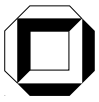

Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Fakultät für **Informatik**
Lehrstuhl für Programmiersysteme

Rechnergestützte Verbesserung von textbasierten Softwarespezifikationen mit Hilfe von Ontologien

Diplomarbeit am
Institut für Programmstrukturen und Datenorganisation
Lehrstuhl Programmiersysteme
Prof. Dr. Walter F. Tichy
Fakultät für Informatik
Universität Karlsruhe (TH)

von

Torben Brumm

Betreuer:

Prof. Dr. Walter F. Tichy

Betreuender Mitarbeiter:

Dipl.-Inform. Sven J. Körner

Tag der Anmeldung: 01. Februar 2009
Tag der Abgabe: 24. Juli 2009

Hiermit erkläre ich, dass ich diese Diplomarbeit selbstständig verfasst habe. Alle nicht von mir stammenden Inhalte sind durch Angabe der Quelle kenntlich gemacht.

Karlsruhe, den 24. Juli 2009

Torben Brumm

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

1. Einleitung

Die Qualität einer Softwarespezifikation hat einen großen Einfluss auf die Qualität der späteren Software. Unvollständige oder missverständliche Spezifikationen führen zu fehlerhafter Software. In dieser Arbeit wird vorgestellt, wie mit Hilfe von Ontologien Probleme in textuellen Spezifikationen identifiziert und Verbesserungsvorschläge generiert werden können, um die Qualität der Spezifikationen zu erhöhen.

Ein Großteil der Softwarespezifikationen wird in natürlicher Sprache erstellt. Formulierungen in natürlicher Sprache haben den Nachteil, dass sie unvollständig und missverständlich sein können. Experten müssen deshalb sicherstellen, dass die Wünsche des Kunden eindeutig in der Spezifikation formuliert werden. Nur so kann der Entwickler die Software so implementieren, wie der Kunde es sich gewünscht hat. Dazu stellt der Experte Nachfragen an den Kunden, um die Lücken und Missverständnisse in der Spezifikation zu beseitigen. Die Qualität und dementsprechend auch der Nutzen der Nachfragen hängen von den Fähigkeiten des Experten ab.

Unterstützung für den Experten durch ein Werkzeug wäre hilfreich. Dazu benötigt solch ein Werkzeug semantisches Wissen über die Spezifikationen, um Lücken und Missverständnisse zu finden. Ontologien können dieses semantische Wissen liefern. Das in dieser Arbeit vorgestellte Werkzeug RESI (*Requirements Engineering Specification Improver*) nutzt Ontologien, um Probleme in Spezifikationen zu finden. Somit ermöglicht es dem Experten, leichter Nachfragen an den Kunden stellen zu können.

RESI konzentriert sich dabei auf die Problemarten, die laut Rupp [58] mit der höchsten Priorität zu beseitigen sind. Es findet die Probleme, indem es geeignete Ontologien unter Angabe von Informationen aus der Spezifikation befragt. Durch das Wissen in den Ontologien werden die Probleme identifiziert und Verbesserungsvorschläge generiert. Somit kann RESI den Experten bei seiner Arbeit unterstützen, ihn aber nicht komplett ersetzen. Der Experte muss immer noch den persönlichen Kontakt mit dem Kunden suchen, um die Nachfragen zu stellen, die sich durch die Arbeit mit RESI ergeben haben.

Zuerst wird diese Arbeit im folgenden Kapitel 2 motiviert. In Kapitel 3 wird dargestellt, wo sich RESI im weiten Feld des Requirements Engineering befindet, welche Probleme

in Spezifikationen auftreten können und wie andere Arbeiten diese Probleme angehen. Wie diese Probleme mit Hilfe von Ontologien identifiziert und beseitigt werden können, steht in Kapitel 4. Die konkrete Implementierung von RESI wird in Kapitel 5 vorgestellt und in Kapitel 6 evaluiert. Die Zusammenfassung und der Ausblick in Kapitel 7 bilden den Abschluss dieser Arbeit.

2. Motivation

Die Anforderungsermittlung nimmt eine zentrale Rolle in der Softwareentwicklung ein. Sie ist der erste Schritt auf dem Weg zum fertigen Produkt, bei dem die Ideen des Kunden in eine schriftliche Form gebracht werden. Ihr Ergebnis ist das erste Dokument, das der Entwickler im Laufe der Softwareerstellung verwendet und auf dem der weitere Entwicklungsprozess aufbaut: die Spezifikation. Die Spezifikation stellt das wichtigste textuelle Bindeglied zwischen dem Kunden und dem Entwickler dar. Sie wird sowohl in der Kommunikation zwischen Kunde und Entwickler als auch als Grundlage für die Arbeit des Entwicklers vielfältig verwendet [55]. Später im Entwicklungsprozess erstellte Dokumente – wie zum Beispiel UML-Diagramme oder Quellcode – sind für den Kunden in den meisten Fällen nicht relevant.

Es ist darauf zu achten, dass die Spezifikation vollständig, korrekt und unmissverständlich für Kunde und Entwickler ist [1]. Fehler, die bereits in der Spezifikation auftreten, finden sich im gesamten Prozess wieder. Im schlimmsten Falle befinden sich diese Fehler auch im fertigen Produkt. Der Aufwand und die Kosten für die Beseitigung von Fehlern, die bereits in der Spezifikation vorhanden waren, potenziert sich im Laufe des Entwicklungsprozesses. [10, 48]. Darüber hinaus sind Fehler in der Spezifikation die häufigsten und die gefährlichsten Fehler, die im Softwareentwicklungsprozess auftreten [47].

Formale Methoden wie SCR [36], SDL [38] und LOTOS [37] dienen dazu, möglichst vollständige und eindeutige Spezifikationen zu erstellen, mit denen der Entwickler arbeiten kann. Diese Methoden haben jedoch den Nachteil, dass der Kunde lernen muss, die daraus resultierenden Spezifikationen zu verstehen. Ansonsten muss für ihn eine nicht formale Repräsentation der Spezifikation erstellt werden, bei der nicht sicher gestellt ist, dass sie inhaltlich mit der formalen Repräsentation für den Entwickler übereinstimmt [19]. Bis jetzt haben sich die formalen Methoden nicht durchsetzen können, ein Großteil der Spezifikationen wird in natürlicher Sprache erstellt [49]. Deshalb konzentriert sich diese Arbeit auf Spezifikationen in natürlicher Sprache. Sie ist für jeden ohne besonderes Training verständlich. Jeder kann in natürlicher Sprache formulieren, wenn auch nicht zwingend korrekt.

Wenn der Kunde die Spezifikation selbst formuliert, ist es wahrscheinlich, dass er Dinge

auslässt, die für ihn klar sind (*Curse of Knowledge* [35]), was häufig zu Inkonsistenzen und Widersprüchen führt. Eventuell verwendet der Kunde missverständliche Formulierungen, die für ihn eine andere Bedeutung haben als für den Entwickler. Beispielsweise kann der Kunde mit `Alle Lampen haben einen Lichtschalter` meinen, dass es einen gemeinsamen Lichtschalter für alle Lampen gibt, während der Entwickler denkt, dass jede Lampe ihren eigenen Lichtschalter erhält. Wenn der Entwickler die für ihn geltende Bedeutung umsetzt, entspricht die fertige Software zwar dem Wortlaut der Spezifikation, aber nicht den Vorstellungen des Kunden.

