Die vorliegende Arbeit stellt hierzu jedoch bereits wichtige Erkenntnisse bereit: Zum einen stellen die definierten Indikatoren eine robuste Grundlage zur Konfiguration einer automatischen Strategie als auch zur Auswahl vordefinierter Strategien dar. Zum anderen ist das Zusammenwirken der heterogenen Komponenten grundsätzlich auch ohne vordefinierte Strategien möglich, vorausgesetzt, die Steuerung und Ergebniskonsolidierung einzelner Komponenten wird dem *Controller* als übergeordnete Instanz übertragen.

Hinsichtlich der Performanz des Gesamtsystems ist anzumerken (s. Abschnitt 8.3), dass die bei der Evaluation gemessenen Laufzeiten stark schwanken. So schwankt beispielsweise die Ausführungszeit bei fünf Sätzen zwischen 1,1 und 26,6 s, was auf eine Vielzahl an Einflussfaktoren zurückzuführen ist, beispielsweise die Einbindung der externen Babelfy-Komponente (Netzwerkauslastung). Diesbezüglich zeigte sich, dass das beispielhaft für die lexikalische Disambiguierung implementierte *Caching*-Verfahren eine effiziente Möglichkeit ist, die Gesamtlaufzeit wesentlich zu reduzieren (s. Abschnitt 8.3.4). Allerdings ist es trotz der Tatsache, dass teilweise sehr gute Gesamtlaufzeiten erreicht werden, fraglich, ob diese bereits die Anforderungen des OTF-*Computings* erfüllen. Bei der endgültigen Wertung ist hierbei die vorliegende Hardwarekonfiguration des Testsystems zu beachten, die verhältnismäßig schwach gewählt ist (s. Abschnitt 7.2) und dennoch bereits gute Gesamtlaufzeiten zulässt.

Abschließend ist als zentrales Ergebnis dieser Dissertation festzustellen, das kontextsensitive Indikatoren und Strategien in der Lage sind, stark heterogene Verarbeitungs-

und Kompensationskomponenten bedarfsgerecht auf Anforderungsbeschreibungen anzuwenden sowie die Einzelergebnisse der Komponenten hinsichtlich eines gemeinsamen Kompensationsergebnisses in Einklang zu bringen und strukturiert weiterzugeben. Sie stellen damit eine gute Ergänzung zu bestehenden Arbeiten im Bereich der softwarebasierten Qualitätsverbesserung von natürlichsprachlichen Anforderungsbeschreibungen dar. Allerdings ist auch anzumerken, dass die vorgestellten Indikatoren und Strategien nur so gut funktionieren können, wie die zugrundeliegenden Informationen bzw. Merkmale es zulassen. Anforderungsbeschreibungen wird diesbezüglich eine schlechte Textqualität attestiert (s. Abschnitt 1.4), was die Merkmalserkennung in der Indikatoranwendung erschwert. Darüber hinaus sind die Kompensationsergebnisse nur so gut, wie die Kompensationskomponenten, die diesbezüglich herangezogen werden. Hier kann jedoch positiv in die Zukunft geschaut werden, da die natürliche Sprachverarbeitung sowie das RE als aktive Forschungsgebiete zählen und die gewählten Verarbeitungs- und Kompensationskomponenten sich überwiegend in aktiver Entwicklung befinden. Hier gilt: Verbessert sich die Zuverlässigkeit der einzelnen Komponenten, verbessert sich auch das Gesamtergebnis.

Fazit. Damit leistet die vorliegende Arbeit methodisch gesehen einen Beitrag zur ganzheitlichen Erfassung und Verbesserung sprachlicher Unzulänglichkeiten in nutzergenerierten Anforderungsbeschreibungen, indem erstmalig parallel und sequenziell Ambiguität, Unvollständigkeit und Vagheit behandelt werden. Erst durch den Einsatz linguistischer Indikatoren war es möglich, datengetrieben und bedarfsorientiert die individuelle Textqualität zu optimieren, indem von der klassischen Textanalysepipeline (s. Abschnitt 3.1) abgewichen wurde: Die *ad hoc*-Konfiguration der Kompensationspipeline, ausgelöst durch die *On-The-Fly* festgestellten Defizite in den Anforderungsbeschreibungen der Endanwender, ist ein Alleinstellungsmerkmal.

10 Forschungsausblick

Die vorliegende Arbeit stellt ein für sich genommen abgeschlossenes Forschungsvorhaben dar. Sie ist jedoch darüber hinaus als wissenschaftlicher Beitrag zum Sonderforschungsbereich 901: OTF-*Computing* zu verstehen, in dessen Rahmen die Weiterentwicklung vorgesehen ist. Diesbezüglich wird im Folgenden ein Forschungsausblick gegeben, der an die bisherigen Ergebnisse dieser Arbeit anknüpft und über die reine Ergebnisverbesserung hinausgeht.

10.1 Vom Endanwender lernen

In der vorliegenden Arbeit wird an vielen Stellen von Ressourcen gesprochen (insb. Kapitel 6), die für das Softwaresystem erforderlich sind aber nicht in der erforderlichen Qualität oder im notwendigen Umfang vorliegen. Eine Überlegung, die hierzu bislang noch nicht angestellt wurde, ist die des Erlernens von Wissen und Entscheidungen seitens des Gesamtsystems. Dabei bezieht sich das Lernen tatsächlich auf die Aneignung von Wissen sowie eines bestimmten Verhaltens durch Erfahrungen über die Zeit (Dudenredaktion, 2017a).

Bezogen auf Ressourcen bedeutet das, dass Eingaben von Endanwendern seitens des Systems dazu genutzt werden, die bestehenden Ressourcen zu erweitern. So lässt sich beispielsweise der Datenbestand der Unvollständigkeitskompensation (s. Abschnitt 5.5.5) automatisiert mittels Eingaben von Endanwendern (Anforderungsbeschreibungen) erweitern (Extraktion von FA, Identifikation von Prädikaten, Leerstellen und Kontext). Der Variantenreichtum in den Anforderungsbeschreibungen führt darüber hinaus dazu, dass die Kompensationskomponenten eine wachsende Anzahl an kontextspezifischen Kompensationsalternativen pro Prädikat erhält. Somit wird die Kompensation insgesamt präziser indem Kompensationsalternativen besser in den jeweiligen Kontext einer kompensationsbedürftigen Anforderungsbeschreibung einbettet werden können. Allerdings ist sicherzustellen, dass die Eingaben, die Endanwendern tätigen, weiterhin vertraulich behandelt werden: Es muss eine Anonymisierung der Datenbestände erfolgen.

Darüberhinaus ist Lernen in der Entscheidungsfindung möglich, denn an mehreren Stellen im Kompensationsprozess ist davon auszugehen, dass das System sich wiederholt gleich entscheidet, da in einem gegebenen Kontext beispielweise nur eine plausible Disambiguierung für ein Lexem oder einen Satz in Frage kommt. In diesem Fall liegt zwar eine potentielle Ambiguität vor, eine Disambiguierung ist aber strenggenommen nicht performant, da mit hoher Wahrscheinlichkeit bereits vor der Disambiguierung auf Basis der Vergangenheitswerten bestimmt werden kann, welche Lesart korrekt ist. Ein naheliegendes Beispiel hierfür ist die lexikalische Disambiguierung. Es ist davon auszugehen, dass Endanwender, die eine E-Mail-Applikation beschreiben, bei-

spielsweise das Prädikat „*send*“ immer in der gleichen Lesart verwenden (z. B. auch bei „*attachment*“ und „*write*“). Um die Kompensation performant zu gestalten, ist eine erweiterte *Whitelist* denkbar, die Disambiguierungsergebnisse und weitere Informationen über den Kontext enthält und der eigentlichen Disambiguierung vorgeschaltet wird. Befindet sich ein Lexem auf der *Whitelist* und handelt es sich um einen ähnlichen Kontext sowie Domäne, kann die (zeitaufwändigere) Disambiguierung über Babelfy übersprungen werden und die wahrscheinliche Lesart zugeordnet werden.

10.2 Extraktion und Erweiterung funktionaler Abläufe

In dieser Arbeit wird eine funktionale Anforderung, vereinfacht dargestellt, als eine Aneinanderreihung semantischer Kategorien verstanden (z. B. Rolle, Aktion, Objekt), deren Instantiierung unvollständig, vage oder mehrdeutig sein kann und die somit als kompensationsbedürftig gilt. Diese Betrachtungsweise lässt dabei bislang (mit Ausnahme der referentiellen Disambiguierung) das Zusammenwirken mehrerer FA außen vor. Aufbauend auf den bisherigen, in dieser Arbeit dargestellten, Ergebnissen wird diesbezüglich angestrebt, auch funktionale Abläufe aus den Anforderungsbeschreibungen zu extrahieren – somit satzübergreifend die gewünschten Funktionalitäten (Prozesswörter) zu extrahieren und in eine Ausführungsreihenfolge zu bringen. Dies ist von besonderer Relevanz, da Anforderungsbeschreibungen es zulassen, dass Prozesswörter in beliebiger Reihenfolge kombiniert und über Temporalausdrücke wie Temporaladverben (z. B. „*afterwards*“) oder temporale Präpositionen (z. B. „*before*“) koordiniert werden (Landhäußer, 2016, S. 92 ff.). Ein Beispiel hierfür ist die FA „*I want to **send** emails to my friends: First I need to **write** them and then I want to **attach** my files*“. Die bisherige Betrachtungweise nimmt für solche FA eine intendierte funktionale Ausführungsreihenfolge an (vgl. Abbildung 10.1, A).

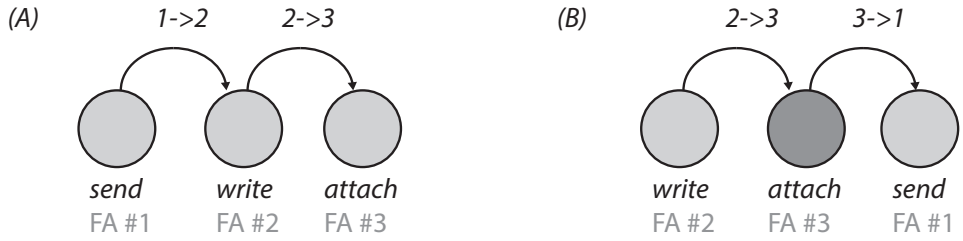


Abbildung 10.1: Gegenüberstellung generierter Funktionsabläufe

Diese Annahme erscheint, aufgrund der geringen Textqualität (s. Abschnitt 1.4) sowie der Möglichkeit der spontanen Anforderungsverschriftlichung (s. Abschnitt 5.1), als gewagt. Sie ist für die vorliegende Arbeit jedoch ausreichend. Wird allerdings auch die Ausführungsreihenfolge betrachtet, fällt auf, dass das Ergebnis wie in Abbildung 10.1 (A) nicht ausführbar ist, da das Senden („*send*“) der E-Mail unmöglich vor dem Schreiben („*write*“) erfolgen kann. Folglich unterliegen die Prozesswörter in der Ausführung mindestens einer temporalen Anordnung (vgl. Abbildung 10.1, B). Sie unterliegen aber nicht nur einer temporalen, sondern auch einer hierarchischen Anordnung (vgl. Abbildung 10.2): So ist das Senden zwar zunächst als gleichwertiges

Prozesswort zu verstehen, es setzt aber zugleich (in diesem Beispiel) das Schreiben eines Textes und das Anhängen („*attach*“) einer Datei voraus.

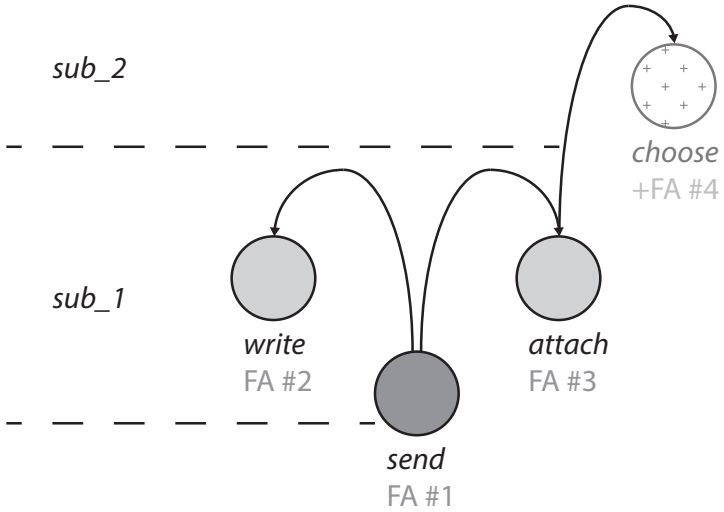


Abbildung 10.2: Gegenüberstellung generierter Funktionsabläufe

Neben der angesprochenen Ausführungsreihenfolge genannter Prozesswörter ist die Erweiterung bestehender Prozessabläufe zu diskutieren. So setzt nicht nur das Senden einer E-Mail mit Anhang das Schreiben und das Anhängen voraus. Vielmehr setzt beispielsweise auch das Anhängen einen weiteren Prozessschritt voraus, nämlich die Auswahl („*choose*“) entsprechender Dateien – ein Prozesswort, was in der initialen Anforderungsbeschreibung keine Verwendung findet (vgl. Abbildung 10.2). Hierzu ist eine entsprechende Wissensressource erforderlich, die Prozesswörter, deren hierarchische Beziehung zu anderen Prozesswörtern sowie Abhängigkeiten, linguistische Relationen (z. B. Synonymie) und Kontextinformationen (z. B. Domäne) enthält. Eine solche Ressource existiert derzeit noch nicht.

Literaturverzeichnis

ACL Wiki (2016). POS Tagging (State of the art). https://www.aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art)&oldid=11577. Zuletzt abgerufen am 30.03.2017.

Agarwal, R. und Boggess, L. (1992). A Simple but Useful Approach to Conjunct Identification. In *Proceedings of the 30th Annual Meeting on ACL*, ACL'92, Seiten 15–21, Stroudsburg, PA, USA. ACL.

Agirre, E., Baldwin, T. und Martinez, D. (2008). Improving Parsing and PP attachment Performance with Sense Information. In *Proceedings of the Annual Meeting of the ACL*, Seiten 317–325, Columbus, OH, USA. ACL.

Agirre, E. und Edmonds, P. (Herausgeber) (2007). *Word Sense Disambiguation: Algorithms and Applications*, Band: 33. *Text, Speech and Language Technology*. Springer, Baskenland, Spanien / Oxford, UK.

Albayrak, O., Kurtoglu, H. und Biaki, M. (2009). Incomplete Software Requirements and Assumptions Made by Software Engineers. In *Proceedings of the 16th APSEC*, Seiten 333–339, Batu Ferringhi, Penang, Malaysia. IEEE.

Allen, J. (1995). *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, New York, NY, USA / Wokingham, UK / Amsterdam, Niederlande / Bonn u. a.

Alshazly, A. A., Elfatatry, A. M. und Abougabal, M. S. (2014). Detecting defects in software requirements specification. *Alexandria Engineering Journal*, 53(3):513–527.

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S. und Collins, M. (2016). Globally Normalized Transition-Based Neural Networks. In *Proceedings of the 54th Annual Meeting of the ACL*, Seiten 2442 – 2452, Berlin. ACL.

Apache Software Foundation (2012). Apache OpenNLP Developer Documentation. https://opennlp.apache.org/documentation/1.5.2-incubating/manual/opennlp.html#tools.chunker. Zuletzt abgerufen am 12.12.2016.

Apache Software Foundation (2016). Apache Solr 6.3.0: Solr Features. http://lucene.apache.org/solr/features.html. Zuletzt abgerufen am 08.12.2016.

Aurum, A., Petersson, H. und Wohlin, C. (2002). State-of-the-art: software inspections after 25 years. *Software Testing, Verification and Reliability*, 12(3):133–154.

Avci, O. (2008). Warum entstehen in der Anforderungsanalyse Fehler? Eine Synthese empirischer Befunde der letzten 15 Jahre. In *Industrialisierung des Software-Managements: Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -Wartung im Fachbereich Wirtschaftsinformatik*, LNI, Seiten 89–103, Stuttgart. GI.

Bailey, D., Lierler, Y. und Susman, B. (2015). Prepositional Phrase Attachment Problem Revisited: How VERBNET Can Help. In *Proceedings of the 11th IWCS*, Seiten 12–22, London, UK. ACL, ACL.

Bajwa, I. S., Lee, M. und Bordbar, B. (2012). Resolving Syntactic Ambiguities in Natural Language Specification of Constraints. In Gelbukh, A. (Herausgeber), *Computational Linguistics and Intelligent Text Processing*, Band: 7181. *LNCS*, Seiten 178–187. Springer, Berlin / Heidelberg.

Baker, C. F., Fillmore, C. J. und Lowe, J. B. (1998). The Berkeley FrameNet Project. In *Proceedings of the 36th Annual Meeting of the ACL and 17th International Conference on COLING*, Band: 1, Seiten 86–90, Montreal, QC, Kanada. ACL.

Bakshi, R. N. (2000). *A Course In English Grammar*. Orient Longman, Hyderabad, TS, Indien.

Baldwin, B. (1997). CogNIAC: High Precision Coreference with Limited Knowledge and Linguistic Resources. In *Proceedings of ANARESOLUTION '97*, Seiten 38–45, Madrid, Spanien. ACL.

Baldwin, T. und Lui, M. (2010). Language Identification: The Long and the Short of the Matter. In *Proceedings of the HLT: The 2010 Annual Conference of the NAACL*, Seiten 229–237, Los Angeles, CA, USA. ACL.

Balzert, H. (2003). *JSP für Einsteiger - Dynamische Websites mit Java Server Pages erstellen*. IT lernen. W3L-Verlag, Herdecke / Dortmund.

Balzert, H. (2009). *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, Heidelberg, 3. Auflage.

Banerjee, S. und Pedersen, T. (2002). An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet. In Gelbukh, A. (Herausgeber), *Proceedings of the CICLing 2002*, LNCS, Seiten 136–145, Mexico City, Mexiko / Berlin / Heidelberg. Springer.

Bano, M. (2015). Addressing the Challenges of Requirements Ambiguity: A Review of Empirical Literature. In *Proceedings of the 5th International Workshop on EmpiRE*, Seiten 21–24, Ottawa, ON, Kanada. IEEE.