Deshalb werden Experten benötigt, die in Zusammenarbeit mit dem Kunden eine Spezifikation erstellen, die den Vorstellungen des Kunden entspricht [59] und dabei so vollständig, widerspruchsfrei und unmissverständlich wie möglich ist. Ausgangspunkt ist eine erste, noch fehlerhafte Liste von Anforderungen, die der Kunde allein nach seinen Vorstellungen erstellt hat. Möglicherweise waren andere Experten daran beteiligt, das initiale Dokument zu erstellen. Diese fehlerhafte Spezifikation überprüft der Experte auf Widersprüche, Unvollständigkeiten und Missverständnisse. Im Dialog mit dem Kunden räumt er die Probleme durch geschicktes Nachfragen aus.[1]

Wenn dieser Prozess durch ein softwarebasiertes System unterstützt wird, vereinfacht dies die Arbeit des Experten. Aktuelle Werkzeuge für die Anforderungsermittlung konzentrieren sich auf die Verwaltung und die Modellierung von Anforderungen, jedoch nicht auf das Beheben von Fehlern in der Spezifikation [70]. Des Weiteren fehlt bestehenden softwarebasierten Systemen die Möglichkeit, intelligente, auf den Text bezogene Nachfragen zu stellen. Dieses fehlende semantische Wissen kann von Ontologien zur Verfügung gestellt werden. Es dient als Grundlage für diese Arbeit, die als Ergebnis das Werkzeug RESI (*Requirements Engineering Specification Improver*) hat. RESI ist ein Dialogsystem für die Expertenanalyse von textuellen Anforderungen, das Ontologien nutzt.

[1] Selbstverständlich ist es auch möglich, dass der Experte keinen Zwischenschritt über eine fehlerhafte Spezifikation macht, sondern direkt bei der initialen Erstellung darauf achtet, die Spezifikation korrekt zu erstellen. Da es aber prinzipiell egal ist, ob der Experte jede Anforderung einzeln ermittelt und sie direkt im Dialog mit dem Kunden verbessert oder ob er zuerst alle Anforderungen ermittelt und sie anschließend im Dialog mit dem Kunden verbessert, wird der Einfachheit halber nur der zweite Fall betrachtet.

3. Verwandte Arbeiten

Requirements Engineering ist ein sehr weitläufiges Feld, wie Cheng und Atlee in *Research Directions in Requirements Engineering* [12] durch einen ausführlichen Überblick über die momentane Forschung belegen. Sie unterteilen die untersuchten Arbeiten dabei in zwei Dimensionen: Aufgabe und Technologie. Aufgaben sind Ermittlung, Modellierung, Anforderungsanalyse, Validierung & Verifikation sowie Anforderungsmanagement. Die Technologiedimension unterscheidet zwischen Notationen, Methoden (+ Strategien, Ratschläge) und Techniken (+ Analysen, Werkzeuge). Nach dieser Unterteilung ist RESI ein Werkzeug für die Anforderungsanalyse (siehe auch Abbildung 3.1).

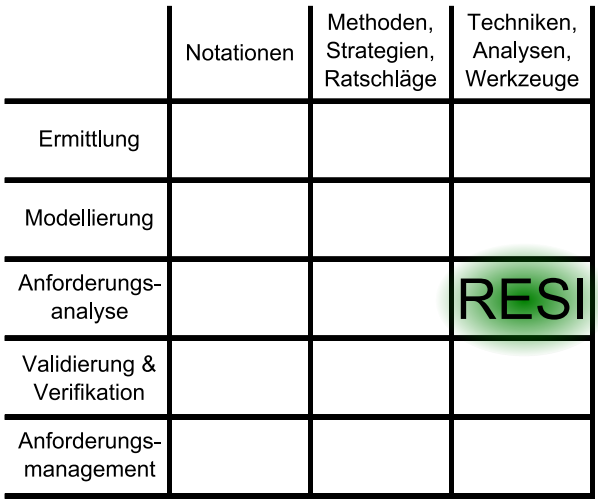

Abbildung 3.1.: RESI im Requirements Engineering nach Cheng und Atlee [12]

3.1. Eingliederung in ein größeres System

RESI ist Teil des Forschungsprojekts AutoModel [45], was ein Teil des SAL$_E$MX-Projekts [30] ist. SAL$_E$MX kann aus einer natürlichsprachlichen Spezifikation ein Spezifikationsmodell erstellen. Ein Überblick über SAL$_E$MX und AutoModel befindet sich in Abbildung 3.2.

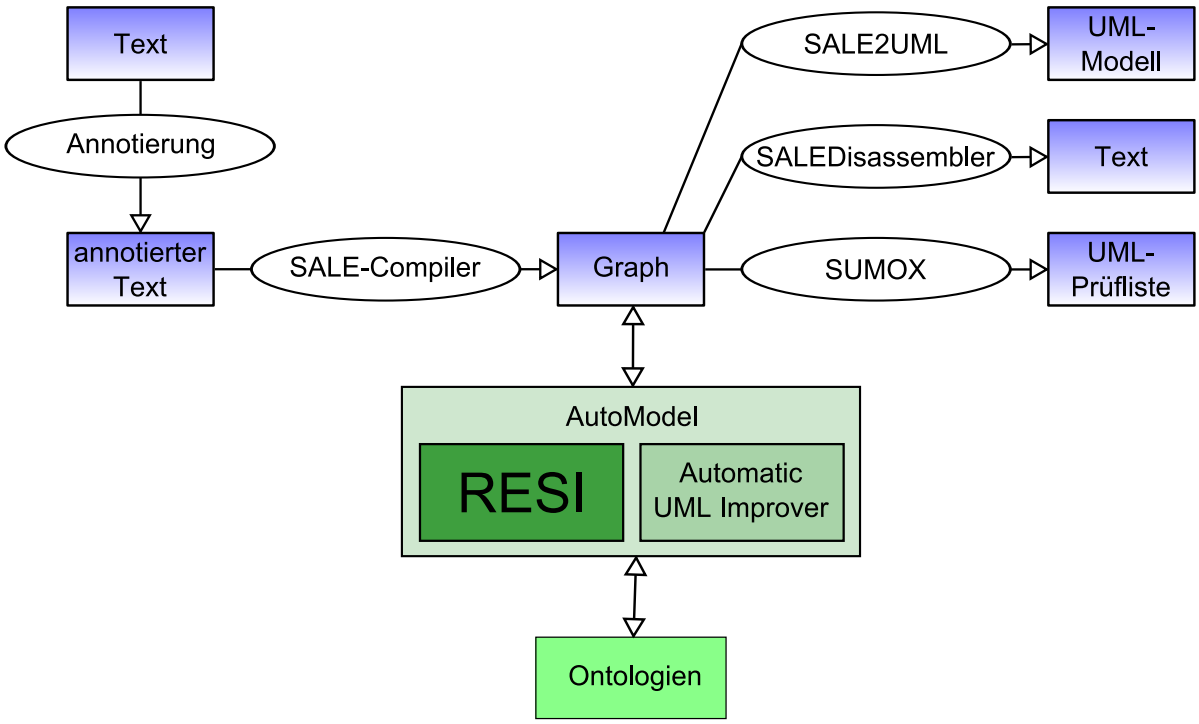

Abbildung 3.2.: SALEMX im Überblick

Zuerst wird der Text der Spezifikation mit thematischen Rollen [31] annotiert. Ein kleiner Teil der thematischen Rollen ist in Tabelle 3.1 dargestellt. Durch die thematischen Rollen werden in den Text semantische Informationen eingefügt, die die Automatisierung der Modellierung vereinfachen [33]. Dieser annotierte Text wird dann durch den SALE-Compiler in einen Graphen umgewandelt, um ihn für den weiteren Prozess leichter verarbeitbar zu machen.

Hier setzt AUTOMODEL an, um weitere Informationen in den Graphen (und damit die Spezifikation) einzufügen, die das zu erstellende Modell verbessern. AUTOMODEL nutzt Ontologien, um implizite Informationen zu gewinnen, die ein menschlicher Benutzer durch seinen gesunden Menschenverstand gewinnen würde. RESI dient dabei als Vorverarbeitungsschritt, indem es hilft, Probleme in der Ausgangsspezifikation zu beseitigen.

Die andere Komponente von AUTOMODEL, der *Automatic UML Improver* [46], automatisiert Entscheidungen, die ein Benutzer beim Erstellen eines UML-Diagramms treffen muss. So kann zum Beispiel die thematische Rolle `ACT` – die in den meisten Fällen durch ein Verb repräsentiert wird – als Funktion, als Relation oder als Zustandsübergang im UML-Modell dargestellt werden. Für jeden `ACT` ist aber nur eine der drei Formen korrekt. Der *Automatic UML Improver* übernimmt die Entscheidung, welche die korrekte Form

Tabelle 3.1.: Thematischen Rollen und ihre Bedeutung

Thematische Rolle	Erklärung
AG	Die Person oder Sache, die eine Handlung ausführt.
ACT	Die Handlung, die von einer Person oder Sache ausgeführt wird.
PAT	Die Person oder Sache, die von einer Handlung betroffen ist oder an der eine Handlung ausgeführt wird.
HAB	Die Habe; der Besitz; die Person oder Sache, die von einer anderen Person oder Sache besessen (auch kurzzeitig), erhalten oder weitergegeben wird.
POSS	Der (gegenwärtige) Besitzer eines Elements.

für den jeweiligen Einzelfall ist und vermerkt sie im Graphen.

Aus dem angereicherten Graphen kann mit Hilfe von SAL$_E$2UML automatisch ein UML-Modell generiert werden [29]. Mit Hilfe des SAL$_E$Disassembler kann aus dem durch AUTOMODEL angereicherten Graphen wieder eine natürlichsprachliche Spezifikation erstellt werden. Außerdem ist es möglich, aus dem Graphen mittels SUMO$_x$ [32] eine UML-Prüfliste zu erstellen. Mit Hilfe dieser Prüfliste können aus der Spezifikation manuell erstellte UML-Modelle auf ihre Korrektheit (im Sinne von Vollständigkeit und Richtigkeit) überprüft werden.

3.2. Probleme in Spezifikationen

Arbeiten zur Anforderungsermittlung fordern, dass Spezifikationen vollständig, korrekt und unmissverständlich sein sollen [1]. Es gibt jedoch nur wenige Abhandlungen darüber, welche Probleme im Einzelnen auftreten und was man konkret beim Erstellen von Spezifikationen beachten sollte.

Rupp beschreibt im Buchkapitel *Das SOPHIST-REgelwerk – Psychotherapie für Anforderungen* [58] wie man sogenannte sprachliche Defekte erkennt und beseitigt. Dazu stellt sie das *SOPHIST-**RE**gelwerk* vor, das auf dem Metamodell der Sprache des Therapieansatzes Neuro-Linguistisches Programmieren (NLP) basiert.

Sprachliche Defekte entstehen unabsichtlich durch zwei Prozesse: Einerseits transfor-

miert der Kunde die Realität – die im Bezug auf die Anforderungsermittlung die gewünschte Software darstellt – in ein Bild in seinem Kopf. Dabei filtert er für ihn unwichtige Dinge heraus und legt Schwerpunkte auf für ihn wichtige Dinge. Andererseits wandelt er das Bild in seinem Kopf in natürliche Sprache um, wenn er die Anforderungen kommuniziert. Auch hierbei finden – meist unbewusst – Umformungen und Filterungen statt. Rupp unterteilt sprachliche Defekte nach Bandler [3] in drei Kategorien: Tilgung, Generalisierung und Verzerrung.

„Tilgung (englisch: Deletion) ist ein Prozess, durch den wir unsere Aufmerksamkeit selektiv bestimmten Dimensionen unserer [im Moment möglichen] Erfahrungen zuwenden und andere ausschließen. [...] Tilgung reduziert die Welt auf Ausmaße, mit denen wir umgehen können.“ [3]
Ein Beispiel für Tilgung ist das Filtern eines Stimmengewirrs in einem Raum voller Leute, damit der Gesprächspartner verstanden werden kann. Tilgung in Spezifikationen kann dafür sorgen, dass wichtige Informationen verloren gehen.

„Generalisierung (englisch: Generalization) ist der Prozess, durch den Elemente oder Teile eines persönlichen Modells von der ursprünglichen Erfahrung abgelöst werden, um dann die gesamte Kategorie, von der diese Erfahrung ein Beispiel darstellt, zu verkörpern.“[3]
Beispielsweise stellt eine heiße Herdplatte, die man angefasst hat und wegen derer man keine Herdplatten mehr anfasst, eine Generalisierung dar. Durch Generalisierung können in Spezifikationen Anforderungen entstehen, bei denen Ausnahmefälle fehlen.

„Verzerrung (englisch: Distortion) ist der Prozess, der es uns ermöglicht, in unserer Erfahrung sensorischer Einzelheiten eine Umgestaltung vorzunehmen.“ [3]
Eine Verzerrung findet zum Beispiel dann statt, wenn man von einer falschen Entscheidung (einem einzelnen Ereignis) spricht, obwohl es ein andauernder Entscheidungsprozess war. In Spezifikationen können durch Verzerrung Sachverhalte falsch dargestellt werden.

Rupp gibt für die drei Arten der sprachlichen Defekte jeweils mehrere Manifestationen an, von denen einige in Kapitel 4 ausführlich vorgestellt werden.

Im Buchkapitel *Ambiguity in Requirements Specification* [7] geben Berry und Kamsties einen Überblick über Mehrdeutigkeiten in Spezifikationen. Sie erklären, wie wichtig es ist, dass man Mehrdeutigkeiten beseitigt, geben Beispiele und erklären Möglichkeiten, wie man die Probleme, die durch Mehrdeutigkeiten entstehen, vermeiden kann.

In *From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity - A Handbook* [9] stellen Berry et al. die möglichen Mehrdeutigkeiten in Spezifikationen (und

Verträgen) ausführlicher vor. Sie zeigen anhand von Beispielen aus realen Dokumenten Probleme auf und geben Lösungsmöglichkeiten. Unter den vorgestellten Mehrdeutigkeiten ist zum Beispiel das englische Wort *all*, das viele Gefahren birgt. Es ist oftmals nicht *jeder* aus der mit *all* quantifizierten Menge gemeint, sondern es existieren Ausnahmen, die der Autor nicht bedacht hat. So beachtet `All persons have a unique national insurance number` nicht, dass es Menschen ohne oder mit mehr als einer Sozialversicherungsnummer gibt. Dieses Problem wird auch in *The Dangerous 'All' in Specifications* [6] von Berry und Kamsties behandelt.

Ein weiteres Problem mit *all* (und mit Plural allgemein) ergibt sich, wenn die im Plural stehende Menge etwas besitzt. Besitzt jeder der Menge ein eigenes Objekt oder gibt es nur ein gemeinsames für die Gruppe? Bei `All lights in the room are connected to a single switch` ist nicht klar, wie viele Schalter existieren: Genau einer oder einer pro Lampe? Außer in ihrem Handbuch wird dieses Problem von Berry und Kamsties auch in *The Syntactically Dangerous All and Plural in Specifications* [8] beschrieben.

Unter den vielen anderen Mehrdeutigkeiten, die im Handbuch behandelt werden, finden sich auch die Probleme, die durch falsche Positionierung von *only*, *also* und *even* oder durch die Verwendung von *a*, *all*, *any*, *each*, *one*, *some* und *the* als Quantoren ergeben.

3.3. Lösungen für diese Probleme

Um diese und ähnliche Probleme zu beheben, gibt es verschiedene Lösungsansätze. Eine Möglichkeit sind Inspektionen der Spezifikation, die auf der Idee der Design- und Codeinspektionen von Fagan [24] basieren. Bereits seit mehreren Jahrzehnten wird diese Technik für Spezifikationen eingesetzt [2]. Vorhandene Inspektionstechniken umfassen entweder die individuelle frei wählbare Fehlersuche (*Ad-hoc*) oder basieren auf Prüflisten[1] oder Szenarios[2] [54].