Bansal, M. und Klein, D. (2012). Coreference Semantics from Web Features. In *Proceedings of the 50th Annual Meeting of the ACL*, Band: 1. *ACL'12*, Seiten 389–398, Stroudsburg, PA, USA. ACL.

Bäumer, F. S. und Geierhos, M. (2016). Running out of Words: How Similar User Stories Can Help To Elaborate Individual Natural Language Requirement Descriptions. In Dregvaite, G. und Damasevicius, R. (Herausgeber), *Proceedings of the ICIST 2016*, CCIS, Seiten 549–558, Druskininkai, Litauen. Springer.

Baumgartner, M., Klonk, M., Pichler, H., Seidl, R. und Tanczos, S. (2013). *Agile Testing: Der agile Weg zur Qualität*. Carl Hanser Verlag, München.

Bell, T. E. und Thayer, T. A. (1976). Software Requirements: Are They Really a Problem? In *Proceedings of the 2nd ICSE*, ICSE'76, Seiten 61–68, Los Alamitos, CA, USA. IEEE.

Beneken, G. (o. D.). Informelle und formale Spezifikation. `http://www.software-kompetenz.de/servlet/is/15728/`. Zuletzt abgerufen am 24.07.2015.

Bengtson, E. und Roth, D. (2008). Understanding the Value of Features for Coreference Resolution. In *Proceedings of the 2008 Conference on EMNLP*, Seiten 294–303, Honolulu / Urbana, IL, USA. ACL.

Berghuber, M. (2008). Ambiguität. `http://www.rheton.sbg.ac.at/rheton/2008/12/ambiguitaet/`. Zuletzt abgerufen am 23.10.2015.

Bergsten, H. (2004). *JavaServer Pages*. O'Reilly Verlag, Sebastopol, CA, USA, 3. Auflage.

Berkeley NLP Group (2016). Berkeley Coreference Resolution System. `http://nlp.cs.berkeley.edu/projects/coref.shtml`. Zuletzt abgerufen am 16.05.2016.

Berry, D. M. (2000). The Requirements Iceberg and Various Icepicks Chipping at it. `http://www.ieee.li/pdf/viewgraphs/iceberg.pdf`. Zuletzt abgerufen am 23.10.2015.

Berry, D. M., Kamsties, E. und Krieger, M. M. (2003). From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity – A Handbook. Version 1.0. `https://cs.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf`. Zuletzt abgerufen am 16.11.2015.

Berzins, V., Martell, C., Luqi und Adams, P. (2008). Innovations in Natural Language Document Processing for Requirements Engineering. In Paech, B. und Martell, C. (Herausgeber), *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs: 14th Monterey Workshop 2007. Revised Selected Papers*, Seiten 125–146. Springer, Monterey, CA, USA / Berlin / Heidelberg.

Bhargav, A. und Kumar, B. (2010). *Secure Java: For Web Application Development*. CRC Press, Boca Raton, FL, USA.

Bhat, M., Ye, C. und Jacobsen, H.-A. (2014). Orchestrating SOA Using Requirement Specifications and Domain Ontologies. In Franch, X., Ghose, A., Lewis, G. und

Bhiri, S. (Herausgeber), *Service-Oriented Computing*, Band: 8831. *LNCS*, Seiten 403–410. Springer, Berlin / Heidelberg.

Björkelund, A., Bohnet, B., Hafdell, L. und Nugues, P. (2010). A High-Performance Syntactic and Semantic Dependency Parser. In *COLING 2010: Demonstrations*, Seiten 33–36, Beijing, China. COLING 2010 Organizing Committee.

Björkelund, A. und Kuhn, J. (2014). Learning Structured Perceptrons for Coreference Resolution with Latent Antecedents and Non-local Features. In *Proceedings of the 52nd Annual Meeting of the ACL*, Seiten 47–57, Baltimore, MD, USA. ACL.

Boehm, B. W. (1984). Verifying and Validating Software Requirements and Design Specifications. *IEEE Software*, Seiten 75–88.

Bohnet, B. (2010). Very High Accuracy and Fast Dependency Parsing is Not a Contradiction. In *Proceedings of the 23rd COLING*, COLING'10, Seiten 89–97, Beijing, China / Stroudsburg, PA, USA. ACL.

Bos, J. und Spenader, J. (2011). An annotated corpus for the analysis of VP ellipsis. *Language Resources and Evaluation*, 45(4):463–494.

Brants, S., Dipper, S., Hansen, S., Lezius, W. und Smithm, G. (2002). The TIGER Treebank. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, TLT'02, Seiten 24–41, Sozopol, Bulgarien.

Bray, I. (2002). *An Introduction to Requirements Engineering.* Pearson Education, Harlow, Essex, UK.

Breindl, E. und Donalies, E. (2012). Intensitätspartikel. `http://hypermedia.ids-mannheim.de/call/public/sysgram.ansicht?v_id=391`. IDS Mannheim. Zuletzt abgerufen am 28.09.2016.

Briscoe, T. (2006). An introduction to tag sequence grammars and the RASP system parser. Technischer Bericht 662, University of Cambridge, Cambridge, UK. UCAM-CL-TR-662. Erreichbar unter: `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-662.pdf`. Zuletzt abgerufen am 28.04.2016.

Briscoe, T. und Carroll, J. (2002). Robust Accurate Statistical Annotation of General Text. In *Proceedings of the 3rd LREC*, Las Palmas, Spanien. ELRA.

Briscoe, T., Carroll, J. und Watson, R. (2006). The Second Release of the RASP System. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, COLING-ACL'06, Seiten 77–80, Stroudsburg, PA, USA. ACL.

Brugger, R. (2009). *IT-Projekte strukturiert realisieren: Situationen analysieren, Lösungen konzipieren – Vorgehen systematisieren, Sachverhalte visualisieren – UML und EPKs nutzen.* Vieweg+Teubner Verlag, Wiesbaden, 2. Auflage.

Bryl, V., Giuliano, C., Serafini, L. und Tymoshenko, K. (2010). Using Background Knowledge to Support Coreference Resolution. In *Proceedings of the 19th ECAI*, Seiten 759–764, Amsterdam, Niederlande. IOS Press.

Bucchiarone, A., Gnesi, S., Fantechi, A. und Trentanni, G. (2010). An Experience in Using a Tool for Evaluating a Large Set of Natural Language Requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC'10, Seiten 281–286, New York, NY, USA. ACM.

Budanitsky, A. und Hirst, G. (2001). Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and other Lexical Resources, Second Meeting of the NAACL*, Band: 2, Pittsburgh, PA, USA. ACL.

Budanitsky, A. und Hirst, G. (2006). Evaluating WordNet-based Measures of Lexical Semantic Relatedness. *Computational Linguistics*, 32(1):13–47.

Bues, M. (1994). *Offene Systeme: Strategien, Konzepte und Techniken für das Informationsmanagement*. Springer, Berlin / Heidelberg.

Bußmann, H. (1983). *Lexikon der Sprachwissenschaft*. Kröners Taschenausgabe. Alfred Kröner Verlag, Stuttgart.

Carstensen, K.-U., Ebert, C., Ebert, C., Jekat, S., Klabunde, R. und Langer, H. (Herausgeber) (2010). *Computerlinguistik und Sprachtechnologie: Eine Einführung*. Spektrum Akademischer Verlag, Heidelberg, 3. Auflage.

Carter, S., Weerkamp, W. und Tsagkias, M. (2012). Microblog language identification: Overcoming the limitations of short, unedited and idiomatic text. *Language Resources and Evaluation*, 47(1):195–215.

Ceccato, M., Kiyavitskaya, N., Zeni, N., Mich, L. und Berry, D. M. (2004). Ambiguity Identification and Measurement in Natural Language Texts. Technischer Bericht DIT-04-111, University of Trento, Trento, Italien.

Chaimongkol, P., Aizawa, A. und Tateisi, Y. (2014). Corpus for Coreference Resolution on Scientific Papers. In Calzolari, N., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J. und Piperidis, S. (Herausgeber), *Proceedings of the 9th International Conference on LREC*, Reykjavik, Island. ELRA.

Chantree, F., Kilgarriff, A., De Roeck, A. und Willis, A. (2005). Disambiguating Coordinations Using Word Distribution Information. In *Proceedings of RANLP*, Borovets, Bulgarien.

Chantree, F., Nuseibeh, B., De Roeck, A. und Willis, A. (2006). Identifying Nocuous Ambiguities in Natural Language Requirements. In *Proceedings of the 14th IEEE RE*, Seiten 59–68, Minneapolis, MN, USA. IEEE.

Chantree, F., Willis, A., Kilgarriff, A. und De Roeck, A. (2007). Detecting Dangerous Coordination Ambiguities Using Word Distribution. In *RANLP IV: Selected papers from RANLP 2005*, Seiten 287–296. John Benjamins Publishing, Amsterdam, Niederlande / Philadelphia, PA, USA.

Charniak, E. (1997). Statistical Parsing with a Context-free Grammar and Word Statistics. In *Proceedings of the 24th National Conference on AAAI and 9th Conference on IAAI*, AAAI'97/IAAI'97, Seiten 598–603, Providence, RI, USA. AAAI Press.

Chen, D. und Manning, C. D. (2014). A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on EMNLP*, Seiten 740–750, Doha, Katar. ACL.

Choi, J. D. und McCallum, A. (2013). Transition-based Dependency Parsing with Selectional Branching. In *Proceedings of the 51st Annual Meeting of the ACL*, Seiten 1052–1062, Sofia, Bulgarien. ACL.

Choi, J. D., Tetreault, J. und Stent, A. (2015). It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. In *Proceedings of the 53rd Annual Meeting of the ACL and the 7th IJCNLP*, Seiten 387–396, Beijing, China. ACL.

Collins, M. und Brooks, J. (1995). Prepositional Phrase Attachment through a Backed-Off Model. In Yarowsky, D. und Church, K. W. (Herausgeber), *Proceedings of the 3rd Workshop on Very Large Corpora*, Seiten 27–38, Cambridge, MA, USA. ACL.

Collins, M. J. (1996). A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting on ACL*, ACL'96, Seiten 184–191, Stroudsburg, PA, USA. ACL.

Cook, T. (2002). *Mastering JSP*. SYBEX, Alameda, CA, USA.

Corley, C. und Mihalcea, R. (2005). Measuring the Semantic Similarity of Texts. In *Proceedings of the ACL Workshop on EMSEE*, EMSEE'05, Seiten 13–18, Stroudsburg, PA, USA. ACL.

Crockford, D. (2006). JSON: The Fat-Free Alternative to XML. `http://www.json.org/xml.html`. Zuletzt abgerufen am 02.12.2015.

Culotta, A., Wick, M., Hall, R. und McCallum, A. (2007). First-Order Probabilistic Models for Coreference Resolution. In *Proceedings of the HLT conference / Meeting of the NAACL*, Seiten 81–88, Rochester, NY, USA. ACL.

Davies, M. (2016). Word frequency data – Corpus of Contemporary American English. `http://www.wordfrequency.info/free.asp?s=y`. Zuletzt abgerufen am 08.09.2016.

Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A. und Theofanos, M. (1993). Identifying and Measuring Quality in a Software Requirements Specification. In *Proceedings of the 1st International Software Metrics Symposium*, Seiten 141–152, Baltimore, MD, USA. IEEE.

Decker, B., Ras, E., Rech, J., Jaubert, P. und Rieth, M. (2007). Wiki-Based Stakeholder Participation in Requirements Engineering. *IEEE Software*, 24(2):28–35.

Deeptimahanti, D. K. und Sanyal, R. (2009). An Innovative Approach for Generating Static UML Models from Natural Language Requirements. In Kim, T.-h., Fang, W.-C., Lee, C. und Arnett, K. P. (Herausgeber), *Advances in Software Engineering*, Band: 30, Seiten 147–163. Springer, Berlin / Heidelberg.

Deeptimahanti, D. K. und Sanyal, R. (2011). Semi-automatic Generation of UML Models from Natural Language Requirements. In *Proceedings of the 4th ISEC*, ISEC'11, Seiten 165–174, New York, NY, USA. ACM.

Denert, E. (2013). *Software-Engineering: Methodische Projektabwicklung*. Springer, Berlin / Heidelberg / New York, NY, USA. 1. korrigierter Nachdruck.

Dias Cardoso, P. M. und Roy, A. (2016). Language Identification for Social Media: Short Messages and Transliteration. In *Proceedings of the 25th International Conference Companion on WWW*, WWW'16 Companion, Seiten 611–614, Montreal, QC, Kanada. International WWW Conferences Steering Committee.

Dittmann, J. und Thieroff, R. (2009). *Richtiges Deutsch leicht gemacht*. Bertelsmann Wahrig. Wissenmedia, Gütersloh / München.

Doddington, G., Mitchell, A., Przybocki, M., Ramshaw, L., Strassel, S. und Weischedel, R. (2004). The Automatic Content Extraction (ACE) Program – Tasks, Data, and Evaluation. In *Proceedings of the 4th International LREC*, Seite 837–840, Lissabon, Portugal. ELRA.

Dollmann, M. (2016). Frag die Anwender: Extraktion und Klassifikation von funktionalen Anforderungen aus User-Generated-Content. Masterarbeit, Universität Paderborn, Paderborn.

Dollmann, M. und Geierhos, M. (2016). On- and Off-Topic Classification and Semantic Annotation of User-Generated Software Requirements. In *Proceedings of the Conference on EMNLP*, Austin, TX, USA. ACL.

Dönninghaus, S. (2005). *Vagheit der Sprache: Begriffsgeschichte und Funktionsbeschreibung anhand der tschechischen Wissenschaftssprache*. Slavistische Studien Bücher. Harrassowitz Verlag, Wiesbaden.

Drechsler, R., Soeken, M. und Wille, R. (2014). Automated and Quality-driven Requirements Engineering. In *Proceedings of the 2014 IEEE/ACM ICCAD*, ICCAD'14, Seiten 586–590, Piscataway, NJ, USA. IEEE.

Dudenredaktion (Herausgeber) (2016). *Duden – Deutsches Universalwörterbuch: Das umfassende Bedeutungswörterbuch der deutschen Gegenwartssprache*. Dudenredaktion, Berlin, 8. Auflage.

Dudenredaktion (2017a). Duden, Stichwort: Lernen. `http://www.duden.de/node/665539/revisions/1613414/view`. Zuletzt abgerufen am 04.03.2017.

Dudenredaktion (2017b). Duden, Stichwort: Standardsprache. `http://www.duden.de/node/679032/revisions/1165333/view`. Zuletzt abgerufen am 05.03.2017.

Dudenredaktion (2017c). Duden, Stichwort: Text. `http://www.duden.de/node/654612/revisions/1370114/view`. Zuletzt abgerufen am 03.02.2017.

Dunkel, J. und Holitschke, A. (2003). *Softwarearchitektur für die Praxis*. Xpert.press. Springer, Hannover / Berlin / Heidelberg.

Dunning, T. (1994). Statistical Identification of Language. Technischer Bericht MCCS 94-273, New Mexico State University, Las Cruces, NM, USA.

Durrett, G. und Klein, D. (2013). Easy Victories and Uphill Battles in Coreference Resolution. In *Proceedings of the Conference on EMNLP*, Seattle, WA, USA. ACL.

Dustdar, S., Gall, H. und Hauswirth, M. (2003). *Was ist Software-Architektur?*, Seiten 1–11. Springer, Berlin / Heidelberg.

Ernst, M. (2003). *Syntaktische Ambiguität: Eine sprachübergreifende Typisierung auf der Basis des Französischen und Spanischen*, Band: 261. *Europäische Hochschulschriften XXI*. Peter Lang, Würzburg / Frankfurt am Main. Zugl.: Dissertation an der Universität Würzburg, 2002.

España, S., Condori-Fernandez, N., Gonzalez, A. und Pastor, O. (2009). Evaluating the Completeness and Granularity of Functional Requirements Specifications: A Controlled Experiment. In *Proceedings of the 17th IEEE RE*, RE'14, Seiten 161–170, Atlanta, GA, USA. IEEE.

Essberger, J. (2012). *English Prepositions List – 150 Prepositions*. Englishclub.com, Cambridge, UK.

Fabbrini, F., Fusani, M., Gnesi, S. und Lami, G. (2000). Quality Evaluation of Software Requirements Specifications. In *Proceedings of the Conference of the Software and Internet Quality Week*, Seiten 1–18, San Francisco, CA, USA.

Fabbrini, F., Fusani, M., Gnesi, S. und Lami, G. (2001). The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool. In *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, Seiten 97–105, Greenbelt, MD, USA. IEEE.

Fagan, M. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211.

Fahney, R., Gartung, T., Glunde, J., Hoffmann, A. und Valentini, U. (2012). *Requirements Engineering und Projektmanagement*. Springer, Berlin / Heidelberg.

Fantechi, A. und Spinicci, E. (2005). A Content Analysis Technique for Inconsistency Detection in Software Requirements Documents. In *Proceedings of the WER 2005, VIII Workshop on Requirements Engineering*, Seiten 245–256, Porto, Portugal.

Femmer, H. (2013). Reviewing Natural Language Requirements with Requirements Smells – A Research Proposal. http://www4.in.tum.de/~femmer/works/idoese13.pdf. Zuletzt abgerufen am 27.08.2016.

Femmer, H., Fernández, D. M., Juergens, E., Klose, M., Zimmer, I. und Zimmer, J. (2014). Rapid Requirements Checks with Requirements Smells: Two Case Studies. In *Proceedings of the 1st International Workshop on RCoSE*, RCoSE'14, Seiten 10–19, New York, NY, USA. ACM.

Femmer, H., Fernández, D. M., Wagner, S. und Eder, S. (2016a). Rapid Quality Assurance with Requirements Smells. *Journal of Systems and Software*. In Press, Corrected Proof. Erreichbar unter: http://www4.in.tum.de/~femmer/works/2016-requirements_smells-jss.pdf. Zuletzt abgerufen am 27.08.2016.

Femmer, H., Hauptmann, B. und Widera, A. (2016b). Requirements-Smells: Automatische Unterstützung bei der Qualitätssicherung von Anforderungsdokumenten. *OBJEKTspektrum*, 2:14–19.

Ferlein, J. und Hartge, N. (2008). *Technische Dokumentation für internationale Märkte: Haftungsrechtliche Grundlagen - Sprache - Gestaltung - Redaktion und Übersetzung*. Expert Verlag, Renningen.