Kamsties et al. stellen in *Detecting Ambiguities in Requirements Documents Using Inspections* [41] eine eigens entwickelte Inspektionstechnik für Spezifikationen vor. Die Technik ist eine Kombination von Prüflisten und Szenarios, weil bisher vorhandene Prüflisten nicht genug Akzeptanz erreicht haben oder zu unspezifisch beziehungsweise zu

[1] Bei Prüflisten arbeitet der Inspektor eine Liste von Fragen ab, um Fehler zu finden.

[2] Der Inspektor versetzt sich in ein bestimmtes Szenario (z.B. liest er den Text aus der Perspektive eines Testers), um Fehler zu finden.

eindimensional sind [43]. Sie weisen darauf hin, dass es nicht möglich ist, mit einer realistischen Menge an Ressourcen alle Mehrdeutigkeiten in Spezifikationen zu finden. Sie zeigen anhand eines Experiments, dass mit Hilfe ihrer Technik Mehrdeutigkeiten besser gefunden werden als durch die Umwandlung in eine (semi-)formale Repräsentation wie SCR oder UML. Die Erfolgsquote der Umwandlung in formale Repräsentationen wurde in einer früheren Studie ermittelt [42]. In dieser wird gezeigt, dass formale Ansätze nicht alle Fehler in Spezifikationen finden können.

Eine weitere Möglichkeit, Fehler in Spezifikationen zu vermeiden, ist, die natürliche Sprache, in der die Spezifikation verfasst wird, einzuschränken. Oftmals werden dafür Muster verwendet, die vorschreiben, nach welcher Struktur Sätze geschrieben werden müssen, um einen bestimmten Sinn zu haben. Durch den genau definierten Sinn von Satzkonstrukten werden Mehrdeutigkeiten vermieden. Ein bekannter Vertreter dieses Ansatzes ist *Attempto Controlled English* (ACE) von Fuchs et al. [27]. ACE schränkt die Sprache zusätzlich darin ein, dass alle Verben nur in der dritten Person Singular und im aktiven Präsens stehen dürfen. Eine weitere Einschränkung – unter vielen anderen – ist es, dass nur Wörter aus einem definierten Vokabular erlaubt sind. Durch die ACE-Erweiterung von Schwertel [60] können auch Mehrdeutigkeiten im Plural kontrolliert werden, wodurch ACE flexibler wird.

Denger et al. stellen in *Higher Quality Requirements Specifications through Natural Language Patterns* [20] ihren musterbasierten Ansatz vor, um Mehrdeutigkeiten in Spezifikationen zu vermeiden. Sie konzentrieren sich dabei auf eingebettete Systeme. Nach einer Evaluierung ihres Ansatzes weisen sie darauf hin, dass es schwierig sei, vorhandene Spezifikationen in eine musterbasierte Form zu bringen.

Ein weiteres System, das eine eingeschränkte Sprache verwendet, ist Propel [61]. Bei Propel wird parallel ein endlicher Zustandsautomat und eine natürlichsprachliche Spezifikation entwickelt. Die natürlichsprachliche Spezifikation ist in einer durch Muster eingeschränkten Sprache formuliert (*Disciplined Natural Language*). Beide Repräsentationen sind äquivalent, müssen allerdings auch konsistent gehalten werden. Somit hat man gleichzeitig eine formale und eine nicht formale Repräsentation der selben Spezifikation.

Die Einschränkung der natürlichen Sprache hat aber einen ähnlichen Nachteil wie formale Spezifikationen: Das entsprechende System muss erst erlernt werden. Außerdem kann es nicht sofort auf bestehende Spezifikationen angewandt werden. Bestehende Spezifikationen können auf ihre Qualität und mögliche Probleme untersucht werden und anhand der ermittelten Probleme verbessert werden.

Davis et al. stellen in *Identifying and measuring quality in a software requirements specification* [18] 24 Kriterien vor, die die Qualität einer Spezifikation bestimmen. Für 18

dieser Kriterien geben sie Metriken an, um konkrete Maßzahlen zu bestimmen und somit die Qualität von Spezifikationen zu ermitteln. Sie weisen darauf hin, dass es keine perfekte Spezifikation geben kann, weil sich die Qualitätskriterien gegenseitig beeinflussen. So kann man beispielsweise zur Erhöhung der Lesbarkeit Erklärungen mehrfach einbauen, allerdings erhöht das auch die (unerwünschte) Redundanz.

Wilson et al. untersuchen in *Automated Analysis of Requirement Specifications* [72] natürlichsprachliche Spezifikationstexte auf das Vorkommen bestimmter Ausdrücke. Durch Kategorisierung dieser Ausdrücke werden Metriken gewonnen, die als Qualitätsindikatoren für die Spezifikation dienen sollen, indem sie Maßzahlen für Vollständigkeit, Konsistenz etc. darstellen.

Chantree et al. beschäftigen sich in *Identifying Nocuous Ambiguities in Natural Language Requirements* [11] mit Mehrdeutigkeiten in Spezifikationen. Vorhandene Mehrdeutigkeiten sollen automatisch gefunden und manuell vom Spezifikationsersteller beseitigt werden. Sie unterscheiden schädliche und unschädliche Mehrdeutigkeiten. Bei unschädlichen Mehrdeutigkeiten wird von jedem die gleiche Bedeutung angenommen, sodass diese Mehrdeutigkeiten praktisch keine Relevanz haben, während schädliche Mehrdeutigkeiten für unterschiedliche Leser unterschiedliche Bedeutungen haben und dementsprechend Probleme hervorrufen können. Die schädlichen werden noch weiter unterteilt in anerkannte Mehrdeutigkeiten, bei denen der Leser merkt, dass eine Mehrdeutigkeit vorliegt und nicht anerkannte Mehrdeutigkeiten, bei denen der Leser denkt, dass es nur eine Interpretation gibt, obwohl mehrere möglich sind. Die Autoren konzentrieren sich in ihrer Arbeit auf Mehrdeutigkeiten im Zusammenhang mit *und*- und *oder*-Verknüpfungen wie zum Beispiel `Display categorized instructions and documentation`, bei dem nicht klar ist, ob `categorized` auch zu `documentation` gehört oder nicht. Sie entwickeln mehrere Heuristiken, um die unschädlichen unter allen vorhandenen Mehrdeutigkeiten zu identifizieren. Diese Mehrdeutigkeiten müssen dem Spezifikationsersteller nicht mehr zur Beseitigung vorgelegt werden. Überprüft wird die Qualität der Heuristiken durch Abgleich mit Ergebnissen, die menschliche Inspektoren ermittelt haben.

Fabbrini et al. stellen in *The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool* [23] ihr Werkzeug namens QuARS (Quality Analyzer of Requirement Specification) vor. QuARS untersucht Spezifikationen in natürlicher Sprache nach bestimmten Wörtern, die als Indikatoren für Probleme dienen. Entsprechend der Auftrittshäufigkeit dieser Indikatoren werden dann die Sätze als problematisch oder nicht problematisch eingestuft, so dass der Benutzer die problematischen Sätze nachbessern kann. Bei den identifizierbaren Problemen handelt es sich um Ausprägungen von Mehrdeutigkeiten, Unvollständigkeiten, Inkonsistenzen und Verständlichkeitsdefiziten. Berry et al. stellen in *A New Quality Model for Natural Language Requirements Specifications* [5] Erweiterungen für QuARS vor, um mehr

Probleme (insbesondere andere Ausprägungen von Mehrdeutigkeiten) identifizieren zu können.

QuARS und andere ähnliche Werkzeuge werden in *Application of Linguistic Techniques for Use Case Analysis* von Fantechi et al. [25] als Basis für eigene Metriken verwendet. Diese Metriken wenden sie dann auf Anwendungsfälle in natürlicher Sprache an, um deren Qualität zu bestimmen.