Ferrari, A., dell' Orletta, F., Spagnolo, G. O. und Gnesi, S. (2014). Measuring and Improving the Completeness of Natural Language Requirements. In Salinesi, C. und van de Weerd, I. (Herausgeber), *Requirements Engineering: Foundation for Software Quality*, Band: 8396. *LNCS*, Seiten 23–38. Springer, Essen.

Fettke, P. (2012). Enzyklopädie der Wirtschaftsinformatik, Stichwort: Objektorientierte Modellierung. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Hauptaktivitaten-der-Systementwicklung/Problemanalyse-/Objektorientierte-Modellierung. Zuletzt abgerufen am 28.07.2015.

Firesmith, D. (2007). Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them. *Journal of Object Technology*, 6(1):17–33.

Firesmith, D. G. (2005). Are Your Requirements Complete? *Journal of Object Technology*, 4(2):27–43.

Flati, T. und Navigli, R. (2014). Three Birds (in the LLOD Cloud) with One Stone: BabelNet, Babelfy and the Wikipedia Bitaxonomy. In *Proceedings of SEMANTiCS 2014*, Seiten 10–13, Leipzig.

Fowler, M., Beck, K., Brant, J., Opdyke, W. und Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Object Technology Series. Addison-Wesley, Westford, MA, USA.

Friedrich, F., Mendling, J. und Puhlmann, F. (2011). Process Model Generation from Natural Language Text. In *Proceedings of the 23rd CAiSE*, CAiSE'11, Seiten 482–496, Berlin / Heidelberg. Springer.

Fries, N. (1980). *Ambiguität und Vagheit: Einführung und kommentierte Bibliographie*, Band: 84. *Linguistische Arbeiten*. De Gruyter, Tübingen.

Gale, W. A., Church, K. W. und Yarowsky, D. (1992). One Sense Per Discourse. In *Proceedings of the Workshop on Speech and Natural Language*, HLT'91, Seiten 233–237, Stroudsburg, PA, USA. ACL.

Gausemeier, J., Czaja, A. M., Wiederkehr, O., Dumitrescu, R., Tschirner, C. und Steffen, D. (2013). Studie: Systems Engineering in der industriellen Praxis. *9. Paderborner Workshop: Entwurf mechatronischer Systeme*.

Geierhos, M. (2010). *BiographIE: Klassifikation und Extraktion karrierespezifischer Informationen*. Doktorarbeit, LMU München, München.

Geierhos, M. und Bäumer, F. S. (2016). How to Complete Customer Requirements: Using Concept Expansion for Requirement Refinement. In Métais, E., Meziane, F., Saraee, M., Sugumaran, V. und Vadera, S. (Herausgeber), *Proceedings of the 21st NLDB*, Manchester, UK. Springer.

Geierhos, M. und Bäumer, F. S. (2017). Guesswork? Resolving Vagueness in User-Generated Software Requirements. In Christiansen, H., Jiménez López, M. D., Loukanova, R. und Moss, L. (Herausgeber), *Partiality and Underspecification in Information, Languages, and Knowledge*, Kapitel 3, Seiten 65–107. Cambridge Scholars Publishing, Cambridge, UK.

Geierhos, M., Schulze, S. und Bäumer, F. S. (2015). What did you mean? Facing the Challenges of User-generated Software Requirements. In Loiseau, S., Filipe, J., Duval, B. und van den Herik, J. (Herausgeber), *Proceedings of the 7th ICAART*, Special Session on PUaNLP 2015, Seiten 277–283, Lissabon, Portugal. SCITEPRESS – Science and Technology Publications.

Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O. und Moreno, V. (2013). A Framework to Measure and Improve the Quality of Textual Requirements. *Requirements Engineering*, 18(1):25–41.

Geurts, P., Ernst, D. und Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Ghaddar, A. und Langlais, P. (2016). WikiCoref: An English Coreference-annotated Corpus of Wikipedia Articles. In Calzolari, N., Choukri, K., Declerck, T., Goggi, S., Grobelnik, M., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J. und Piperidis, S. (Herausgeber), *Proceedings of the 10th International Conference on LREC*, Paris, Frankreich. ELRA.

Ghazarian, A. (2009). A Case Study of Defect Introduction Mechanisms. In van Eck, P., Gordijn, J. und Wieringa, R. (Herausgeber), *Advanced Information Systems Engineering*, Band: 5565. *LNCS*, Seiten 156–170. Springer, Berlin / Heidelberg.

Gill, K. D., Raza, A., Zaidi, A. M. und Kiani, M. M. (2014). Semi-Automation for Ambiguity Resolution in Open Source Software Requirements. In *Proceedings of the 27th CCECE*, CCECE'14, Seiten 1–6, Toronto, ON, Kanada. IEEE.

Gillick, D. (2009). Sentence Boundary Detection and the Problem with the U.S. In *Proceedings of HLT: The 2009 Annual Conference of the NAACL*, NAACL-Short'09, Seiten 241–244, Stroudsburg, PA, USA. ACL.

Gleich, B., Creighton, O. und Kof, L. (2010). Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources. In Wieringa, R. und Persson, A. (Herausgeber), *Requirements Engineering: Foundation for Software Quality*, Band: 6182. *LNCS*, Seiten 218–232. Springer, Berlin / Heidelberg.

Goldberg, M. (1999). An Unsupervised Model for Statistically Determining Coordinate Phrase Attachment. In *Proceedings of the 37th Annual Meeting of the ACL*, ACL'99, Seiten 610–614, Stroudsburg, PA, USA. ACL.

Goldberg, Y. und Nivre, J. (2013). Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the ACL*, 1:403–414.

Grande, M. (2011). *100 Minuten für Anforderungsmanagement - Kompaktes Wissen nicht nur für Projektleiter und Entwickler*. Vieweg+Teubner Verlag / Springer Fachmedien, Wiesbaden.

Grechenig, T. (2010). *Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten*. Pearson Studium, München / Boston, MA, USA / Massachusetts, MA, USA.

Grefenstette, G. (1995). Comparing Two Language Identification Schemes. In *Proceedings of the 3rd JADT*, JADT'95, Seiten 263–268, Rom, Italien. Erreichbar unter: `http://www.uvm.edu/~pdodds/teaching/courses/2009-08UVM-300/docs/others/everything/grefenstette1995a.pdf`. Zuletzt abgerufen am 12.01.2017.

Grice, H. P. (1991). *Studies in the Way of Words*. Harvard University Press, Cambridge, UK / Massachusetts, MA, USA / London, UK.

Grishman, R. und Sundheim, B. (1996). Message Understanding Conference – 6: A Brief History. In *Proceedings of COLING'96*, Seiten 466–471, Kopenhagen, Dänemark. ACM.

Grosz, B. J., Joshi, A. K. und Weinstein, S. (1995). Centering: A Framework for Modeling the Local Coherence of Discourse. *Computional Linguistics*, 21(2):203–225.

Guha, A., Iyyer, M., Bouman, D. und Boyd-Graber, J. L. (2015). Removing the Training Wheels: A Coreference Dataset that Entertains Humans and Challenges Computers. In *The 2015 Conference of the NAACL: HLT*, Seiten 1108–1118, Denver, CO, USA. ACL.

Gumm, H.-P. und Sommer, M. (2012). *Einführung in die Informatik*. Oldenbourg Wissenschaftsverlag, München, 10. Auflage.

Haberfellner, R., Nagel, P., Becker, M., Bücher, A. und von Massow, H. (1994). Systemgestaltung. In Daenzer, W. F. und Huber, F. (Herausgeber), *Systems Engineering: Methoden und Praxis*. Verlag Industrielle Organisation Zürich, Zürich, Schweiz, 8. Auflage.

Haghighi, A. und Klein, D. (2009). Simple Coreference Resolution with Rich Syntactic and Semantic Features. In *Proceedings of the 2009 Conference on EMNLP*, Seiten 1152–1161, Singapur. ACL.

Hajič, J., Vidová-Hladká, B. und Pajas, P. (2001). The Prague Dependency Treebank: Annotation Structure and Support. In *Proceedings of the IRCS Workshop on Linguistic Databases*, Seiten 105–114, Philadelphia, PA, USA. University of Pennsylvania.

Hajičová, E., Abeillé, Hajič, J., Mírovský, J. und Urešová, Z. (2010). Treebank Annotation. In Indurkhya, N. und Damerau, F. J. (Herausgeber), *Handbook of Natural Language Processing*, Kapitel 8, Seiten 167–188. CRC Press, London, UK / New York, NY, USA u. a.

Hall, A. (1990). Seven Myths of Formal Methods. *IEEE Software*, 7(5):11–19.

Hall, A. und Chapman, R. (2002). Correctness by Construction: Developing a Commercial Secure System. *IEEE Software*, 19(1):18–25.

Hammer, N. und Bensmann, K. (2011). *Webdesign für Studium und Beruf: Webseiten planen, gestalten und umsetzen*. X.media.press. Springer, Berlin / Heidelberg, 2. Auflage.

Hammer, U. (2013). *Lexikon der Wirtschaftsinformatik*, Kapitel Adaptierbarkeit, Seite 6. Springer, 3. Auflage.

Hamp, B. und Feldweg, H. (1997). GermaNet - a Lexical-Semantic Net for German. In *Proceedings of the ACL workshop Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, Madrid, Spanien. ACL.

Harabagiu, S. M., Bunescu, R. C. und Maiorano, S. J. (2001). Text and Knowledge Mining for Coreference Resolution. In *Proceedings of the 2nd Meeting of the NAACL on Language Technologies*, NAACL'01, Seiten 1–8, Stroudsburg, PA, USA. ACL.

Hardt, D. (1997). An Empirical Approach to VP Ellipsis. *Computational Linguistics*, 23(4):525–541.

Henrich, V. und Hinrichs, E. (2010). GernEdiT - The GermaNet Editing Tool. In *Proceedings of the 7th LREC*, Seiten 2228–2235, Valletta, Malta. ELRA.

Hindle, D. und Rooth, M. (1993). Structural Ambiguity and Lexical Relations. *Computational Linguistics*, 19(1):103–120.

Hirst, G. und St-Onge, D. (1995). Lexical chains as representations of context for the detection and correction of malapropisms. In Fellbaum, C. (Herausgeber), *WordNet: An Electronic Lexical Database*, Seiten 306–332. MIT Press, Toronto, ON, Kanada.

Ho, T.-N., Chong, T. Y., Do, V. H., Pham, V. T. und Chng, E. S. (2016). Improving Efficiency of Sentence Boundary Detection by Feature Selection. In Nguyen, T. N., Trawiński, B., Fujita, H. und Hong, T.-P. (Herausgeber), *Proceedings of the 8th ACIIDS*, Seiten 594–603, Berlin / Heidelberg / Da Nang, Vietnam. Springer.

Hoffmann, D. W. (2013). *Software-Qualität*. eXamen.press. Springer, Karlsruhe / Berlin / Heidelberg, 2. Auflage.

Hoffmann, L. (Herausgeber) (2009). *Handbuch der deutschen Wortarten*. De Gruyter Lexikon Series. Walter de Gruyter, Dortmund / Berlin / New York, NY, USA.

Holtmann, J., Meyer, J. und von Detten, M. (2011). Automatic Validation and Correction of Formalized, Textual Requirements. In *Proceedings of the 4th ICSTW*, Seiten 486–495, Berlin. IEEE.

Honnibal, M., Goldberg, Y. und Johnson, M. (2013). A Non-Monotonic Arc-Eager Transition System for Dependency Parsing. In *Proceedings of the 7th CONLL*, Seiten 163–172, Sofia, Bulgarien. ACL.

Hood, C. und Wiebel, R. (2005). *Optimieren von Requirements Management & Engineering*. Springer, Berlin / Heidelberg u. a.

HSE (2003). Out of control: Why control systems go wrong and how to prevent failure. `http://automatie-pma.com/wp-content/uploads/2015/02/hsg238.pdf`. Zuletzt abgerufen am 18.01.2016.

Hsia, P., Davis, A. und Kung, D. (1993). Status Report: Requirements Engineering. *IEEE Software*, 10(6):75–79.

Huang, L., Fayong, S. und Guo, Y. (2012). Structured Perceptron with Inexact Search. In *Proceedings of the 2012 Conference of the NAACL: HLT*, NAACL HLT'12, Seiten 142–151, Stroudsburg, PA, USA. ACL.

Huang, Z., Zeng, G., Xu, W. und Celikyilmaz, A. (2009). Accurate Semantic Class Classifier for Coreference Resolution. In *Proceedings of the 2009 Conference on EMNLP*, Band: 3. *EMNLP'09*, Seiten 1232–1240, Stroudsburg, PA, USA. ACL.

Huertas, C. und Juárez-Ramírez, R. (2012). NLARE, a Natural Language Processing Tool for Automatic Requirements Evaluation. In *Proceedings of the CUBE International Information Technology Conference*, CUBE'12, Seiten 371–378, New York, NY, USA. ACM.

Huertas, C. und Juárez-Ramírez, R. (2013). Towards assessing the quality of functional requirements using english/spanish controlled languages and context free grammar. In *Proceedings of the 3rd International Conference on DICTAP*, Seiten 234–241, Ostrava, Tschechische Republik. SDIWC.

Huma, Z., Gerth, C., Engels, G. und Juwig, O. (2012). A UML-based Rich Service Description Language for Automatic Service Discovery of Heterogeneous Service Partners. In Kirikova, M. und Stirna, J. (Herausgeber), *Proceedings of the CAiSE'12 Forum*, Band: 855, Seiten 90–97, Gdansk, Polen. CEUR-WS.org. Erreichbar unter: `http://ceur-ws.org/Vol-855/paper11.pdf`. Zuletzt abgerufen am 12.01.2017.

Husain, S. und Beg, R. (2015). Advances in Ambiguity less NL SRS: A review. In *Proceedings of ICETECH 2015*, Seiten 221–225, Coimbatore, TN, Indien. IEEE.

Hußmann, H. (1993). Zur formalen Beschreibung der funktionalen Anforderungen an ein Informationssystem. Technischer bericht, Institut für Informatik, Technische Universität München. TUM-I9332.

IEEE (1991). *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. IEEE, New York, NY, USA.

IEEE (1998). *IEEE Std 830-1998 - Recommended Practice for Software Requirements Specifications*. IEEE, New York, NY, USA.

IEEE (2011). *ISO/IEC/IEEE 29148 – Systems and software engineering – Life cycle processes – Requirements engineering*. IEEE, New York, NY, USA. ISO/IEC/IEEE 29148:2011(E).

Jiang, J. J. und Conrath, D. W. (1997). Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In *Proceedings of ROCLING 1997*, Taipei, Taiwan. ACL.

Jungmann, A. (2016). *Towards On-The-Fly Image Processing*. Dissertation, Universität Paderborn, Paderborn.

Jurafsky, D. und Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2. Auflage.

Kadlec, T. und Fröhlich, S. (2013). *Praxiswissen Responsive Webdesign*. O'Reillys Basics. O'Reilly Verlag, Köln, 1. Auflage.

Kaiya, H. und Saeki, M. (2005). Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach. In *Proceedings of the 5th QSIC*, QSIC'05, Seiten 223–230, Melbourn, VIC, Australien. IEEE.

Kaiya, H. und Saeki, M. (2006). Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In *Proceedings of the 14th IEEE RE*, RE '06, Seiten 189–198, Minneapolis, MN, USA. IEEE.

Kalenborn, A. (2014). *Angebotserstellung und Planung von Internet-Projekten: Die werkzeugbasierte „Modeling by Example"-Methode*. Springer Fachmedien, Wiesbaden.

Kamata, M. I. und Tamai, T. (2007). How Does Requirements Quality Relate to Project Success or Failure? In *Proceedings of the 15th IEEE RE*, Seiten 69–78, Delhi, DL, Indien. IEEE.

Kamsties, E. (2005). Understanding Ambiguity in Requirements Engineering. In Aurum, A. und Wohlin, C. (Herausgeber), *Engineering and Managing Software Requirements*, Seiten 245–266. Springer, Berlin / Heidelberg.

Kamsties, E., Berry, D. M. und Paech, B. (2001). Detecting Ambiguities in Requirements Documents Using Inspections. In *Proceedings of the 1st WISE*, WISE'01, Seiten 68–80, Paris, Frankreich.

Kamsties, E. und Paech, B. (2000). Taming Ambiguity in Natural Language Requirements. In *Proceedings of the 13th ICSOFT-EA*, Seiten 1–8, Paris, Frankreich.

Kim, J.-D., Ohta, T., Tateisi, Y. und Tsujii, J. (2003). GENIA corpus – A semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(1):i180–i182.

Kipper-Schuler, K. (2005). *VerbNet: a broad-coverage, comprehensive verb lexicon.* Doktorarbeit, Universiy of Pennsylvania, Philadelphia, PA, USA. Erreichbar unter: `http://verbs.colorado.edu/~kipper/Papers/dissertation.pdf`. Zuletzt abgerufen am 03.06.2016.

Kiss, T. und Strunk, J. (2006). Unsupervised Multilingual Sentence Boundary Detection. *Computional Linguistics*, 32(4):485–525.

Kiyavitskaya, N., Zeni, N., Mich, L. und Berry, D. M. (2008). Requirements for Tools for Ambiguity Identification and Measurement in Natural Language Requirements Specifications. *Requirements Engineering*, 13(3):207–239.

Klein, D. und Manning, C. D. (2003). Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting on ACL - Volume 1*, ACL'03, Seiten 423–430, Stroudsburg, PA, USA. ACL.

Klose, M. und Wrigley, D. (2014). *Einführung in Apache Solr.* O'Reilly Verlag, Köln, 1. Auflage.

Kluck, N. (2014). *Der Wert der Vagheit*, Band: 5. *Linguistics & Philosophy.* De Gruyter, Münster / Berlin.

Knott, D. (2016). *Mobile App Testing: Praxisleitfaden für Softwaretester und Entwickler mobiler Anwendungen.* dpunkt.verlag, Heidelberg / Paderborn, 1. Auflage.

Kobdani, H., Schütze, H., Schiehlen, M. und Kamp, H. (2011). Bootstrapping Coreference Resolution Using Word Associations. In *Proceedings of the 49th Annual Meeting of the ACL: HLT*, Band: 1. *HLT'11*, Seiten 783–792, Stroudsburg, PA, USA. ACL.

Körner, S. J. (2014). *RECAA - Werkzeugunterstützung in der Anforderungserhebung.* Doktorarbeit, Karlsruher Institut für Technologie, Karlsruhe. KIT Scientific Publishing.