Wenn Spezifikationen als qualitativ hochwertig evaluiert wurden, kann man sie zum Verbessern neuer Spezifikationen verwenden. Pisan tut dies in *Extending Requirement Specifications Using Analogy* [53], indem bereits vorhandene Spezifikationen zur Vervollständigung einer neuen Spezifikation benutzt werden. Es wird versucht, in den alten Spezifikationen analoge Anforderungen zu denen zu finden, die in der neuen Spezifikation vorhanden sind. Andere verwandte Anforderungen aus den alten Spezifikationen werden dann über die gleiche Analogie wieder auf die neue Spezifikation abgebildet, um mögliche neue Anforderungen zu ermitteln, die anschließend manuell evaluiert werden müssen.

3.4. Ontologien im Requirements Engineering

Anstatt eine ältere Spezifikation zur Verbesserung einer neuen zu verwenden, ist es naheliegend, ein allgemeineres Modell zu verwenden, das sich nicht nur auf eine einzelne Spezifikation bezieht.

Kaiya und Saeki verwenden in *Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach* [39] und *Using Domain Ontology as Domain Knowledge for Requirements Elicitation* [40] Domänenontologien, um Spezifikationen zu verbessern. Dabei werden Anforderungen aus der Spezifikation auf Konzepte der Domänenontologie abgebildet und aus diesen Abbildungen Maßzahlen für Korrektheit, Vollständigkeit, Konsistenz und Eindeutigkeit festgelegt. Wenn zum Beispiel Anforderungen auf zwei Konzepte abgebildet werden, die sich gegenseitig ausschließen, ist die Spezifikation inkonsistent. Mit Hilfe der Maßzahlen sollen Spezifikationsersteller erkennen, wie die Spezifikation verbessert werden sollte. Für diesen Ansatz wird eine gute Domänenontologie benötigt, die die Autoren aus Spezifikationsdokumenten aus der gleichen Domäne gewinnen wollen.

Kiyavitskaya et al. stellen in *Requirements for tools for ambiguity identification and measurement in natural language requirements specifications* [44] vor, wie sie das Problem der Mehrdeutigkeiten in Spezifikationen mit Hilfe von Softwarewerkzeugen beheben würden.

Dabei teilen sie das Problem in zwei Schritte auf: Im ersten Schritt werden die mehrdeutigen Sätze bestimmt, bevor im zweiten Schritt dem Nutzer mitgeteilt wird, warum der Satz mehrdeutig ist, um ihm zu ermöglichen, die Mehrdeutigkeit zu beseitigen. Für den ersten Schritt wird in dem Artikel ein selbst entwickeltes Werkzeug vorgestellt, das mit Hilfe eines Wörterbuchs die Anzahl von Bedeutungen eines Wortes bestimmt und daraus Maßzahlen berechnet. Diese Maßzahlen stellen einen Indikator für die Mehrdeutigkeit eines Satzes dar. Die berechneten Werte sind nicht zwingend ein guter Indikator für Mehrdeutigkeit, da semantische Mehrdeutigkeiten nicht berücksichtigt werden. Außerdem kann es sein, dass manche Wörter in einem spezifischen Kontext (wie er in einer Spezifikation gegeben ist) weniger Bedeutungen haben als durch das Wörterbuch bestimmt. Für den zweiten Schritt wird kein konkretes Werkzeug vorgestellt, sondern es werden nur Voraussetzungen und Probleme für ein solches Werkzeug gezeigt. RESI verwendet ein ähnliches Verfahren zum Ermitteln und Beseitigen von Mehrdeutigkeiten, das im Abschnitt 5.3.4 vorgestellt wird.

3.5. Ontologien

RESI nutzt verschiedene Ontologien, um semantisches Wissen aus Spezifikationen zu gewinnen. Ontologien haben unterschiedliche Schwerpunkte und sind mit unterschiedlichen Zielen entwickelt worden. Die von RESI verwendeten Ontologien werden im Folgenden vorgestellt.

Die Entwicklung der ausschließlich englischen Ontologie Cyc [14] (vom englischen *en**cyc**lopedia*) wurde 1984 von Dr. Douglas Lenat begonnen. Seit 1994 wird das Projekt von der eigens dafür gegründeten Cycorp, Inc. fortgeführt. Cyc beschreibt Konzepte aus der realen Welt und ihre Beziehungen untereinander in Form von Zusicherungen. Diese Zusicherungen werden in einer eigenen prädikatenlogischen Sprache (CycL [17]) formuliert und manuell eingegeben. Cyc besteht aus fast 3 Millionen Zusicherungen und über 300.000 Konzepten. Die komplette Ontologie ist für wissenschaftliche Zwecke als ResearchCyc [16] zugänglich, der Allgemeinheit steht die eingeschränkte Variante OpenCyc [15] zur Verfügung.

WordNet [52] ist eine lexikalische Datenbank für die englische Sprache. Sie wird seit 1985 am Cognitive Science Laboratory der Princeton University entwickelt und umfasst momentan rund 150.000 Wörter. WordNet unterscheidet Substantiv, Verb, Adjektiv und Adverb und bietet für die einzelnen Wortarten unterschiedliche Relationen. So können zum Beispiel Substantive als Synonyme oder Antonyme deklariert oder für Adverbien die entsprechenden Adjektive angegeben sein. Mit Hilfe von WordNet können auch Grundformen von Verben und Substantiven ermittelt werden.

ConceptNet [34] ist eine Ontologie, die seit 2000 im Rahmen des *Open Mind Common Sense*-Projekts entwickelt wird. Sie besteht aus Konzepten der realen Welt, die über Relationen miteinander in Verbindung stehen. Die Relationen werden aus Texten über Mustererkennung eingelesen oder durch Benutzer über Formulare eingegeben. Anschließend werden sie evaluiert, indem Benutzer die Relationen als wahr oder falsch bewerten und aus der Anzahl dieser Bewertungen Qualitätsmaßzahlen berechnet werden. ConceptNet enthält die englischen und die portugiesischen Daten (insgesamt über 1 Million Aussagen) aus dem *Open Mind Common Sense*-Projekt. In dessen Rahmen werden aber auch Daten auf koreanisch, japanisch, holländisch, italienisch, spanisch und französisch erhoben.

Die Ontologie YAGO [64] kombiniert WordNet [52] mit dem Inhalt der englischen Wikipedia [71]. Dazu wurden die Wikipedia-Seiten mit den WordNet-Begriffen verknüpft und mit Hilfe von Algorithmen (zum Beispiel durch Verwendung der Wikipedia-Kategorien, die den einzelnen Seiten zugewiesen sind) und manuellen Anpassungen in Relationen gesetzt. Dadurch enstanden rund 5 Millionen Relationen. Darüber hinaus enthält YAGO circa 40 Millionen Kontext-Relationen. Eine Kontext-Relation wurde erstellt, wenn auf einer Wikipedia-Seite ein Link zu einer anderen Wikipedia-Seite (und damit einem anderen Begriff in YAGO) existierte.

4. Probleme in Spezifikationen

In diesem Kapitel werden Probleme in natürlichsprachlichen Spezifikationen vorgestellt, die bei der Entwicklung von RESI betrachtet wurden, und an jeweils einem Beispiel verdeutlicht. Diese Probleme wurden auf Lösbarkeit mit Hilfe von Ontologien untersucht, indem linguistische Elemente auf Begriffe der Ontologien abgebildet wurden. Es wurden Möglichkeiten untersucht, wie die Ontologien das Wissen zur Verfügung stellen können, das benötigt wird, um die Probleme zu identifizieren und zu beheben. Die daraus resultierenden konzeptuellen Lösungen finden sich ebenfalls in diesem Kapitel. In Abschnitt 5.3 werden die konkreten Implementierungen der Problemlösungen vorgestellt.

4.1. Hochpriore Probleme

Nach Rupp [58] sind folgende Probleme mit hoher Priorität zu beseitigen:

1. Nominalisierung
2. Unvollständig spezifizierte Prozesswörter
3. Substantive ohne Bezugsindex
4. Unvollständig spezifizierte Bedingungen
5. Modaloperatoren der Notwendigkeit

4.1.1. Nominalisierung

Problem

Bei einer Nominalisierung kann ein komplexer Prozess in einem einzelnen Substantiv zusammengefasst sein. Dadurch gehen Informationen, die für die Beschreibung des Prozesses wichtig sind, verloren. [58]

Beispiel

```
Nach der Anmeldung wird der Benutzer zu seiner persönlichen
   Seite weitergeleitet.
```

`Anmeldung` ist ein komplexer Vorgang, der ausführlicher beschrieben werden muss.

Lösung

Mit Hilfe von Ontologien wird überprüft, ob Substantive eine Nominalisierung darstellen. Die Ontologie schlägt dann vor, welches Vollverb stattdessen verwendet werden sollte. In Abschnitt 5.3.1 wird die Regel Überprüfe auf Nominalisierungen beschrieben, mit deren Hilfe RESI Nominalisierungen aufspürt.

4.1.2. Unvollständig spezifizierte Prozesswörter

Problem

Prozesswörter erfordern oft mehr als ein Argument[1], um vollständig spezifiziert zu sein. Wenn nicht alle Argumente für ein Prozesswort angegeben werden, können wichtige Informationen für die vollständige Prozessbeschreibung fehlen. Ein häufig anzutreffender Spezialfall dieses Problems ist die Verwendung von Passivsätzen, bei denen die ausführende Person oder Einheit fehlt. [58]

[1] Argumente sind Phrasen, die in syntaktischem Zusammenhang mit dem Prozesswort stehen, zum Beispiel Subjekt und Objekt.

Beispiel

```
Benutzername und Passwort werden eingegeben.
```

Wo werden diese Daten eingegeben? Wer gibt diese Daten ein?

Lösung

Aus einer Ontologie erfährt man, welche Argumente zu einem Prozesswort gehören. Wenn man diese Argumente mit dem tatsächlichen Inhalt des Satzes abgleicht, findet man heraus, welche Argumente noch fehlen und dementsprechend noch spezifiziert werden müssen. RESI führt diese Überprüfung mit Hilfe der Regel Vervollständige Prozesswörter durch, die in Abschnitt 5.3.2 beschrieben wird.

4.1.3. Substantive ohne Bezugsindex

Problem

Substantive, die unvollständig oder nicht spezifiziert sind, werden von Linguisten als Substantive ohne oder mit zu wenig Bezugsindizes bezeichnet. Wenn sich ein Substantiv auf eine Menge bezieht, ist nicht immer klar, auf welchen Teil dieser Menge (die ganze Menge, einzelne Mitglieder etc.) es sich bezieht. Damit ist dieses Substantiv nicht ausreichend spezifiziert. Beispiele dafür sind die falsche Verwendung von Artikeln statt Quantoren vor Substantiven oder die Verwendung von Universalquantoren ohne auf Ausnahmefälle zu achten. [58, 6]

Beispiel

```
Die persönlichen Daten können auf der persönlichen Seite
   eingesehen werden.
```

Welche persönlichen Daten? Alle oder nur die wichtigen? Und welche persönliche Seite? Können Peters persönliche Daten auf Pauls persönlicher Seite eingesehen werden?

Lösung

Durch eine Ontologie wird die korrekte Bedeutung eines Artikels oder Quantors ermittelt. Diese wird dann mit der gewünschten Bedeutung verglichen, um herauszufinden, ob Anpassungen nötig sind. RESI benutzt die Regel Überprüfe Artikel & Quantoren, um Artikel und Quantoren zu überprüfen. In Abschnitt 5.3.3 wird sie beschrieben.

4.1.4. Unvollständig spezifizierte Bedingungen

Problem

Mit Hilfe von Konditionalsätzen wird das Verhalten beschrieben, das unter einer bestimmten Voraussetzung eintritt (*wenn...dann...*). Dabei fehlt häufig die Beschreibung des Verhaltens, das eintritt, wenn die Voraussetzung nicht gegeben ist (*sonst...*). [58]

Beispiel

```
Im Falle einer erfolgreichen Anmeldung wird der Anwender zu
   seiner persönlichen Seite weitergeleitet.
```

Was passiert, wenn die Anmeldung nicht erfolgreich war?

Lösungsansatz

Eine Möglichkeit ist es, mit Hilfe von Ontologien Signalwörter zu bestimmen, die eine Bedingung einleiten, um dann diese Bedingungen auf Vollständigkeit zu überprüfen.

4.1.5. Modaloperatoren der Notwendigkeit

Problem

Modaloperatoren der Notwendigkeit werden dazu verwendet, um ein Verhalten zu beschreiben, das im Normalfall eintritt (etwas *soll* geschehen). Damit muss hierbei (analog zu den unvollständig spezifizierten Bedingungen) angegeben werden, was im Ausnahmefall geschehen soll. [58]

Beispiel

```
Die Anmeldedaten sollen überprüft werden, indem die
   Datenbank befragt wird.
```

Was passiert, wenn die Datenbank nicht erreichbar ist? Wird gewartet oder abgebrochen?

Lösungsansatz

Es ist möglich, mit Hilfe einer Ontologie alle Modalverben der Notwendigkeit zu ermitteln und anschließend auf vorhandene Ausnahmefälle zu überprüfen.

4.2. Weitere Probleme

Mit Hilfe von Ontologien lassen sich auch andere Probleme in Spezifikationen beheben. Zwei weitere Probleme, die ebenfalls mit RESI behoben werden, werden hier vorgestellt.

4.2.1. Mehrdeutige Begriffe (Polysemie)

Problem

Begriffe, die unterschiedlich interpretiert werden können, weil sie mehrere Bedeutungen haben, können dazu führen, dass der Leser nicht weiß, welche Bedeutung gemeint ist. Im schlimmsten Fall geht der Leser ohne darüber nachzudenken davon aus, dass seine Interpretation des Begriffs richtig ist, obwohl sie nicht der gemeinten Bedeutung entspricht. [57, 21]

Beispiel

```
Läufer
```

Der Sportler? Der Teppich? Die Schachfigur?

Lösung

In einer Ontologie wird nachgeschlagen, welche möglichen Interpretationen für ein einzelnes Wort existieren (wie es auch Kiyavitskaya et al. tun [44]). Dadurch wird überprüft, ob noch Bedeutungen existieren, die nicht bedacht wurden und die ein Leser in den Satz hinein interpretieren könnte. Die in Abschnitt 5.3.4 vorgestellte Regel Überprüfe auf mehrdeutige Wörter übernimmt diese Funktion in RESI.

4.2.2. Mehrere Begriffe für das gleiche Objekt

Problem

Mehrere Begriffe, die sich auf das gleiche Objekt beziehen, sorgen für Verwirrung.

Beispiel

```
Jeder Anwender hat eine persönliche Seite, auf der die
   persönlichen Daten des Benutzers dargestellt werden.
```

Ist der `Anwender` und der `Benutzer` äquivalent? Oder haben die beiden unterschiedliche Eigenschaften?

Lösung

Durch eine Ontologie werden ähnliche (und synonyme) Wortpaare im Text ermittelt. Durch Ersetzen des einen Begriffs durch sein Synonym wird die Anzahl der Begriffe für ein Objekt gesenkt. RESI nutzt dafür die Regel Überprüfe auf Wörter gleicher Bedeutung, die in Abschnitt 5.3.5 präsentiert wird. Auch ist es möglich, einen neuen Begriff einzuführen, der die beiden ähnlichen Begriffe unter sich vereint oder einen der Begriffe als Oberbegriff für den anderen zu definieren (beides nach dem Prinzip der Oberklasse aus der objektorientierten Programmierung).