Körner, S. J. und Brumm, T. (2010). Natural Language Specification Improvement with Ontologies. *International Journal of Semantic Computing*, 03(04):445–470.

Krüger, G. und Hansen, H. (2014). *Java Programmierung – Das Handbuch zu Java 8*. O'Reilly Verlag, Köln, 8. Auflage.

Krüger, G. und Stark, T. (2009). *Handbuch der Java Programmierung – Standard Edition Version 6*. Addison-Wesley, München, 5. Auflage.

Kürschner, W. (2008). *Grammatisches Kompendium: Systematisches Verzeichnis grammatischer Grundbegriffe*. Francke Verlag, Tübingen, 6. Auflage. Aktualisierte Auflage.

Kurth, H. (1991). Formale Spezifikation und Verifikation - Ein Überblick. In Pfitzmann, A. und Raubold, E. (Herausgeber), *VIS '91 Verläßliche Informationssysteme*, Band: 271. *Informatik-Fachberichte*, Seiten 45–66. Springer, Berlin / Heidelberg.

Laitenberger, O. und DeBaud, J.-M. (2000). An encompassing life cycle centric survey of software inspection. *Journal of Systems and Software*, 50(1):5 – 31.

Lami, G. (2005). QuARS: A Tool for Analyzing Requirements. Technischer Bericht ESC-TR-2005-014, Carnegie Mellon University.

Landhäußer, M. (2016). *Eine Architektur für Programmsynthese aus natürlicher Sprache*. Dissertation, Karlsruher Institut für Technologie, Karlsruhe. KIT Scientific Publishing.

Landhäußer, M., Körner, S. J., Keim, J., Tichy, W. F. und Krisch, J. (2015). DeNom: A Tool to Find Problematic Nominalizations using NLP. In *Proceedings of the 2nd International Workshop on AIRE*, Seiten 9–16, Ottawa, ON, Kanada. IEEE.

Langer, H., Mehl, S. und Volk, M. (1997). Hybride NLP-Systeme und das Problem der PP-Anbindung. In *Hybride konnektionistische, statistische und symbolische Ansätze zur Verarbeitung natürlicher Sprache*, Saarbrücken / Freiburg. Workshop auf der 21. Deutschen Jahrestagung für Künstliche Intelligenz.

Langer, S. (2002). Grenzen der Sprachenidentifizierung. In *Tagungsband KONVENS 2002*, Seiten 99–106, Saarbrücken. DFKI.

Laparra, E. und Rigau, G. (2013). ImpAr: A Deterministic Algorithm for Implicit Semantic Role Labelling. In *Proceedings of the 51st Annual Meeting of the ACL*, Seiten 1180–1189, Sofia, Bulgarien. ACL.

Lapata, M. und Keller, F. (2005). Web-based models for natural language processing. *ACM Transactions on Speech and Language Processing*, 2(1).

Laplante, P. A. (2007). *What Every Engineer Should Know about Software Engineering*. CRC Press, London, UK / New York, NY, USA.

Laplante, P. A. (2013). *Requirements Engineering for Software and Systems*. Applied Software Engineering Series. Auerbach Publications, Boca Raton, FL, USA, 2. Auflage.

Lappin, S. und Leass, H. J. (1994). An Algorithm for Pronominal Anaphora Resolution. *Computional Linguistics*, 20(4):535–561.

Laurent, P. und Cleland-Huang, J. (2009). Lessons Learned from Open Source Projects for Facilitating Online Requirements Processes. In Glinz, M. und Heymans, P. (Herausgeber), *Proceedings of the 15th REFSQ*, Band: 5512, Seiten 240–255. Springer, Berlin / Heidelberg.

Leacock, C. und Chodorow, M. (1998). Combining Local Context and WordNet Similarity for Word Sense Identification. *WordNet: An electronic lexical database*, 49(2):265–283.

Lee, H., Chang, A., Peirsman, Y., Chambers, N., Surdeanu, M. und Jurafsky, D. (2013). Deterministic Coreference Resolution Based on Entity-centric, Precision-ranked Rules. *Computational Linguistics*, 39(4):885–916.

Lee, H., Peirsman, Y., Chang, A., Chambers, N., Surdeanu, M. und Jurafsky, D. (2011). Stanford's Multi-pass Sieve Coreference Resolution System at the CONLL-2011 Shared Task. In *Proceedings of the 15th Conference on CONLL: Shared Task*, CONLL Shared Task '11, Seiten 28–34, Stroudsburg, PA, USA. ACL.

Lehmann, C. (2013). Semasiologie der paradigmatischen lexikalischen Relationen. `http://www.christianlehmann.eu/ling/lg_system/sem/semasiolog_lexikal_relation.php`. Zuletzt abgerufen am 29.10.2015.

Lehrndorfer, A. (1996). *Kontrolliertes Deutsch: linguistische und sprachpsychologische Leitlinien für eine (maschinell) kontrollierte Sprache in der technischen Dokumentation.* Tübinger Beiträge zur Linguistik. Gunter Narr Verlag, Tübingen.

Lei, T., Xin, Y., Zhang, Y., Barzilay, R. und Jaakkola, T. (2014). Low-Rank Tensors for Scoring Dependency Structures. In *Proceedings of the 52nd Annual Meeting of the ACL*, Seiten 1381–1391, Baltimore, MD, USA. ACL.

Lewandowski, T. (1994). *Linguistisches Wörterbuch. Bd. 2. [I - R].* Quelle und Meyer, Heidelberg / Wiesbaden, 6. Auflage.

Lin, D. (1998). An Information-Theoretic Definition of Similarity. In *Proceedings of the 15th ICML*, Seiten 296–304, San Francisco, CA, USA. Morgan Kaufmann Publishers.

Lobin, H. und Heringer, H. J. (2010). *Computerlinguistik und Texttechnologie.* UTB 3282. Wilhelm Fink Verlag, Paderborn.

Löbner, S. (2003). *Semantik: eine Einführung.* De Gruyter Studienbuch. Walter de Gruyter, Berlin / New York, NY, USA.

Lopes Margarido, I., Faria, J. P., Vidal, R. M. und Vieira, M. (2011). Classification of Defect Types in Requirements Specifications: Literature Review, Proposal and Assessment. In *Proceedings of the 6th CISTI*, Seiten 1–6, Chaves, Portugal. IEEE.

López, R. und Pardo, T. A. S. (2015). Experiments on Sentence Boundary Detection in User-Generated Web Content. In Gelbukh, A. (Herausgeber), *Proceedings of the 16th CICLing*, Seiten 227–237, Kairo, Ägypten / Cham, Schweiz. Springer.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M. und Brinkkemper, S. (2016). Improving agile requirements: the Quality User Story framework and tool. *Requirements Engineering*, 21(3):383–403.

Lui, M. und Baldwin, T. (2014). Accurate Language Identification of Twitter Messages. In *Proceedings of the 5th Workshop on LASM*, Seiten 17–25, Göteborg, Schweden. ACL.

Maamouri, M., Bies, A., Buckwalter, T. und Mekki, W. (2004). The Penn Arabic Treebank: Building a Large-Scale Annotated Arabic Corpus. In *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*, Seiten 102–109, Kairo, Ägypten.

Magnini, B., Negri, M., Prevete, R. und Tanev, H. (2002). A WordNet-based Approach to Named Entities Recognition. In *Proceedings of the 2002 Workshop on SEMANET*, Band: 11. *SEMANET'02*, Seiten 1–7, Stroudsburg, PA, USA. ACL.

Manning, C. D. und Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, London, UK u. a.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J. und McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *ACL System Demonstrations*, Seiten 55–60, Baltimore, MD, USA. ACL.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K. und Schasberger, B. (1994). The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on HLT*, HLT'94, Seiten 114–119, Stroudsburg, PA, USA. ACL.

Marcus, M. P., Marcinkiewicz, M. A. und Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Markert, K. und Nissim, M. (2005). Comparing Knowledge Sources for Nominal Anaphora Resolution. *Computational Linguistics*, 31(3):367–402.

Martins, A. F. T., Almeida, M. B. und Smith, N. A. (2013). Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the Annual Meeting of the ACL*, Seiten 617–622, Sofia, Bulgarien. ACL.

Massey, A. K., Rutledge, R. L., Anton, A. I. und Swire, P. P. (2014). Identifying and Classifying Ambiguity for Regulatory Requirements. In *Proceedings of the 22nd International Requirements Engineering Conference*, Seiten 83–92, Karlskrona, Schweden. IEEE.

Matsuoka, J. und Lepage, Y. (2011). Ambiguity Spotting using WordNet Semantic Similarity in Support to Recommended Practice for Software Requirements Specifications. In *Proceedings of the 7th International Conference on NLP-KE*, Seiten 479–484, Tokushima, Japan. IEEE.

McCarthy, D., Koeling, R., Weeds, J. und Carroll, J. (2004). Finding Predominant Word Senses in Untagged Text. In *Proceedings of the 42nd Annual Meeting of the ACL*, Seite 280–287, Barcelona, Spanien. ACL.

McLauchlan, M. (2004). Thesauruses for Prepositional Phrase Attachment. In *Proceedings of the CONLL 2004*, Seiten 73–80, Boston, MA, USA. ACL.

McShane, M. und Babkin, P. (2015). Automatic Ellipsis Resolution: Recovering Covert Information from Text. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, Seiten 572–578, Austin, TX, USA. AAAI, AAAI Press.

McShane, M. und Babkin, P. (2016). Detection and Resolution of Verb Phrase Ellipsis. In *Linguistic Issues In Language Technology*, Band: 13. CSLI Publications.

Mehl, S., Langer, H. und Volk, M. (1998). Statistische Verfahren zur Zuordnung von Präpositionalphrasen. In *Proceedings of the Konvens '98*, Bonn. Peter Lang.

Mehler, A. und Lobin, H. (2004). *Automatische Textanalyse: Systeme und Methoden zur Annotation und Analyse natürlichsprachlicher Texte*. VS Verlag für Sozialwissenschaften, Wiesbaden, 1. Auflage.

Menzel, I., Mueller, M., Gross, A. und Doerr, J. (2010). An Experimental Comparison Regarding the Completeness of Functional Requirements Specifications. In *Proceedings of the 18th IEEE RE*, Seiten 15–24, Sydney, NSW, Australien. IEEE.

Mihalcea, R. (2003). The Role of Non-Ambiguous Words in Natural Language Disambiguation. In *Proceedings of the RANLP 2003*, Borovets, Bulgarien. Erreichbar unter: `https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.ranlp03.pdf`. Zuletzt abgerufen am 07.08.2016.

Mihalcea, R. (2010). Word Sense Disambiguation. In Sammut, C. und Webb, G. I. (Herausgeber), *Encyclopedia of Machine Learning*, Seiten 1027–1030. Springer, Boston, MA, USA.

Miller, G. A. (1995). WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41.

Mitkov, R. (1998). Robust pronoun resolution with limited knowledge. In *Proceedings of the 17th COLING*, COLING'98, Montreal, QC, Kanada. ACL.

Mitkov, R. (1999). *Anaphora Resolution: The State of the Art*. Research report. University of Wolverhampton, Wolverhampton, UK. Research Group in Computational Linguistics and Language Engineering.

Mitkov, R. (2014). *Anaphora Resolution*. Studies in Language and Linguistics. Taylor & Francis, New York, NY, USA. Neuauflage der 2002 erstmals publizierten Veröffentlichung.

Moens, M.-F., Li, J. und Chua, T.-S. (Herausgeber) (2014). *Mining User Generated Content*. CRC Press, Leuven, Belgien / Beijing, China / Singapur.

Monrose, F. und Rubin, A. (1997). Authentication via Keystroke Dynamics. In *Proceedings of the 4th ACM Conference on CCS*, CCS'97, Seiten 48–56, New York, NY, USA. ACM.

Montoyo, A., Suarez, A., Rigau, G. und Palomar, M. (2005). Combining Knowledge- and Corpus-based Word-Sense-Disambiguation Methods. In *Journal of Artificial Intelligence Research*, Band: 23. AAAI.

Moro, A., Cecconi, F. und Navigli, R. (2014a). Multilingual Word Sense Disambiguation and Entity Linking for Everybody. In *Proceedings of the 13th ISWC*, Seiten 25–28, Riva del Garda, Italien. Springer.

Moro, A., Raganato, A. und Navigli, R. (2014b). Entity Linking meets Word Sense Disambiguation: a Unified Approach. *Transactions of the ACL*, 2:231–244.

Morris, M. (2011). Sentence Tutorial – What is Sentence Detection? `http://alias-i.com/lingpipe/demos/tutorial/sentences/read-me.html`. Zuletzt abgerufen am 28.04.2016.

Nadh, K. und Huyck, C. (2009). Prepositional Phrase Attachment Ambiguity Resolution Using Semantic Hierarchies. In *The 9th International Conference on Artificial Intelligence and Applications*, Innsbruck, Österreich. ACTA Press.

Nakov, P. und Hearst, M. (2005). Using the Web As an Implicit Training Set: Application to Structural Ambiguity Resolution. In *Proceedings of the Conference on HLT and EMNLP*, HLT'05, Seiten 835–842, Stroudsburg, PA, USA. ACL.

Naumann, S. (2003). XML als Beschreibungssprache syntaktisch-annotierter Korpora. In Seewald-Heeg, U. (Herausgeber), *Sprachtechnologie für die multilinguale Kommunikation – Textproduktion, Recherche, Übersetzung, Lokalisierung – Beiträge der GLDV-Frühjahrstagung 2003*, Seiten 12–25. Gardez! Verlag.

Navigli, R. (2009). Word Sense Disambiguation: A Survey. In *ACM Computing Surveys*, Band: 41. ACM, New York, NY, USA.

Navigli, R. und Ponzetto, S. P. (2012a). BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. In *Artificial Intelligence*, Band: 193, Seiten 217–250. Elsevier, Essex, UK.

Navigli, R. und Ponzetto, S. P. (2012b). Joining Forces Pays Off: Multilingual Joint Word Sense Disambiguation. In *Proceedings of the 2012 Joint Conference on EMNLP and CONLL*, Seiten 1399–1410, Jeju, Korea. ACL.

Ng, V. (2007). Semantic Class Induction and Coreference Resolution. In *Proceedings of the 45th Annual Meeting of the ACL*, Seiten 536–543, Prag, Tschechische Republik. ACL.

Nielsen, J. und Loranger, H. (2006). *Web Usability*. Addison-Wesley, München, 1. Auflage.

Nigam, A., Arya, N., Nigam, B. und Jain, D. (2012). Tool for Automatic Discovery of Ambiguity in Requirements. *International Journal of Computer Science Issues*, 9(2):350–356.

O'Connor, B. und Heilman, M. (2013). ARKref: a rule-based coreference resolution system. *ArXiv e-prints*, Seiten 1–10. Erreichbar unter: `http://arxiv.org/abs/1310.1975`. Zuletzt abgerufen am 06.04.2016.

Oepen, S., Flickinger, D., Toutanova, K. und Manning, C. D. (2004). LinGO Redwoods – A Rich and Dynamic Treebank for HPSG. *Research on Language and Computation*, 2(4):575–596.

Okumura, A. und Muraki, K. (1994). Symmetric Pattern Matching Analysis for English Coordinate Structures. In *Proceedings of the 4th Conference on ANLP*, ANLC'94, Seiten 41–46, Stuttgart / Stroudsburg, PA, USA. ACL.

Osborne, M. und MacNish, C. K. (1996). Processing Natural Language Software Requirement Specifications. In *Proceedings of the 2nd International Conference on Requirements Engineering*, Seiten 229–236, Colorado Springs, CO, USA. IEEE.

Palmer, M., Gildea, D. und Kingsbury, P. (2005). The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1):71–106.

Pantel, P. und Lin, D. (2000). An Unsupervised Approach to Prepositional Phrase Attachment Using Contextually Similar Words. In *Proceedings of the 38th Annual Meeting on ACL*, ACL'00, Seiten 101–108, Stroudsburg, PA, USA. ACL.

Partsch, H. A. (2010). *Requirements-Engineering systematisch: Modellbildung für softwaregestützte Systeme*. eXamen.press. Springer, Berlin / Heidelberg.

Passonneau, R. J., Ramelson, T. und Xie, B. (2015). Named Entity Recognition from Financial Press Releases. In Fred, A., Dietz, G. J. L., Aveiro, D., Liu, K. und Filipe, J. (Herausgeber), *Proceedings of the 6th IC3K*, Seiten 240–254. Springer, Rom, Italien.

Patwardhan, S., Banerjee, S. und Pedersen, T. (2003). Using Measures of Semantic Relatedness for Word Sense Disambiguation. In *Proceedings of the 4th CICLing*, Seiten 241–257, Mexico City, Mexiko. ACL.

Pekar, V., Felderer, M. und Breu, R. (2014). Improvement Methods for Software Requirement Specifications: A Mapping Study. In *Proceedings of the 9th QUATIC*, Seiten 242–245, Guimarães, Portugal. IEEE.

Petermann, K. (2014). *Verbale und nonverbale Vagheit in englisch- und deutschsprachigen Interviews*, Band: 118. *Forum für Fachsprachen-Forschung*. Frank & Timme, Berlin.

Pfeifer, W. (o. D.). Etymologisches Wörterbuch (nach Pfeifer): Ambiguität. `http://www.dwds.de/?qu=Ambiguit%C3%A4t`. Zuletzt abgerufen am 22.10.2015.

Philippo, E. J., Heijstek, W., Kruiswijk, B., Chaudron, M. R. und Berry, D. M. (2013). Requirement Ambiguity Not as Important as Expected – Results of an Empirical Evaluation. In Doerr, J. und Opdahl, A. L. (Herausgeber), *Requirements Engineering: Foundation for Software Quality*, Band: 7830. *LNCS*, Seiten 65–79. Springer, Berlin / Heidelberg.

Pinkal, M. (1985). *Logik und Lexikon: Die Semantik des Unbestimmten*. Grundlagen der Kommunikation. Walter de Gruyter, Berlin / New York, NY, USA.

Pinkal, M. (1991). *Semantik / Semantics: Ein internationales Handbuch der zeitgenössischen Forschung.*, Band: 6. *Handbücher zur Sprach- und Kommunikationswissenschaft*, Kapitel Vagheit und Ambiguität, Seiten 250–270. Walter de Gruyter, Berlin / New York, NY, USA.