5. Implementierung

In diesem Kapitel wird RESI im Detail vorgestellt. RESI ist ein Werkzeug, das Probleme, die im vorherigen Kapitel erläutert wurden, mit Hilfe von Ontologien behebt. Zuerst wird die grundlegende Funktionalität von RESI erklärt, bevor der interne Aufbau beleuchtet wird. Nach einer genauen Funktionsbeschreibung, wie im Einzelnen die Probleme entdeckt und behandelt werden, folgen Implementierungsdetails und Erweiterungsmöglichkeiten von RESI. Abschließend wird der externe Server beschrieben, auf dem die Ontologien installiert wurden.

5.1. Grundlegende Funktionalität

RESI untersucht Spezifikationen auf Probleme, die in Kapitel 4 beschrieben wurden. Dabei nutzt es Ontologien, um diese Probleme zu identifizieren. Durch Interaktion mit dem RESI-Benutzer können die Probleme noch genauer eingegrenzt und anschließend in der Spezifikation markiert werden. Dabei geht RESI in vier Schritten vor, wie es in Abbildung 5.1 gezeigt wird.

5.1.1. Import der Spezifikation

Zuerst wird eine Spezifikation importiert, damit sie mit Hilfe von RESI verbessert werden kann. Als Beispiel importieren wir folgenden Satz aus einer fiktiven Spezifikation für die Software eines Logistikunternehmens. Der Satz ist englisch, weil die Ontologien, die RESI verwendet, nur die englische Sprache abdecken (siehe auch Abschnitt 6.5).

```
Every pallet is returned after transport.
```

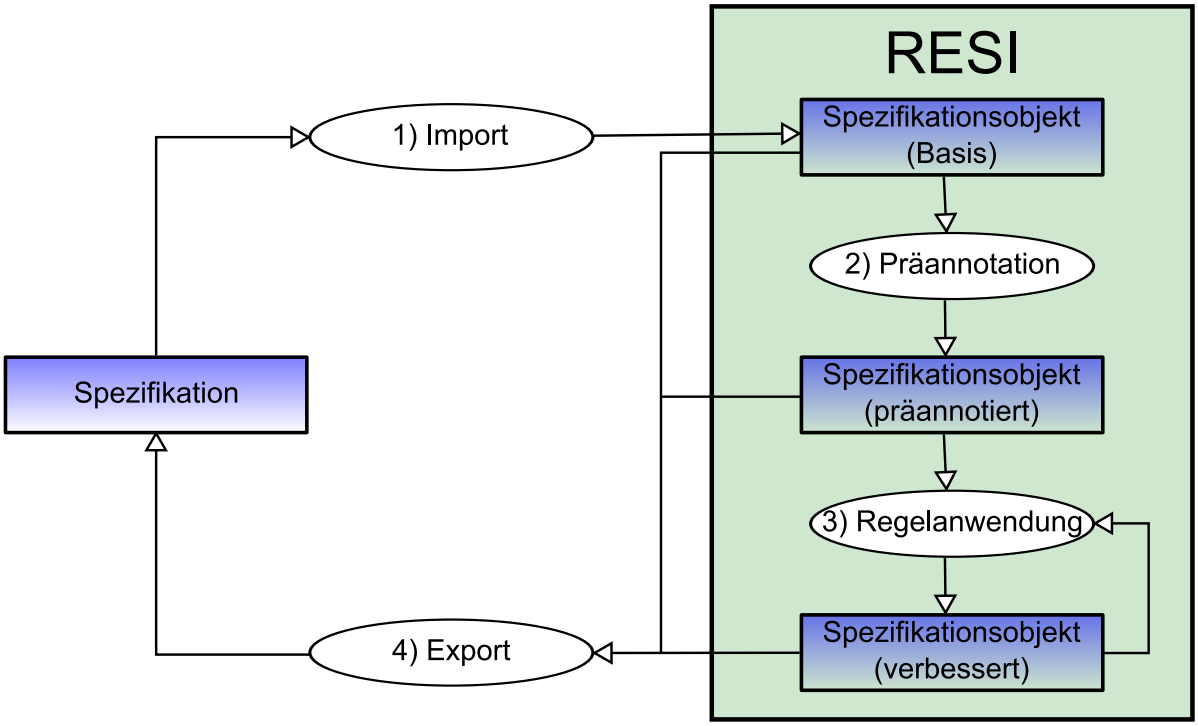

Abbildung 5.1.: Ablauf von RESI

5.1.2. Präannotation

Für den dritten Schritt (Regelanwendung) benötigt RESI weitere Informationen über die Wörter der Spezifikation. Deshalb annotiert es in einer Vorverarbeitungsstufe jedes Wort mit seiner Wortart entsprechend dem Penn Treebank Tagset [50]. Für Verben und Substantive ermittelt RESI zusätzlich die zugehörige Grundform. Die Informationen werden vollautomatisch mit Hilfe von Taggern ermittelt und können anschließend manuell angepasst werden.

```
Every[DT] pallet[NNP|pallet] is[VBZ|be] returned[VBN|return]
    after[IN] transport[NN|transport].
```

Im Beispiel werden die Wortart-Tags in rot und die Grundformen in blau dargestellt. Tags von Verben beginnen mit *VB*, die von Substantiven mit *NN*. Artikel und Quantoren werden durch *DT* (für *determiner*) dargestellt, *IN* steht für Präpositionen oder unterordnende Konjunktionen.

5.1.3. Regelanwendung

Hinweise auf Probleme in Spezifikationen werden von RESI durch sogenannte Regeln gefunden. Dabei ist eine Regel für eine Art von Problem zuständig. Jede Regel kann unabhängig auf eine präannotierte Spezifikation angewandt werden. Regeln stellen Anfragen an Ontologien, wobei der RESI-Benutzer entscheidet, welche Ontologie befragt werden soll. Während der Regelanwendung kann der RESI-Benutzer Markierungen mit Kommentaren in der Spezifikation setzen, um später problematische Sätze leichter wieder zu finden und sie in Absprache mit dem Kunden anzupassen.

Beispielsweise würde die Regel Überprüfe auf mehrdeutige Wörter unter Verwendung der Ontologie ResearchCyc für unseren Beispielsatz unter Anderem feststellen, dass es für `transport` vier verschiedene Bedeutungen (*Conveyance*, *Transmitting-Something*, *TransportationDevice* und *TransportationEvent*; Erklärungen siehe Anhang C) gibt. Der RESI-Benutzer kann das Wort als mehrdeutig markieren und später in Absprache mit dem Kunden ein besseres, nicht mehrdeutiges Wort suchen. Weitere – detailliertere – Beispiele zur Regelanwendung befinden sich bei den genauen Regelbeschreibungen in Abschnitt 5.3.

Regeln können beliebig oft angewandt werden, um vorherige Ergebnisse zu bearbeiten. Sie müssen nicht auf eine komplette präannotierte Spezifikation angewandt werden, sondern es können bei Bedarf Teile der Spezifikation ausgelassen werden.

5.1.4. Export der Spezifikation

Spezifikationen werden nach der Bearbeitung durch RESI wieder in ihr Ursprungsformat exportiert. Dabei werden alle Informationen, die durch RESI hinzugefügt wurden, ebenfalls mitgespeichert. Eine exportierte Spezifikation kann später wieder importiert werden, um sie mit RESI weiter zu verbessern. Ein Export kann direkt nach dem Import (obwohl es nicht sinnvoll ist), nach der Präannotation und nach dem Anwenden beliebig vieler Regeln durchgeführt werden.

5.2. Entwurf

Die interne Architektur von RESI besteht aus vier Teilen: Regelobjekten mit der Funktionalität zum Finden von Problemen, Graphobjekten zum Zugriff auf die Spezifikation, Ontologieobjekten zum Zugriff auf die externen Ontologien und einer zentralen

Anwendung, die alle Komponenten kontrolliert. Die Teile werden im Detail in diesem Abschnitt vorgestellt, das dazugehörige UML-Diagramm befindet sich in Abbildung 5.2. Zusätzlich gibt es eine Benutzeroberfläche für die Interaktion mit dem Benutzer.

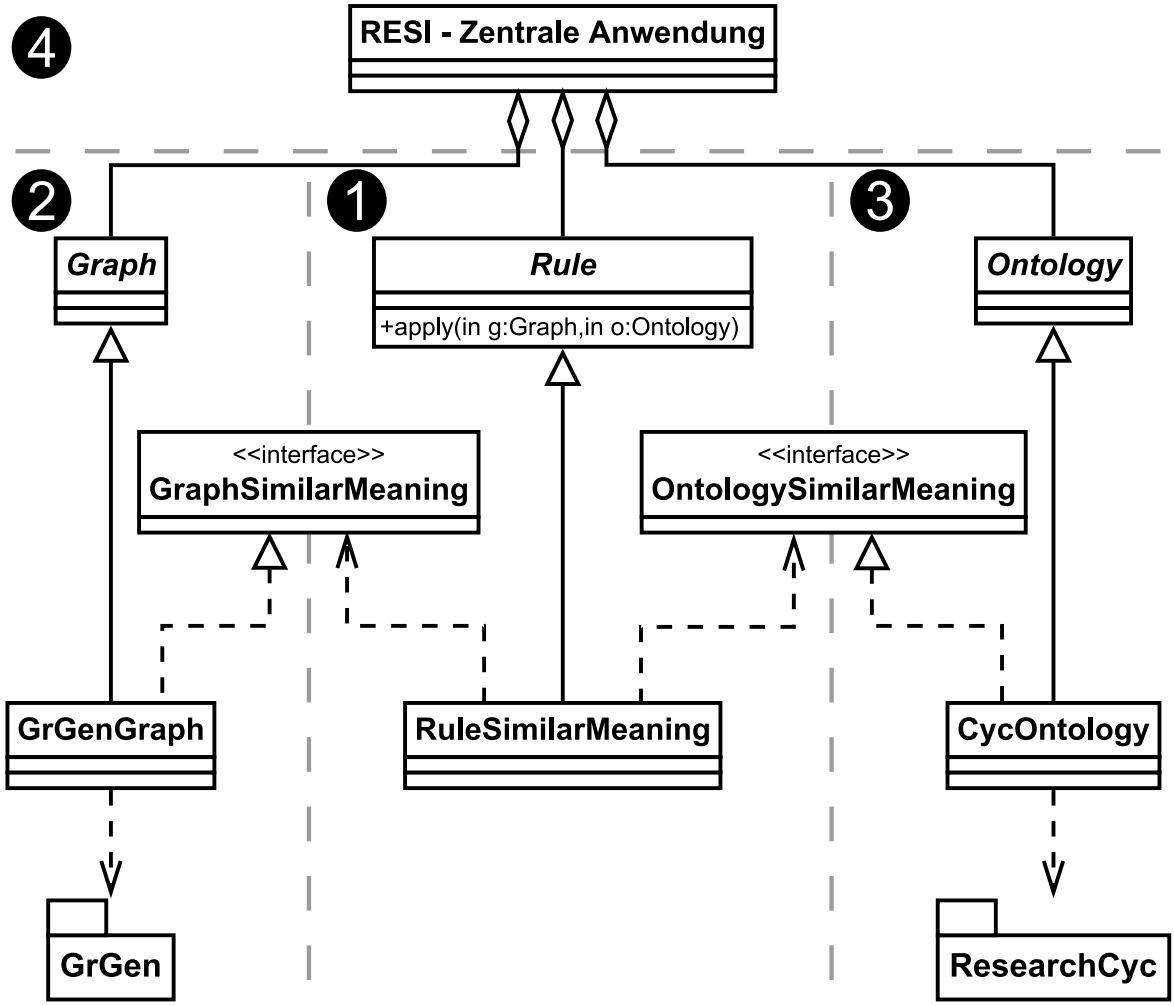

Abbildung 5.2.: UML-Entwurf von RESI

5.2.1. Regeln

Jede Regel dient dazu, eine bestimmte Art von Problemen in Spezifikationen zu finden. Die Funktionalität ist in einer Regelklasse implementiert. Sie liest die nötigen Informationen der Spezifikation aus einem Graphobjekt ein und verarbeitet sie. Wenn weitere Informationen nötig sind, um Probleme zu identifizieren, stellt sie Anfragen an ein Onto-

logieobjekt, das ihr Daten aus einer Ontologie liefert. Gefundene mögliche Probleme und Verbesserungsvorschläge werden dem Benutzer zur Interaktion angezeigt. Sämtliche ermittelten Informationen und Kommentare, die der Nutzer eingibt, werden anschließend zurück in das Graphobjekt geschrieben. Ein detaillierter Ablauf der einzelnen Regeln befindet sich in Abschnitt 5.3.

Jede Regelklasse definiert Schnittstellen für Graph- und Ontologieklassen. Diese Schnittstellen müssen von einer Graph- beziehungsweise einer Ontologieklasse implementiert werden, um die Regel zu unterstützen. In Abbildung 5.2 ist die Regel Überprüfe auf Wörter gleicher Bedeutung mit ihren Schnittstellen abgebildet (*RuleSimilarMeaning* sowie die Schnittstellen *RuleSimilarMeaningGraphInterface* und *RuleSimilarMeaningOntologyInterface*). Der Quelltext der beiden Schnittstellen befindet sich im Anhang B in Quelltext B.1 und Quelltext B.2.

5.2.2. Graphen

Spezifikationen werden in RESI intern als Graphen repräsentiert. Es wurde eine Repräsentation als Graph gewählt, weil diese einfacheren Zugriff auf die Details der Spezifikation ermöglicht als zum Beispiel eine Textrepräsentation. Ein Graphobjekt in RESI enthält einen Graphen, der beim Import einer Spezifikation erstellt wird.

Das Graphobjekt erlaubt den Regeln, Anfragen an die Spezifikation zu stellen und schreibend auf die Spezifikation zuzugreifen. Es unterstützt eine Regel, wenn es die Graphschnittstelle dieser Regel implementiert. Sämtliche hinzugefügten Informationen – sei es durch Anwenden der Regeln oder durch die vorher stattfindende Präannotation – werden im Graphobjekt zwischengespeichert und erst beim Export in die Spezifikation zurückgeschrieben. Die Graphklasse *GraphGrGen* mit ihrer Anbindung an das externe GrGen-System [28], in dem die Spezifikation liegt, ist in Abbildung 5.2 dargestellt.

5.2.3. Ontologien

Ein Ontologieobjekt kommuniziert mit einer externen Ontologie, indem es Anfragen der Regeln in Anfragen der Ontologie umwandelt und anschließend die Antwort der externen Ontologie in ein für die Regel verarbeitbares Format konvertiert. Um eine Regel zu unterstützen, muss es die Ontologieschnittstelle der Regel implementieren.

RESI unterstützt vier Ontologien: ResearchCyc [16], WordNet [52], ConceptNet [34] und

YAGO [64]. Auf Grund ihres unterschiedlichen Entwurfs unterstützen die Ontologien unterschiedlich viele Regeln. Welche Ontologien eine bestimmte Regel unterstützen, steht in den Detailbeschreibungen der Regeln in Abschnitt 5.3. Im UML-Diagramm in Abbildung 5.2 ist die Ontologieklasse *CycOntology*, die ResearchCyc an RESI anbindet, abgebildet.

5.2.4. Zentrale Anwendung

Die zentrale Anwendung kontrolliert, welche der oben aufgeführten Komponenten miteinander interagieren (siehe Screenshot in Abbildung 5.3). Sie verwaltet, *welche* Regeln auf *welche* Graphen mit Hilfe von *welchen* Ontologien angewandt werden.