Pinto, A., Oliveira, H. G. und Alves, A. O. (2016). Comparing the Performance of Different NLP Toolkits in Formal and Social Media Text. In Mernik, M., Leal, J. P. und Oliveira, H. G. (Herausgeber), *Proceedings of the 5th Symposium on Languages, Applications and Technologies*, SLATE'16, Maribor, Slowenien. Dagstuhl Publishing.

Platenius, M. C. (2013). Fuzzy Service Matching in On-the-fly Computing. In *Proceedings of the 2013 9th Joint Meeting on FSE*, ESEC/FSE'13, Seiten 715–718, New York, NY, USA. ACM.

Platenius, M. C. (2016). *Unscharfes Matching von umfassenden Service-Spezifikationen*. Dissertation, Universität Paderborn, Paderborn.

Platenius, M. C., Arifulina, S., Petrlic, R. und Schäfer, W. (2015). Matching of Incomplete Service Specifications Exemplified by Privacy Policy Matching. In Ortiz, G. und Tran, C. (Herausgeber), *Advances in Service-Oriented and Cloud Computing*, Band: 508. *Communications in Computer and Information Science*, Seiten 6–17. Springer, Cham, Schweiz.

Platenius, M. C., Josifovska, K., van Rooijen, L., Arifulina, S., Becker, M., Engels, G. und Schäfer, W. (2016). An Overview of Service Specification Language and Matching in On-The-Fly Computing (v0.3). Technischer Bericht Tr-ri-16-349, Software Engineering Group, Heinz Nixdorf Institut, Universität Paderborn.

Poesio, M. und Artstein, R. (2005). The Reliability of Anaphoric Annotation, Reconsidered: Taking Ambiguity into Account. In *Proceedings of the Workshop on Frontiers in Corpus Annotations II: Pie in the Sky*, CorpusAnno'05, Seiten 76–83, Stroudsburg, PA, USA. ACL.

Pohl, K. (2007). *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. dpunkt.verlag, Heidelberg.

Pohl, K. (2008). *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. dpunkt.verlag, Heidelberg, 2. Auflage.

Pohl, K. und Rupp, C. (2015). *Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*. dpunkt.verlag, Heidelberg, 4. Auflage.

Ponzetto, S. P. und Strube, M. (2006). Exploiting Semantic Role Labeling, WordNet and Wikipedia for Coreference Resolution. In *Proceedings of the Main Conference on HLT / Conference of the NAACL*, HLT-NAACL'06, Seiten 192–199, Stroudsburg, PA, USA. ACL.

Popescu, D., Rugaber, S., Medvidovic, N. und Berry, D. M. (2008). Reducing Ambiguities in Requirements Specifications Via Automatically Created Object-Oriented Models. In Paech, B. und Martell, C. (Herausgeber), *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs: Revised Selected Papers*, Seiten 103–124. Springer, Berlin / Heidelberg / Monterey, CA, USA.

Prokofyev, R., Tonon, A., Luggen, M., Vouilloz, L., Difallah, D. E. und Cudré-Mauroux, P. (2015). SANAPHOR: Ontology-Based Coreference Resolution. In Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K. und Staab, S. (Herausgeber), *Proceedings of the 14th ISWC*, Seiten 458–473. Springer, Bethlehem, PA, USA.

Propbank (2010). Frameset – Predicate: send. `http://verbs.colorado.edu/propbank/framesets-english/send-v.html`. Zuletzt abgerufen am 24.12.2015.

Raganato, A., Camacho-Collados, J. und Navigli, R. (2017). Word Sense Disambiguation: A Unified Evaluation Framework and Empirical Comparison. In *Proceedings of the 15th Conference of the EACL: Volume 1, Long Papers*, Seiten 99–110, Valencia, Spanien. ACL, ACL.

Raghunathan, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D. und Manning, C. (2010). A Multi-pass Sieve for Coreference Resolution. In *Proceedings of the 2010 Conference on EMNLP*, EMNLP'10, Seiten 492–501, Stroudsburg, PA, USA. ACL.

Rahman, A. und Ng, V. (2009). Supervised Models for Coreference Resolution. In *Proceedings of the 2009 Conference on EMNLP*, Seiten 968–977, Singapur. ACL.

Rahman, A. und Ng, V. (2011). Coreference Resolution with World Knowledge. In *Proceedings of the 49th Annual Meeting of the ACL*, Band: 1. *HLT'11*, Seiten 814–824, Stroudsburg, PA, USA. ACL.

Rasooli, M. S. und Tetreault, J. (2015). Yara Parser: A Fast and Accurate Dependency Parser. In *CoRR*, Band: abs/1503.06733, Seiten 1–14. Erreichbar unter: `http://arxiv.org/pdf/1503.06733v1.pdf`. Zuletzt abgerufen am 19.05.2016.

Ratnaparkhi, A. (1998). Statistical Models for Unsupervised Prepositional Phrase Attachment. In *Proceedings of the 17th COLING - Volume 2*, COLING'98, Seiten 1079–1085, Stroudsburg, PA, USA. ACL.

Ratnaparkhi, A., Reynar, J. und Roukos, S. (1994). A Maximum Entropy Model for Prepositional Phrase Attachment. In *Proceedings of the Workshop on HLT*, HLT'94, Seiten 250–255, Stroudsburg, PA, USA. ACL.

Read, J., Dridan, R., Oepen, S. und Solberg, L. J. (2012a). Sentence Boundary Detection: A Long Solved Problem? In *Proceedings of COLING 2012: Posters*, Seiten 985–994, Mumbai, MH, Indien. ACL. Erreichbar unter: `http://www.aclweb.org/anthology/C12-2096`. Zuletzt abgerufen am 24.04.2016.

Read, J., Flickinger, D., Dridan, R., Oepen, S. und Øvrelid, L. (2012b). The WeSearch Corpus, Treebank, and Treecache - A Comprehensive Sample of User-Generated Content. In *Proceedings of the 8th International Conference on LREC*, Seiten 1829–1835, Istanbul, Türkei. ELRA.

Recasens, M., de Marneffe, M.-C. und Potts, C. (2013). The Life and Death of Discourse Entities: Identifying Singleton Mentions. In *Proceedings of the 2013 Conference of the NAACL: HLT*, Seiten 627–633, Atlanta, GA, USA. ACL.

Recasens Potau, M. (2010). *Coreference: Theory, Annotation, Resolution and Evaluation*. Doktorarbeit, University of Barcelona.

Reconcile Development Team (2011). Reconcile – Coreference Resolution Engine. `https://www.cs.utah.edu/nlp/reconcile/`. Zuletzt abgerufen am 16.05.2016.

Reese, R. M. (2015). *Natural Language Processing with Java*. Community Experience Distilled. PACKT Publishing, Birmingham, UK / Mumbai, MH, Indien.

Řehůřek, R. und Kolkus, M. (2009). Language Identification on the Web: Extending the Dictionary Method. In Gelbukh, A. (Herausgeber), *Proceedings of the 10th CICLing*, Seiten 357–368, Berlin / Heidelberg / Mexico City, Mexiko. Springer.

Reisner, S. (2011). Wege zu realistischen Performance-Tests. `http://www.computerwoche.de/a/wege,1232121`. Computerwoche. Zuletzt abgerufen am 23.02.2017.

Resnik, P. (1995). Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In *Proceedings of the 14th IJCAI - Volume 1*, IJCAI'95, Montreal, QC, Kanada. ACM.

Resnik, P. (1999). Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. *Journal of Artificial Intelligence Research*, 11:95–130.

Ristić, I. (2014). *Bulletproof SSL and TLS*. Feisty Duck, London, UK, 2. Auflage.

Robertson, S. und Robertson, J. (2006). *Mastering the Requirements Process*. Pearson Education, 2. Auflage.

Robertson, S. und Robertson, J. (2012). *Mastering the Requirements Process: Getting Requirements Right*. Pearson Education, 3. Auflage.

Rojas, A. B. und Sliesarieva, G. B. (2010). Automated Detection of Language Issues Affecting Accuracy, Ambiguity and Verifiability in Software Requirements Written in Natural Language. In *Proceedings of the NAACL HLT 2010 YIWCALA*, YIWCALA'10, Seiten 100–108, Stroudsburg, PA, USA. ACL.

Roth, M., Diamantopoulos, T., Klein, E. und Symeonidis, A. (2014). Software Requirements: A new Domain for Semantic Parsers. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, Seiten 50–54, Baltimore, MD, USA. ACL, ACL.

Rudrapal, D., Jamatia, A., Chakma, K., Das, A. und Gambäck, B. (2015). Sentence Boundary Detection for Social Media Text. In Sharma, D. M., Sangal, R. und Sherly, E. (Herausgeber), *Proceedings of the 12th ICON*, ICON'15, Seiten 1–7. Erreichbar unter: `http://ltrc.iiit.ac.in/icon2015/icon2015_proceedings/PDF/13_rp.pdf`. Zuletzt abgerufen am 28.04.2016.

Rupp, C. (2007). *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis*. Carl Hanser Verlag, München / Wien, Österreich, 4. Auflage.

Rupp, C. (2012). *Requirements Engineering: Ein Überblick*. dpunkt.verlag, Nürnberg / Heidelberg, 3. Auflage.

Rupp, C. (2013). *Systemanalyse kompakt*. IT kompakt. Springer, Berlin / Heidelberg, 3. Auflage.

Rupp, C. (2014). *Requirements-Engineering und -Management Aus der Praxis von klassisch bis agil*. Hanser, München, 6. Auflage.

Rupp, C. und Queins, S. (2012). *UML 2 glasklar: Praxiswissen für die UML-Modellierung*. Carl Hanser Verlag, 4. Auflage.

Schäfer, U., Spurk, C. und Steffen, J. (2012). A Fully Coreference-annotated Corpus of Scholarly Papers from the ACL Anthology. In *Proceedings of COLING 2012: Posters*, Seiten 1059–1070, Mumbai, MH, Indien. ACL.

Schenk, N. und Chiarcos, C. (2016). Unsupervised Learning of Prototypical Fillers for Implicit Semantic Role Labeling. In *Proceedings of the 2016 Conference of the NAACL: HLT*, Seiten 1473–1479, San Diego, CA, USA. ACL.

Schienmann, B. (2002). *Kontinuierliches Anforderungsmanagement: Prozesse - Techniken - Werkzeuge*. Programmer's Choice. Pearson Deutschland.

Schmuller, J. (2003). *Jetzt lerne ich UML: Der einfache Einstieg in die visuelle Objektmodellierung*. Markt+Technik Verlag, München.

Schneider, G. und Vecellio, S. (2011). *ICT-Systemabgrenzung, Anforderungsspezifikation und Evaluation: Grundlagen zur Systemanalyse und -beschaffung mit Beispielen, Fragen und Antworten*. Compendio Bildungsmedien, Zürich, Schweiz, 1. Auflage.

Schneider, H.-J. (Herausgeber) (1998). *Lexikon Informatik und Datenverarbeitung*. Walter de Gruyter, 4. Auflage.

Schulz, A. (2012). Last- und Performance-Tests optimal durchführen. `http://it-administrator.de/themen/netzwerkmanagement/fachartikel/117300.html`. Zuletzt abgerufen am 23.02.2017.

Schütze, H. (1998). Automatic Word Sense Discrimination. *Computational Linguistics*, 24(1):97–123.

Schwinn, H. (2011). *Requirements Engineering: Modellierung von Anwendungssystemen*. Oldenbourg Verlag, München.

Schwitter, R. (1998). *Kontrolliertes Englisch für Anforderungsspezifikationen*. Doktorarbeit, Universität Zürich.

Sennet, A. (2016). Ambiguity. In Zalta, E. N. (Herausgeber), *The Stanford Encyclopedia of Philosophy*. CSLI, Stanford University. Auflage: Spring 2016.

Shah, U. S. und Jinwala, D. C. (2015). Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey. *SIGSOFT Software Engineering Notes*, 40(5):1–7.

Shull, F., Carver, J., Travassos, G. H., Maldonado, J. C., Conradi, R. und Basili, V. R. (2003). Replicated Studies: Building a Body of Knowledge about Software Reading Techniques. In Juristo, N. und Moreno, A. M. (Herausgeber), *Lecture Notes on Empirical Software Engineering*, Kapitel 2, Seiten 39–84. World Scientific, River Edge, NJ, USA.

Shull, F., Rus, I. und Basili, V. R. (2000). How Perspective-Based Reading Can Improve Requirements Inspections. *Computer*, 33(7):73–79.

Shull, F., Rus, I. und Basili, V. R. (2001). Improving Software Inspections by Using Reading Techniques. In *Proceedings of the 23rd ICSE*, ICSE'01, Seiten 726–727, Washington, DC, USA. IEEE.

Simov, K. (2004). BulTreeBank Project Overview. BulTreeBank Project Technical Report. Technischer Bericht BTB-TR01, Linguistic Modelling Laboratory, Bulgarian Academy of Sciences.

Sommerville, I. (2007). *Software Engineering*. International Computer Science Series. Pearson Education, Essex, UK, 8. Auflage.

Sommerville, I. (2009). Web Chapter 27: Formal Specification. `http://www.SoftwareEngineering-9.com/Web/ExtraChaps/FormalSpec.pdf`. Zuletzt abgerufen am 19.08.2015.

Sommerville, I. (2011). *Software Engineering*. Xpert.press. Pearson, Boston, MA, USA.

Souter, C., Churcher, G., Hayes, J., Hughes, J. und Johnson, S. (1994). Natural Language Identification using Corpus-Based Models. In *HERMES - Journal of Language and Communication in Business*, Band: 7. Faculty of Modern Languages, Aarhus School of Business, Aarhus, Dänemark.

Springer Gabler (2015). Gabler Wirtschaftslexikon, Stichwort: Anspruchsgruppen. `http://wirtschaftslexikon.gabler.de/Archiv/1202/anspruchsgruppen-v6.html`. Zuletzt abgerufen am 20.06.2015.

Standish Group International (1995). The CHAOS Report (1994). `https://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf`. Zuletzt abgerufen am 18.01.2016.

Stanford NLP Group (2016). Stanford Deterministic Coreference Resolution System. `http://nlp.stanford.edu/software/dcoref.html`. Zuletzt abgerufen am 16.05.2016.

Stang, K. (2002). *Projektmanagement, Anforderungsanalyse und externe Qualitätssicherung: IT-Projekte durch umfassendes Anforderungsmanagement erfolgreich gestalten*. vdf Hochschulverlag AG, Zürich, Schweiz.

Statista (2016). The most spoken languages worldwide (speakers and native speaker in millions). `https://www.statista.com/statistics/266808/the-most-spoken-languages-worldwide/`. Zuletzt abgerufen am 28.10.2016.

Stede, M. (2012). *Discourse Processing*. Synthesis Lectures on HLT. Morgan & Claypool Publishers, Potsdam.

Stempfle, H. (1996). Der Einsatz Offener Systeme in der Praxis – Technologien, Standards, Probleme. Diplomarbeit, Fachhochschule Augsburg, Augsburg. Erreichbar unter: `http://www.hs-augsburg.de/inf/diplomarbeiten/langfassungen/stempfle-stork-1996/OpenSystems/einleit.htm#Interoperabilit%E4t_und_Portabilit%E4t`. Zuletzt abgerufen am 22.10.2016.

Stoyanov, V., Cardie, C., Gilbert, N., Riloff, E., Buttler, D. und Hysom, D. (2010). Coreference Resolution with Reconcile. In *Proceedings of the ACL 2010 Conference Short Papers*, Seiten 156–161, Uppsala, Schweden. ACL.

Stoyanov, V., Gilbert, N., Cardie, C. und Riloff, E. (2009). Conundrums in Noun Phrase Coreference Resolution: Making Sense of the State-of-the-Art. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, Seiten 656–664, Suntec, Singapore. ACL.

Strube, M. und Ponzetto, S. P. (2006). WikiRelate! Computing Semantic Relatedness Using Wikipedia. In *Proceedings of the 21st National Conference on Artificial Intelligence*, Band: 2. *AAAI'06*, Seiten 1419–1424, Boston, MA, USA. AAAI Press.

Telljohann, H., Hinrichs, E. W., Kübler, S., Zinsmeister, H. und Beck, K. (2015). Stylebook for the Tübingen Treebank of Written German (TüBa-D/Z). Technischer Bericht, Universität Tübingen.

Theda, M. (2017). Was ist gemeint? Strukturell ambige Sätze als Herausforderung für Parsing-Ansätze. Masterarbeit, Universität Paderborn, Paderborn.

Tichy, W. F., Landhäußer, M. und Körner, S. J. (2015). nlrpBENCH: A Benchmark for Natural Language Requirements Processing. In *Multikonferenz Software Engineering & Management 2015*.

Tiemeyer, E. (2013). *Handbuch IT-Management: Konzepte, Methoden, Lösungen und Arbeitshilfen für die Praxis*. Carl Hanser Verlag, München, 5. Auflage.

Tjong, S. F. (2008). *Avoiding Ambiguity in Requirements Specifications*. Doktorarbeit, University of Nottingham, Nottingham, UK.

Tjong, S. F. und Berry, D. M. (2013). The Design of SREE – A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned. In Doerr, J. und Opdahl, A. L. (Herausgeber), *Requirements Engineering: Foundation for Software Quality*, Band: 7830. *LNCS*, Seiten 80–95. Springer, Berlin / Heidelberg.

Toral, A., Muñoz, R. und Monachini, M. (2008). Named Entity WordNet. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S. und Tapias, D. (Herausgeber), *Proceedings of the 6th LREC*, Marrakech, Marokko. ELRA.

Tye, M. (1998). Routledge Encyclopedia of Philosophy: Vagueness. `https://www.rep.routledge.com/articles/vagueness/v-1/`. Zuletzt abgerufen am 04.01.2016.

Umber, A. und Bajwa, I. S. (2011). Minimizing Ambiguity in Natural Language Software Requirements Specification. In *Proceedings of the 6th ICDIM*, Seiten 102–107, Melbourn, VIC, Australien. IEEE.

Uryupina, O., Poesio, M., Giuliano, C. und Tymoshenko, K. (2012). Disambiguation and Filtering Methods in Using Web Knowledge for Coreference Resolution. In *Cross-Disciplinary Advances in Applied Natural Language Processing: Issues and Approaches*, Seiten 185–201. IGI Global.

Verma, K. und Kass, A. (2008). Requirements Analysis Tool: A Tool for Automatically Analyzing Software Requirements Documents. In Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T. und Thirunarayan, K. (Herausgeber), *Proceedings of the ISWC 2008*, Band: 5318. *LNCS*, Seiten 751–763. Springer, Berlin / Heidelberg.

Versley, Y., Ponzetto, S. P., Poesio, M., Eidelman, V., Jern, A., Smith, J., Yang, X. und Moschitti, A. (2008). BART: a modular toolkit for coreference resolution. In *Proceedings of the 2008 Conference of the ACL*, Seiten 9–12, Columbus, OH, USA. ACL.

Vlas, R. und Robinson, W. N. (2011). A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. In *Proceedings of the 44th HICSS*, Seiten 1–10, Kauai, HI, USA. IEEE.

Vogel, O., Arnold, I., Chughtai, A., Ihler, E., Kehrer, T., Mehlig, U. und Zdun, U. (2009). *Software-Architektur: Grundlagen, Konzepte, Praxis*. Spektrum Akademischer Verlag, Heidelberg, 2. Auflage.

Vöhringer, J. und Fliedl, G. (2011). Adapting the lesk algorithm for calculating term similarity in the context of requirements engineering. In Pokorny, J., Repa, V., Richta, K., Wojtkowski, W., Linger, H., Barry, C. und Lang, M. (Herausgeber), *Information Systems Development: Business Systems and Services: Modeling and Development*, Seiten 781–790. Springer, New York, NY, USA.

Vossen, G., Haselmann, T. und Hoeren, T. (2012). *Cloud-Computing für Unternehmen: Technische, wirtschaftliche, rechtliche und organisatorische Aspekte*. dpunkt.verlag, Münster / Paderborn, 1. Auflage.

VSEK Konsortium (2007a). Adaptierbarkeit. `http://www.software-kompetenz.de/?29860`. Zuletzt abgerufen am 19.10.2016.

VSEK Konsortium (2007b). Formale Spezifikationstechniken. `http://www.software-kompetenz.de/servlet/is/16651`. Zuletzt abgerufen am 19.08.2015.

VSEK Konsortium (2007c). Qualitätsmodell des Software Engineering Instituts (SEI). `http://www.software-kompetenz.de/?18738`. Zuletzt abgerufen am 20.10.2016.

Weischedel, R., Hovy, E., Marcus, M., Palmer, M., Belvin, R., Pradhan, S., Ramshaw, L. und Xue, N. (2011). OntoNotes: A Large Training Corpus for Enhanced Processing. In Olive, J., Christianson, C. und McCary, J. (Herausgeber), *Handbook of Natural Language Processing and Machine Translation – DARPA Global Autonomous Language Exploitation*. Springer, New York, NY, USA.

Weischedel, R., Pradhan, S., Ramshaw, L., Kaufman, J., Franchini, M., El-Bachouti, M., Xue, N., Palmer, M., Hwang, J. D., Bonial, C., Choi, J., Mansouri, A., Foster, M., Hawwary, A.-a., Marcus, M., Taylor, A., Greenberg, C., Hovy, E., Belvin, R. und Houston, A. (2012). OntoNotes Release 5.0 with OntoNotes DB Tool v0.999 beta. Technischer Bericht 2012-09-28, Raytheon BBN Technologies.

Weischedel, R., Pradhan, S., Ramshaw, L., Micciulla, L., Palmer, M., Xue, N., Marcus, M., Taylor, A., Babko-Malaya, O., Hovy, E., Belvin, R. und Houston, A. (2007). OntoNotes Release 1.0 with OntoNotes DB Tool v. 0.9 beta. Technischer Bericht 2007-02-15, BBN Technologies.

Wiegers, K. E. (2005). *Software Requirements*. Microsoft Press, Washington, DC, USA, 2. Auflage.

Witt, A., Heid, U., Sasaki, F. und Sérasset, G. (2009). Multilingual language resources and interoperability. *Language Resources and Evaluation*, 43(1):1–14.

WordNet (2010). wnstats - WordNet 3.0 database statistics. `http://wordnet.princeton.edu/wordnet/man/wnstats.7WN.html`. Zuletzt abgerufen am 28.03.2016.

Wu, Z. und Palmer, M. (1994). Verbs Semantics and Lexical Selection. In *Proceedings of the 32nd Annual Meeting on ACL*, ACL'94, Seiten 133–138, Stroudsburg, PA, USA. ACL.

Xue, N., Xia, F., Chiou, F.-D. und Palmer, M. (2005). The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238. Cambridge University Press.

Yadav, S. B., Bravoco, R. R., Chatfield, A. T. und Rajkumar, T. M. (1988). Comparison of Analysis Techniques for Information Requirement Determination. *Communication of the ACM*, 31(9):1090–1097.

Yang, H., de Roeck, A., Gervasi, V., Willis, A. und Nuseibeh, B. (2010a). Extending Nocuous Ambiguity Analysis for Anaphora in Natural Language Requirements. In *Proceedings of the 18th IEEE RE*, Seiten 25–34, Sydney, NSW, Australien. IEEE.

Yang, H., de Roeck, A., Willis, A. und Nuseibeh, B. (2010b). A Methodology for Automatic Identification of Nocuous Ambiguity. In *Proceedings of the 23rd COLING*, COLING'10, Seiten 1218–1226, Stroudsburg, PA, USA. ACL.

Yang, H., Roeck, A., Gervasi, V., Willis, A. und Nuseibeh, B. (2011). Analysing anaphoric ambiguity in natural language requirements. *Requirements Engineering*, 16(3):163–189.

Yang, H., Willis, A., De Roeck, A. und Nuseibeh, B. (2010c). Automatic Detection of Nocuous Coordination Ambiguities in Natural Language Requirements. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE'10, Seiten 53–62, New York, NY, USA. ACM.

Yang, X. und Su, J. (2007). Coreference Resolution Using Semantic Relatedness Information from Automatically Discovered Patterns. In *Proceedings of the 45th Annual Meeting of the ACL*, Seiten 528–535, Prag, Tschechische Republik. ACL.

Zavrel, J., Daelemans, W. und Veenstra, J. (1997). Resolving PP attachment Ambiguities with Memory-Based Learning. In Ellison, T. (Herausgeber), *CONLL '97: Computational Natural Language Learning*, Seiten 136–144. ACL, Madrid, Spanien.

Zhang, X., Wu, C. und Zhao, H. (2012). Chinese Coreference Resolution via Ordered Filtering. In *Joint Conference on EMNLP and CONLL - Proceedings of the Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes*, Seiten 95–99, Jeju, Korea. ACL.

Teil V

Anhang

Programmoberflächen A

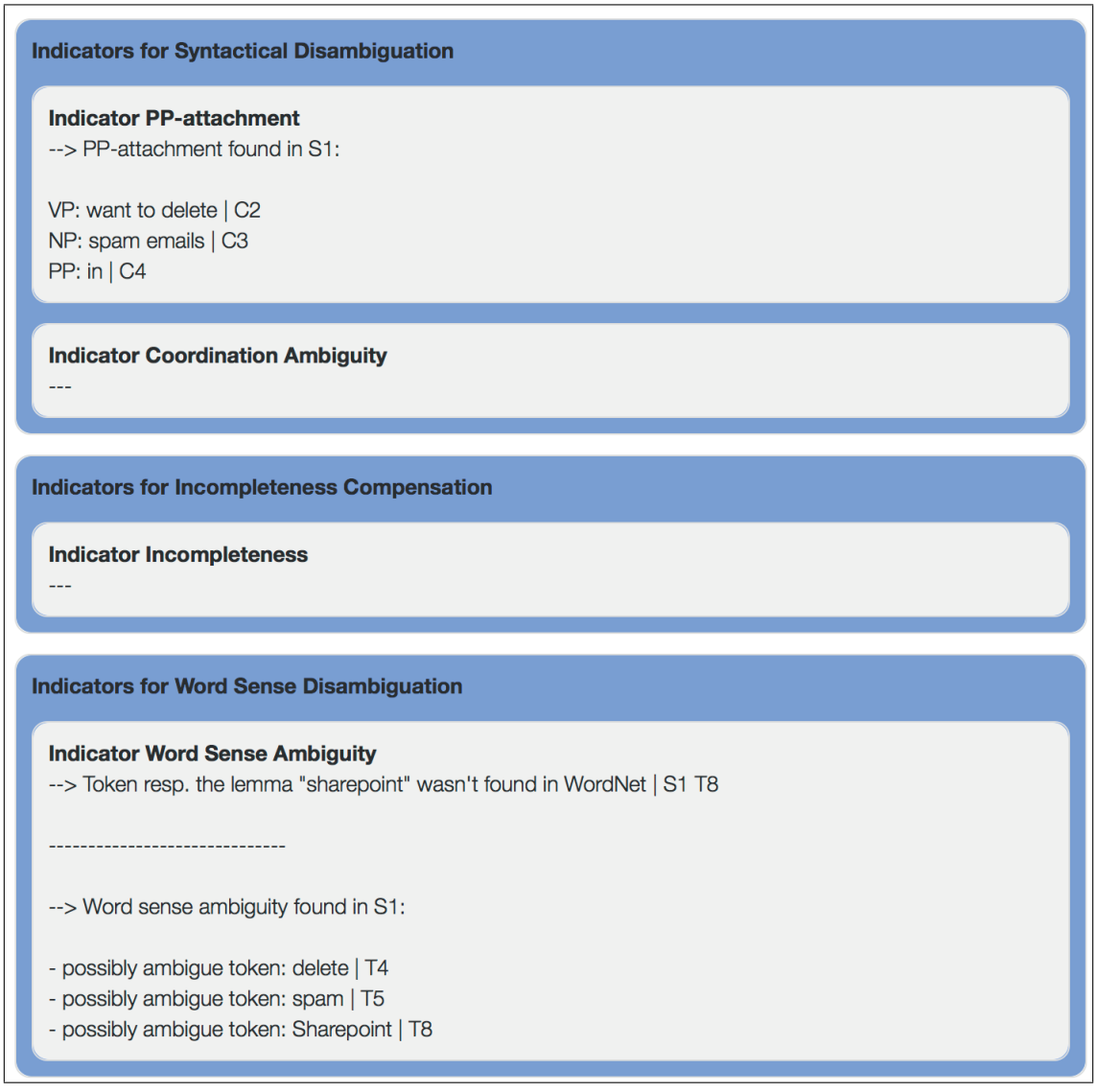


Abbildung A.1: Erläuterungen zur Indikatoranwendung für Endanwender

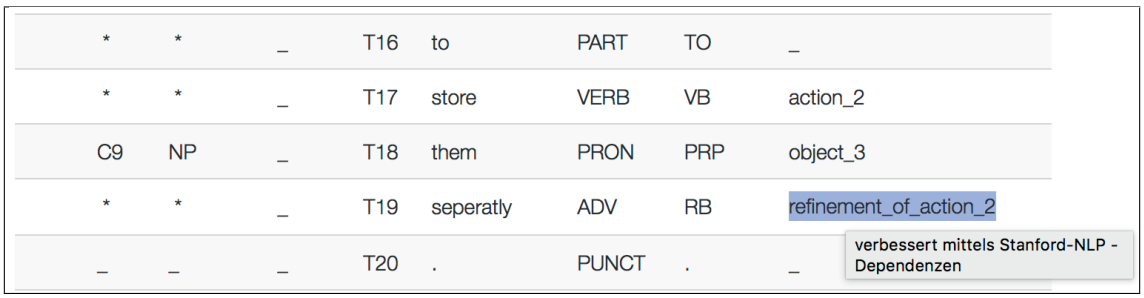


Abbildung A.2: Erläuternde Darstellung der Korrektur mittels *CoreNLP*

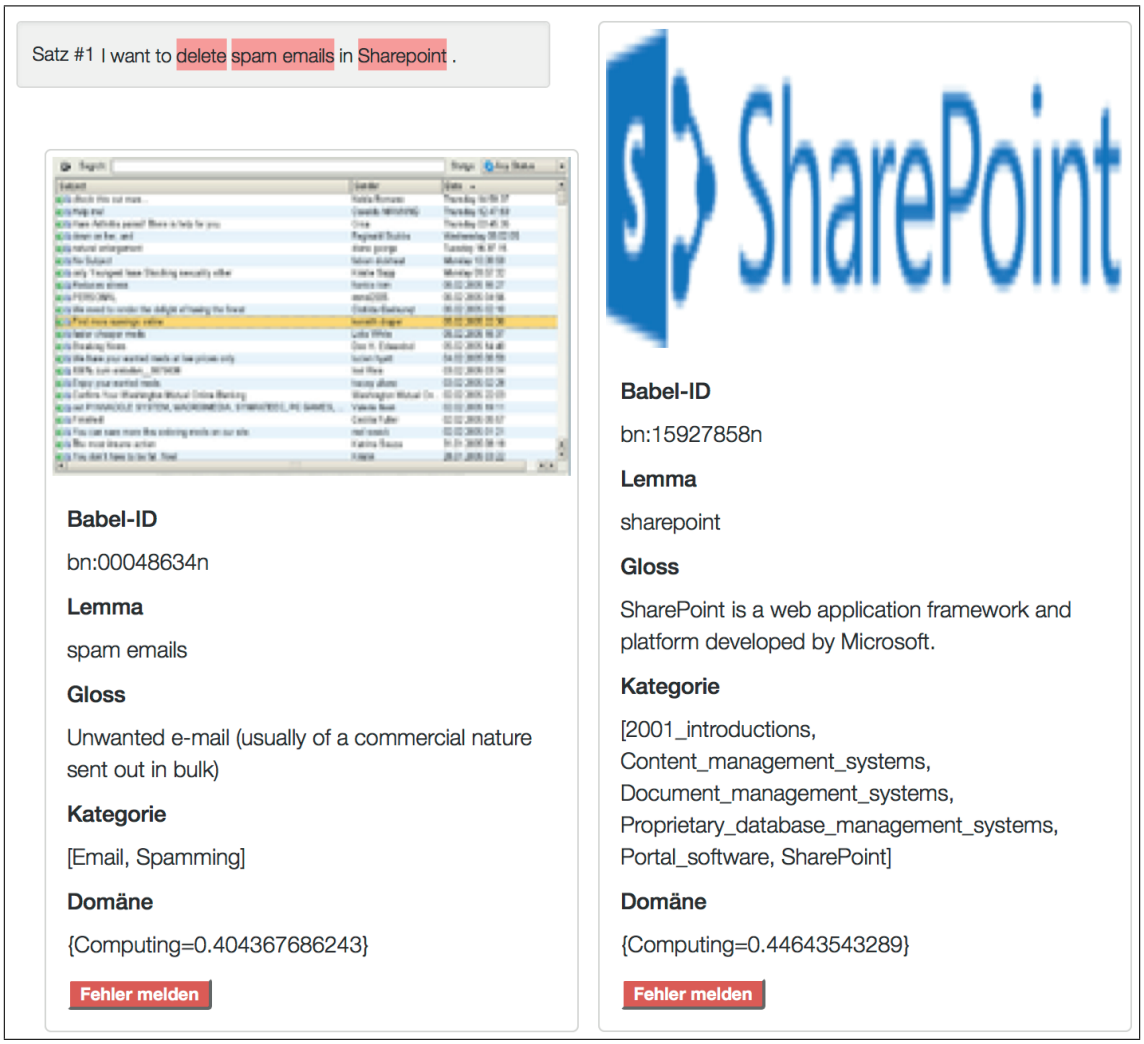


Abbildung A.3: Ergebnis der lexikalischen Disambiguierung mittels Babelfy

	*	*	_	T13	format	VERB	VB	action_2
	C7	NP	_	T14	text	NOUN	NN	object_2
	C8	PP	_	T15	as	ADP	IN	argument_of_action_1
	C9	NP	_	T16	bold	ADJ	JJ	argument_of_action_1
	*	*	_	T17	or	CONJ	CC	argument_of_action_1
	C10	LST	_	T18	italic	ADJ	JJ	argument_of_action_1
	_	_	_	T19	.	PUNCT	.	_

verbessert mittels BabelNet | ehemaliges Pos: NN

Abbildung A.4: Erläuternde Darstellung der POS-Korrektur mittels BabelNet

#	Kette
100000	"application" in sentence 3 "program" in sentence 7 "software" in sentence 9
2	"I" in sentence 2 "I" in sentence 3 "I" in sentence 4 "I" in sentence 5 "my" in sentence 7 "I" in sentence 8 "user" in sentence 8
6	"emails" in sentence 3 "emails" in sentence 6 "emails" in sentence 9
8	"email spam" in sentence 4 "the spam" in sentence 5

Abbildung A.5: Darstellung erkannter Koreferenzketten mittels *CoreNLP*

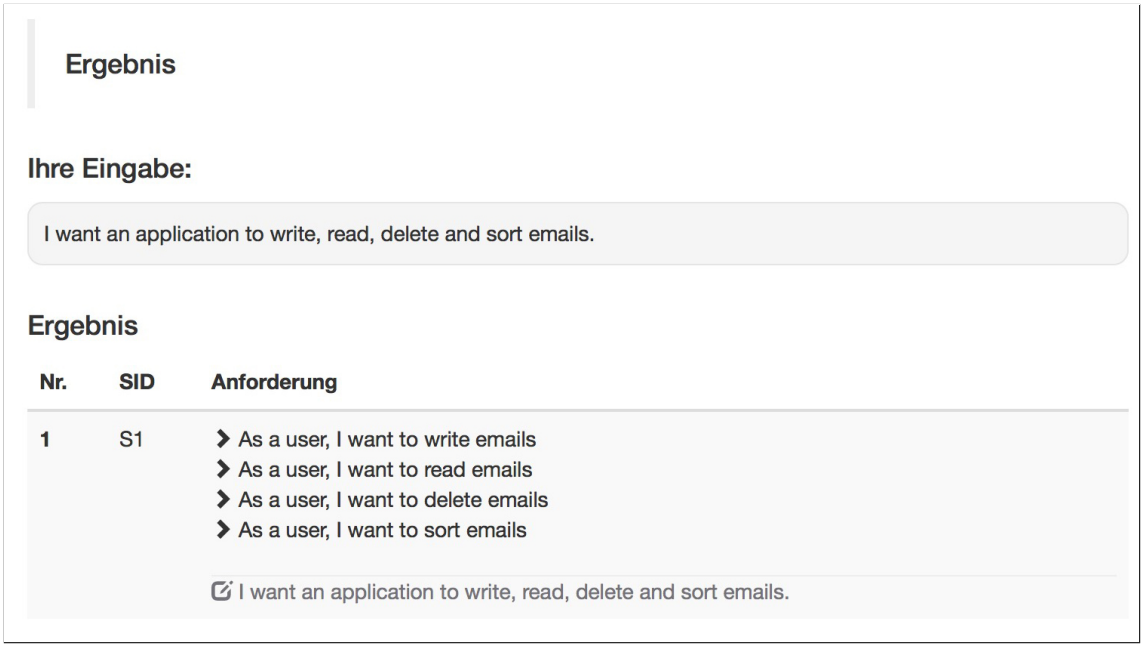


Abbildung A.6: Ergebnis der Verarbeitung

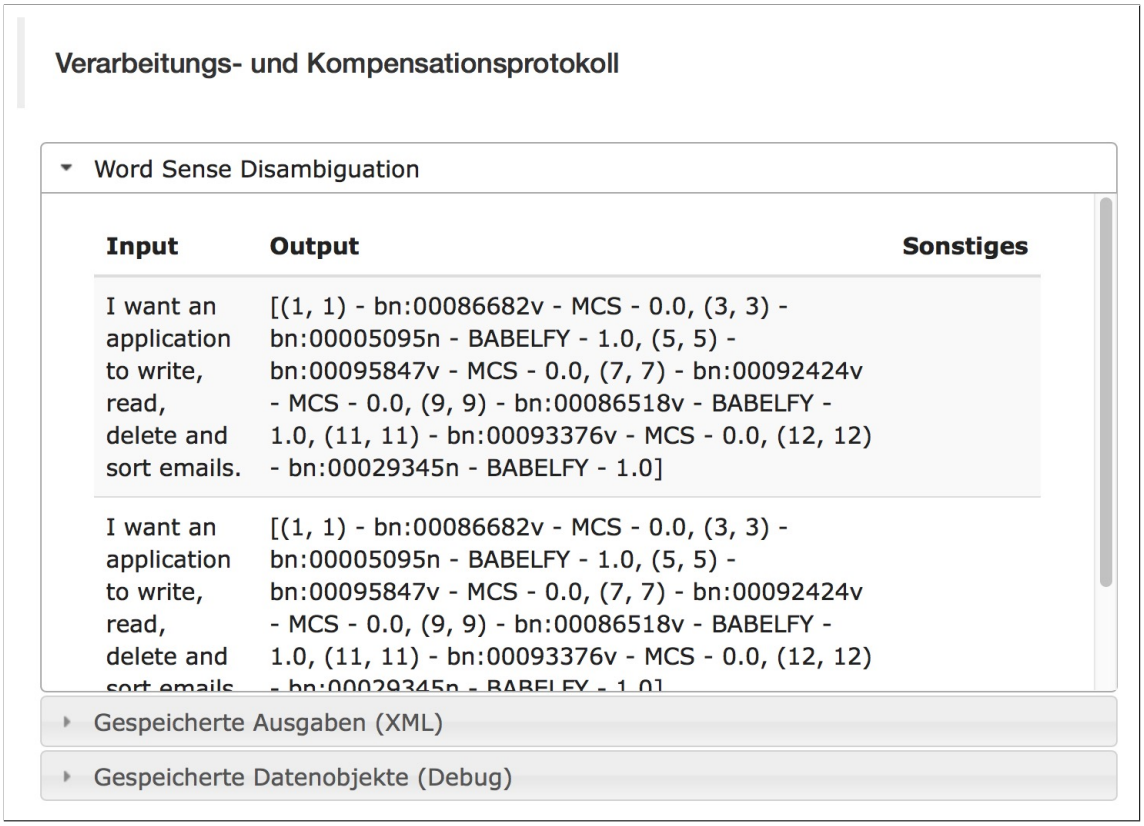


Abbildung A.7: Verarbeitungs- und Kompensationsprotokoll

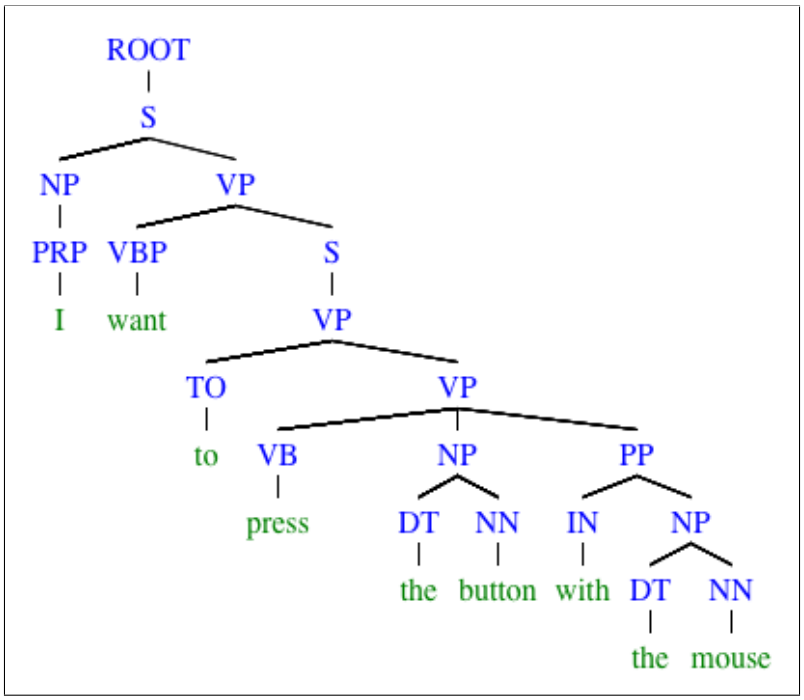


Abbildung A.8: Beispielsyntaxbaum des *Stanford Parsers*

Material zur Evaluation B

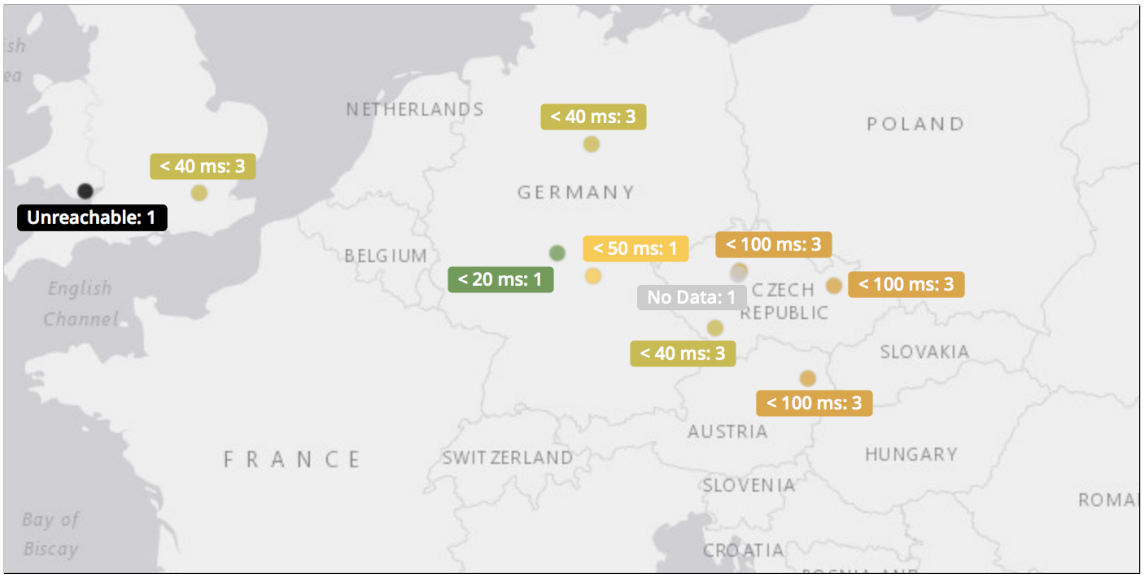


Abbildung B.1: Nach Geschwindigkeit klassifizierte Messtellen (RIPE NCC)

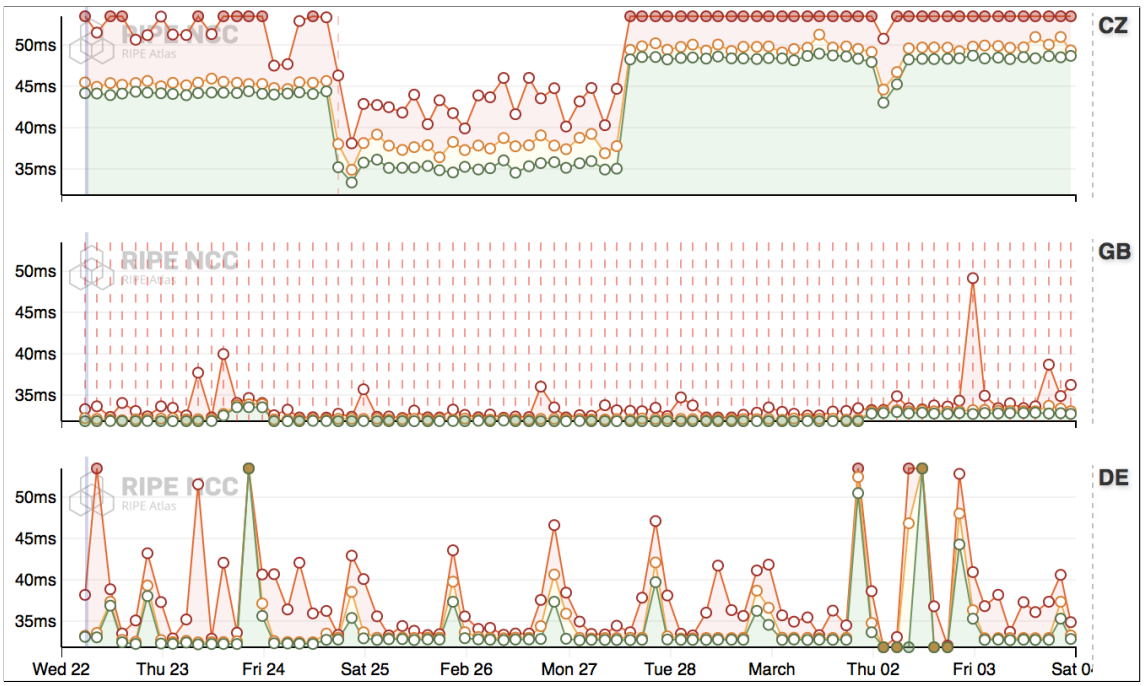
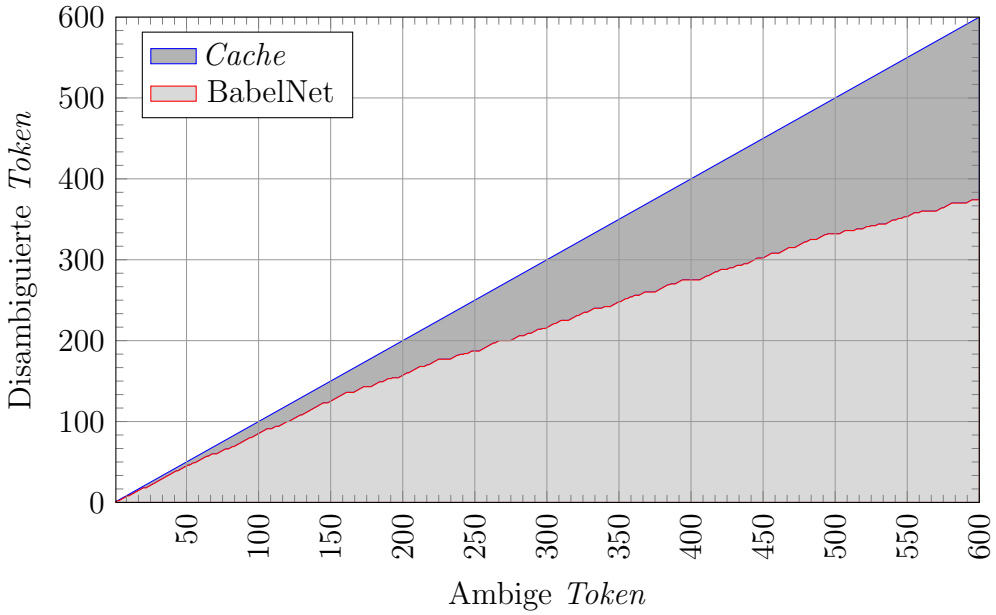


Abbildung B.2: Messergebnisse nach Ländern (Auszug)

(A) Domänenspezifische Anfragen:



(B) Domänenübergreifende Anfragen:

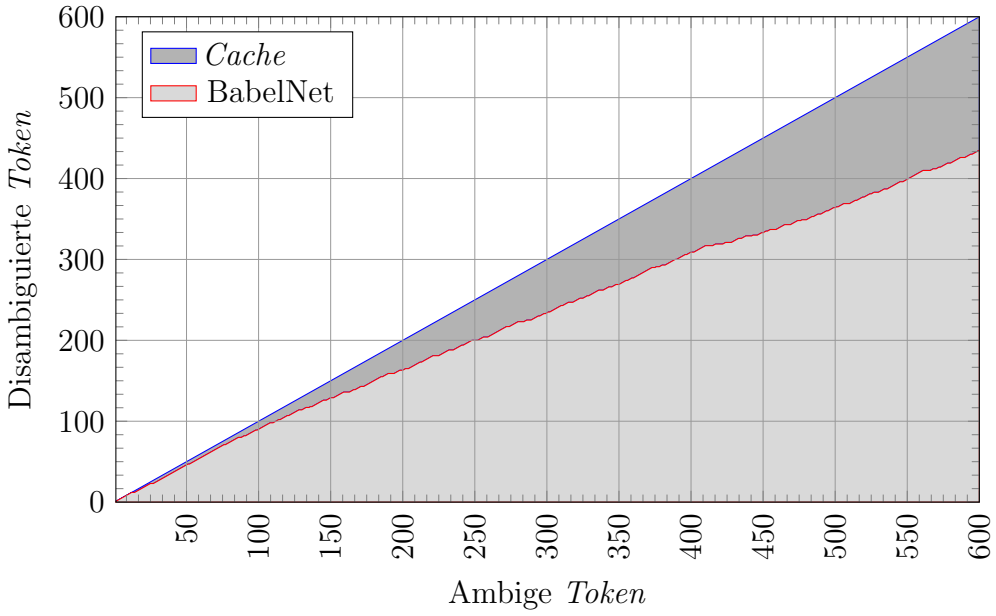


Abbildung B.3: Ressourcenverteilung lex. Disambiguierungsanfragen

Ergänzende Ausführungen C

C.1 Ausgewählte Verfahren der Textvorverarbeitung

Gegenstand dieser Arbeit ist, wie in Abschnitt 3.2 ersichtlich wird, UGC, der weder Struktur noch Textannotationen aufweist und einer Vorverarbeitung bedarf, um die Anwendung der Textverarbeitung zu ermöglichen. Die Vorverarbeitung von Fließtexten (engl. *preprocessing*) ist dabei ein elementarer Schritt im NLP. Ziel ist es, aus einem Dokument „linguistische Einheiten, wie z. B. Wörter, Phrasen, Sätze, Absätze oder Diskursabschnitte“ (Carstensen et al., 2010, S. 264) zu isolieren und den folgenden Verarbeitungsschritten in einer homogenen Form zugänglich zu machen.

Die sequenzielle Anwendung einzelner Textvorverarbeitungsschritte muss auf die Anforderungen der Eingabetexte angepasst werden. Im Folgenden werden Verfahren der Sprachenidentifizierung und Satzgrenzenerkennung exemplarisch vorgestellt, die unter dem Kriterium „Anwendbarkeit auf UGC“ ausgewählt wurden. Diese Verfahren haben im Rahmen dieser Arbeit eine besondere Bedeutung, da sie (auch in Kombination) die Weiterverarbeitung maßgeblich durch eine frühe Strukturanpassung (Unterteilung in Sätze) und Klassifikation (Einteilung in Sprachen) mitgestalten.

C.1.1 Sprachenidentifizierung

Ansätze zur Sprachenidentifizierung (engl. *language identification*) haben die Erkennung der jeweiligen natürlichen Sprache, in der ein Fließtext verfasst wurde, zum Gegenstand (Baldwin und Lui, 2010, S. 229). Dies ist beispielsweise im *Information Retrieval* (IR) relevant (Klose und Wrigley, 2014, S. 62 ff.), in dessen *Preprocessing* eine Vielzahl von Dokumenten unterschiedlicher Sprachen verarbeitet werden. Aber auch in der vorliegenden Arbeit hat diese Identifizierung hohe Relevanz, da sie darüber entscheidet, ob eine FA der Weiterverarbeitung zugeführt wird oder nicht. Da die meisten Komponenten auf eine begrenzte Anzahl von Sprachen zugeschnitten sind, ist eine zuverlässige Sprachenidentifizierung ein wichtiger Schritt der Qualitätssicherung.

Diesbezüglich existieren verschiedene Vorgehensweisen der Sprachenidentifizierung (Souter et al., 1994, S. 183 ff.). So kann ein Fließtext beispielsweise auf das Vorkommen charakteristischer Zeichen (z. B. „ñ“, „ß“) oder spezifischer Funktionswörter (z. B. „*the*“, „*and*“) untersucht werden („*Small Word Technique*“), die für eine Sprache auszeichnend sind (Grefenstette, 1995, S. 265 ff.). Diese Arbeit beschränkt sich auf die Unterscheidung in (1) wortbasierte Sprachenerkenner und (2) N-Gramm basierte Sprachenerkenner (Řehůřek und Kolkus, 2009, S. 359 ff.; Langer, 2002, S. 99 f., 106), die als zuverlässige Vorgehensweisen gelten (Souter et al., 1994).

Wortbasierte Sprachenerkenner (1) basieren auf einem Abgleich mit Wörterbüchern und haben den Vorteil, dass sie manuell angepasst werden können (z. B. Hinzunahme von Fachtermini). Dies bedeutet im Umkehrschluss aber auch einen erhöhten Aufwand

und eine schlechtere Performanz, da unbekannte Wörter nicht zugeordnet werden können. Anders ist dies bei (2). Hier wird ein System auf Basis von N-Grammen trainiert, sodass die „[...] Wahrscheinlichkeit gängiger Bytefolgen in elektronischen Texten [...]“ (Langer, 2002, S. 99 f.) ausschlaggebend für die Klassifikation ist. Ein großer Vorteil ist, dass die Identifikation unabhängig vom (ggf. teils unbekannten) Vokabular erfolgen kann. Ein Nachteil besteht darin, dass Identifizierungsfehler nur schwer nachzuvollziehen und zu korrigieren sind (Langer, 2002, S. 100, 106).

Zur Umsetzung gibt es bereits eine Reihe von Softwarebibliotheken, welche die Sprachenidentifizierung mittels einer Vielzahl von Programmiersprachen ermöglichen (z. B. *language-detector*[170] oder *Compact Language Detector 2*[171]). Darüber hinaus existieren sowohl kostenpflichtige als auch kostenlose *Web Services*, die Sprachenidentifizierung auf Grundlage von Fließtexten mittels Web-Schnittstellen anbieten[172].

Oftmals wird die „[...] automatische Sprachenidentifizierung für elektronische Dokumente, deren Mindestlänge eine bestimmte Wortzahl überschreitet und die regulären Text enthalten, [...] als weitgehend gelöstes Problem“ verstanden (Langer, 2002, S. 99). Dies bedeutet, dass bestehende Verfahren der Sprachenidentifizierung überwiegend gute Resultate erzielen und dass längere und „saubere“ Texte zu besseren Identifizierungsergebnissen führen können (Klose und Wrigley, 2014, S. 62). Das Interesse an der Sprachenidentifizierung von kurzen und „verrauschten“ Texten nimmt jedoch mit der zunehmenden Relevanz von Kurznachrichtendiensten und sozialen Netzwerken rapide zu (z. B. Dias Cardoso und Roy, 2016; Lui und Baldwin, 2014; Carter et al., 2012), was auch der Verarbeitung von qualitativ stark schwankenden Anforderungsbeschreibungen zuträglich ist.

Nach Langer (2002, S. 103) liegen die „[...] Erkennungsraten aller bekannten Algorithmen [bei] über 99% [...]“, wenn es sich bei den Eingabedokumenten um Standarddokumente handelt. Diese Dokumente sind nach Langer (2002, S. 103) monolingual und enthalten regulären Text und mindestens 20 Wörter. Daneben gibt zum Beispiel Dunning (1994) bei der statistischen Sprachenidentifikation auf N-Gramm-Basis an, dass bereits wenige Tausend Wörter als Trainingsdaten ausreichen, um eine gute Perfomanz zu erzielen und bereits ab 10 Zeichen gute und ab Zeichenketten mit 50 Zeichen sehr gute Ergebnisse zu erzielen sind (Dunning, 1994, S. 1). Ein großer Anteil an Fachsprache kann dabei neben Faktoren wie einer kurzen Dokumentenlänge, Wortwiederholungen und Eigennamen negativen Einfluss auf die Erkennungsrate haben (Langer, 2002, S. 102 ff.).

In Hinblick auf die Anforderungen des OTF-*Computings* sind Verfahren zu bevorzugen, die auch bei kurzen und fehlerhaften Texten eine gute Performanz erzielen. Darüber hinaus sind Anforderungsspezifikationen zwar üblicherweise von Fachtermini geprägt, Anforderungsbeschreibungen ähneln in ihren Merkmalen aber eher UGC mit einem geringeren Anteil an Fachtermini. In der Anwendung auf UGC zeigen aktuelle Untersuchungen, dass N-Gramm-basierte Ansätze die robustesten und besten Ergebnisse erzielen (z. B. Dias Cardoso und Roy, 2016).

[170] Siehe weiterführend: `https://github.com/optimaize/language-detector` (Stand: 11.01.17).
[171] Siehe weiterführend: `https://github.com/CLD2Owners/cld2` (Stand: 11.01.17).
[172] Siehe weiterführend: `https://detectlanguage.com` (Stand: 11.01.17).

C.1.2 Satzgrenzenerkennung

Die Satzgrenzenerkennung (engl. *sentence boundary disambiguation, SBD*) wird, wie auch die Sprachenidentifizierung, oftmals als gelöstes Problem verstanden (Read et al., 2012a), welchem mit einer Vielzahl an unterschiedlichen Ansätzen und Methoden begegnet werden kann. Eine Unterteilung dieser Ansätze findet in (1) regelbasierte SBD-Verfahren (Expertenwissen; Heuristiken und Gazetteers) und (2) ML-Ansätze (Annotierte Korpora; Goldstandards) statt (Read et al., 2012a, S. 987). Eine entsprechende Übersicht geben Read et al. (2012a) sowie Kiss und Strunk (2006) und eine Auswahl findet sich in Tabelle C.1 zusammen mit dem jeweiligen F_1-Wert auf verschiedenen Korpora (Read et al., 2012a, S. 989 f.)[173].

Verfahren der Satzgrenzenerkennung werden vielfach auf Fließtexte wie Nachrichtentexte angewendet und erreichen teils sehr gute Ergebnisse (Kiss und Strunk, 2006). Seltener werden die Verfahren zur Vorverarbeitung von UGC wie Produktbewertungen (López und Pardo, 2015) oder Kurznachrichten aus sozialen Netzwerken (Rudrapal et al., 2015) herangezogen.

Aus diesem Grund untersuchen Read et al. (2012a, S. 991) explizit auch die Performanz etablierter Ansätze auf informalen Texten wie UGC. Hierzu ziehen sie Texte der NLP- und Linux-Domäne aus Webblogs (WNB/WLB) heran (Read et al., 2012b), die einige tausend Sätze umfassen. Wie in Tabelle C.1 ersichtlich wird, funktionieren die Verfahren allesamt auf klassischen Fließtexten besser als auf UGC, wenngleich die Ergebnisse auf UGC auch noch immer als solide zu bezeichnen sind.

	Brown	**CDC**	**GENIA**	**WSJ**	**WNB**	**WLB**
CoreNLP	87,7	72,1	**98,8**	91,3	95,3	89,1
LingPipe	93,0	86,3	**99,6**	88,0	94,4	92,7
RASP	**96,8**	96,1	98,9	99,0	95,4	92,8
Splitta	95,4	96,1	99,0	**99,2**	94,0	91,2

Tabelle C.1: Vergleich von Satzendeerkennungstools auf verschiedenen Korpora. Aus Read et al. (2012a, S. 990, 992)

Ein regelbasiertes Verfahren zur Satzgrenzenerkennung beinhaltet das **RASP-System**[174]. Briscoe et al. (2006) bzw. Briscoe und Carroll (2002) stellen mit *Robust Accurate Statistical Parsing* (RASP) ein NLP-System zur syntaktischen Annotation von Freitext zur Verfügung. Es handelt sich demnach hierbei nicht um einen reinen Ansatz zur Satzgrenzenerkennung. RASP kann auf allen Korpora solide Ergebnisse erzielen, wenngleich auch Read et al. (2012a, S. 991) den Perfomanzverlust auf dem CDC-Korpus hervorheben. Darüber hinaus weist Briscoe (2006, S. 24) auf Probleme bei irregulärer Nutzung von Satzzeichen wie zum Beispiel „------“ oder „(A):-“ hin. Da sowohl zur Strukturierung (z. B. Auflistung) als auch zur Trennung von Inhalten in Anforderungsbeschreibungen nur Sonderzeichen zur Verfügung stehen, stellt dieser Umstand ein Problem dar, da die häufige Satzzeichenverwendung zu erwarten ist.

[173] Neben dem *Brown*-Korpus und WSJ-Texten werden das *Conan Doyle Corpus* (CDC), das aus mehreren *Sherlock Holmes*-Geschichten besteht und das GENIA-Korpus, das eine Sammlung von 16.392 Sätzen aus biomedizinischen Forschungszusammenfassungen darstellt, zur Evaluation herangezogen (Kim et al., 2003).

[174] Siehe weiterführend: `http://ilexir.co.uk/applications/rasp/` (Stand: 11.01.17).

Das *Stanford* ***CoreNLP*** beinhaltet ebenfalls einen *Sentence Splitter* (*ssplit*), der nicht auf dem ursprünglichen Fließtext arbeitet, sondern auf einer zuvor tokenisierten Variante (Manning et al., 2014, S. 3). Beispielsweise wird bereits beim Tokenisieren regelbasiert zwischen Satzzeichen, die einen Satz beenden und Satzzeichen, die zum Beispiel eine Abkürzung markieren, unterschieden. Die Evaluationsergebnisse von *CoreNLP* sind auf den Webkorpora nicht überzeugend (vgl. Tabelle C.1).

Read et al. (2012a, S. 991) führen als mögliche Erklärungen zum einen an, dass bestimmte Fehler schon durch den vorgeschalteten Tokenisierer entstehen können und zum anderen, dass die zugrundeliegenden Regeln zur Satzgrenzenerkennung unter Umständen nicht ständig gepflegt und weiterentwickelt werden, wie es bei Systemen der Fall ist, die speziell auf diese Aufgabe zugeschnitten sind. Schließlich umfasst *CoreNLP* eine Vielzahl an NLP-Anwendungen.

Bei dem *Python-Tool* **Splitta**[175] handelt es sich um ein NLP-*Toolkit*, welches primär der Tokenisierung und der SBD dient (Gillick, 2009). Genutzt wird sowohl ein SVM-basierter Ansatz als auch ein Naïve Bayes-basierter Klassifikationsalgorithmus, der auf dem WSJ- sowie auf dem *Brown*-Korpus sehr gute Ergebnisse für die englische Sprache liefert. Auf UGC funktioniert der Ansatz allerdings schlechter, wie Rudrapal et al. (2015, S. 5) am Beispiel von Twitter- und Facebook-Nachrichten aufzeigen.

Um einen auf ML basierenden Ansatz handelt es sich beim *Tool* **LingPipe**[176]. Es stellt ein NLP-*Toolkit* dar, dass neben der Satzgrenzenerkennung unter anderem auch POS-*Tagging* und NER bietet. Zur Erkennung von Satzgrenzen steht standardmäßig ein MEDLINE-Modell als *LingPipe SentenceModel* bereit, welches auf 13 Millionen biomedizinischen Kurztexten trainiert wurde (Reese, 2015, S. 92; Morris, 2011).

Es wird deutlich, dass keines der ausgewählten Verfahren das „perfekte Verfahren" zur Satzgrenzenerkennung auf allen Korpora ist. Dennoch empfehlen sich Systeme wie RASP und LingPipe im OTF-Kontext aufgrund der guten Performanz auf UGC sowie auf strukturierten Texten.

[175] Siehe weiterführend: `http://code.google.com/p/splitta/` (Stand: 17.01.17).
[176] Siehe weiterführend: `http://alias-i.com/lingpipe/` (Stand: 17.01.17).

Das Heinz Nixdorf Institut – Interdisziplinäres Forschungszentrum für Informatik und Technik

Das Heinz Nixdorf Institut ist ein Forschungszentrum der Universität Paderborn. Es entstand 1987 aus der Initiative und mit Förderung von Heinz Nixdorf. Damit wollte er Ingenieurwissenschaften und Informatik zusammenführen, um wesentliche Impulse für neue Produkte und Dienstleistungen zu erzeugen. Dies schließt auch die Wechselwirkungen mit dem gesellschaftlichen Umfeld ein.

Die Forschungsarbeit orientiert sich an dem Programm „Dynamik, Mobilität, Vernetzung: Eine neue Schule des Entwurfs der technischen Systeme von morgen“. In der Lehre engagiert sich das Heinz Nixdorf Institut in Studiengängen der Informatik, der Ingenieurwissenschaften und der Wirtschaftswissenschaften.

Heute wirken am Heinz Nixdorf Institut neun Professoren mit insgesamt 150 Mitarbeiterinnen und Mitarbeitern. Pro Jahr promovieren hier etwa 20 Nachwuchswissenschaftlerinnen und Nachwuchswissenschaftler.

Heinz Nixdorf Institute – Interdisciplinary Research Centre for Computer Science and Technology

The Heinz Nixdorf Institute is a research centre within the University of Paderborn. It was founded in 1987 initiated and supported by Heinz Nixdorf. By doing so he wanted to create a symbiosis of computer science and engineering in order to provide critical impetus for new products and services. This includes interactions with the social environment.

Our research is aligned with the program “Dynamics, Mobility, Integration: Enroute to the technical systems of tomorrow.” In training and education the Heinz Nixdorf Institute is involved in many programs of study at the University of Paderborn. The superior goal in education and training is to communicate competencies that are critical in tomorrows economy.

Today nine Professors and 150 researchers work at the Heinz Nixdorf Institute. Per year approximately 20 young researchers receive a doctorate.

Zuletzt erschienene Bände der Verlagsschriftenreihe des Heinz Nixdorf Instituts

Bd. 344 BRÖKELMANN, J.: Systematik der virtuellen Inbetriebnahme von automatisierten Produktionssystemen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 344, Paderborn, 2015 – ISBN 978-3-942647-63-2

Bd. 345 SHAREEF, Z.: Path Planning and Trajectory Optimization of Delta Parallel Robot. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 345, Paderborn, 2015 – ISBN 978-3-942647-64-9

Bd. 346 VASSHOLZ, M.: Systematik zur wirtschaftlichkeitsorientierten Konzipierung Intelligenter Technischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 346, Paderborn, 2015 – ISBN 978-3-942647-65-6

Bd. 347 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 11. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 29. und 30. Oktober 2015, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 347, Paderborn, 2015 – ISBN 978-3-942647-66-3

Bd. 348 HEINZEMANN, C.: Verification and Simulation of Self-Adaptive Mechatronic Systems. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 348, Paderborn, 2015 – ISBN 978-3-942647-67-0

Bd. 349 MARKWART, P.: Analytische Herleitung der Reihenfolgeregeln zur Entzerrung hochauslastender Auftragsmerkmale. Dissertation, Fakultät für Wirtschaftswissenschaften, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 349, Paderborn, 2015 – ISBN 978-3-942647-68-7

Bd. 350 RÜBBELKE, R.: Systematik zur innovationsorientierten Kompetenzplanung. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 350, Paderborn, 2016 – ISBN 978-3-942647-69-4

Bd. 351 BRENNER, C.: Szenariobasierte Synthese verteilter mechatronischer Systeme. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 351, Paderborn, 2016 – ISBN 978-3-942647-70-0

Bd. 352 WALL, M.: Systematik zur technologieinduzierten Produkt- und Technologieplanung. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 352, Paderborn, 2016 – ISBN 978-3-942647-71-7

Bd. 353 CORD-LANDWEHR, A.: Selfish Network Creation - On Variants of Network Creation Games. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 353, Paderborn, 2016 – ISBN 978-3-942647-72-4

Bd. 354 ANACKER, H.: Instrumentarium für einen lösungsmusterbasierten Entwurf fortgeschrittener mechatronischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 354, Paderborn, 2016 – ISBN 978-3-942647-73-1

Bd. 355 RUDTSCH, V.: Methodik zur Bewertung von Produktionssystemen in der frühen Entwicklungsphase. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 355, Paderborn, 2016 – ISBN 978-3-942647-74-8

Bd. 356 SÖLLNER, C.: Methode zur Planung eines zukunftsfähigen Produktportfolios. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 356, Paderborn, 2016 – ISBN 978-3-942647-75-5

Bd. 357 AMSHOFF, B.: Systematik zur musterbasierten Entwicklung technologieinduzierter Geschäftsmodelle. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 357, Paderborn, 2016 – ISBN 978-3-942647-76-2

Bezugsadresse:
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn

Zuletzt erschienene Bände der Verlagsschriftenreihe des Heinz Nixdorf Instituts

Bd. 358 LÖFFLER, A.: Entwicklung einer modellbasierten In-the-Loop-Testumgebung für Waschautomaten. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 358, Paderborn, 2016 – ISBN 978-3-942647-77-9

Bd. 359 LEHNER, A.: Systematik zur lösungsmusterbasierten Entwicklung von Frugal Innovations. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 359, Paderborn, 2016 – ISBN 978-3-942647-78-6

Bd. 360 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 12. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 8. und 9. Dezember 2016, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 360, Paderborn, 2016 – ISBN 978-3-942647-79-3

Bd. 361 PETER, S.: Systematik zur Antizipation von Stakeholder-Reaktionen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 361, Paderborn, 2016 – ISBN 978-3-942647-80-9

Bd. 362 ECHTERHOFF, O.: Systematik zur Erarbeitung modellbasierter Entwicklungsaufträge. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 362, Paderborn, 2016 – ISBN 978-3-942647-81-6

Bd. 363 TSCHIRNER, C.: Rahmenwerk zur Integration des modellbasierten Systems Engineering in die Produktentstehung mechatronischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 363, Paderborn, 2016 – ISBN 978-3-942647-82-3

Bd. 364 KNOOP, S.: Flachheitsbasierte Positionsregelungen für Parallelkinematiken am Beispiel eines hochdynamischen hydraulischen Hexapoden. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 364, Paderborn, 2016 – ISBN 978-3-942647-83-0

Bd. 365 KLIEWE, D.: Entwurfssystematik für den präventiven Schutz Intelligenter Technischer Systeme vor Produktpiraterie. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 365, Paderborn, 2017 – ISBN 978-3-942647-84-7

Bd. 366 IWANEK, P.: Systematik zur Steigerung der Intelligenz mechatronischer Systeme im Maschinen- und Anlagenbau. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 366, Paderborn, 2017 – ISBN 978-3-942647-85-4

Bd. 367 SCHWEERS, C.: Adaptive Sigma-Punkte-Filter-Auslegung zur Zustands- und Parameterschätzung an Black-Box-Modellen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 367, Paderborn, 2017 – ISBN 978-3-942647-86-1

Bd. 368 SCHIERBAUM, T.: Systematik zur Kostenbewertung im Systementwurf mechatronischer Systeme in der Technologie Molded Interconnect Devices (MID). Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 368, Paderborn, 2017 – ISBN 978-3-942647-87-8

Bd. 369 BODDEN, E.; DRESSLER, F.; DUMITRESCU, R.; GAUSEMEIER, J.; MEYER AUF DER HEIDE, F.; SCHEYTT, C.; TRÄCHTLER, A. (Hrsg.): Intelligente technische Systeme. Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 369, Paderborn, 2017 – ISBN 978-3-942647-88-5

Bd. 370 KÜHN, A.: Systematik zur Release-Planung intelligenter technischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 370, Paderborn, 2017 – ISBN 978-3-942647-89-2

Bd. 371 REINOLD, P.: Integrierte, selbstoptimierende Fahrdynamikregelung mit Einzelradaktorik. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 371, Paderborn, 2017 – ISBN 978-3-942647-90-8

Bezugsadresse:
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn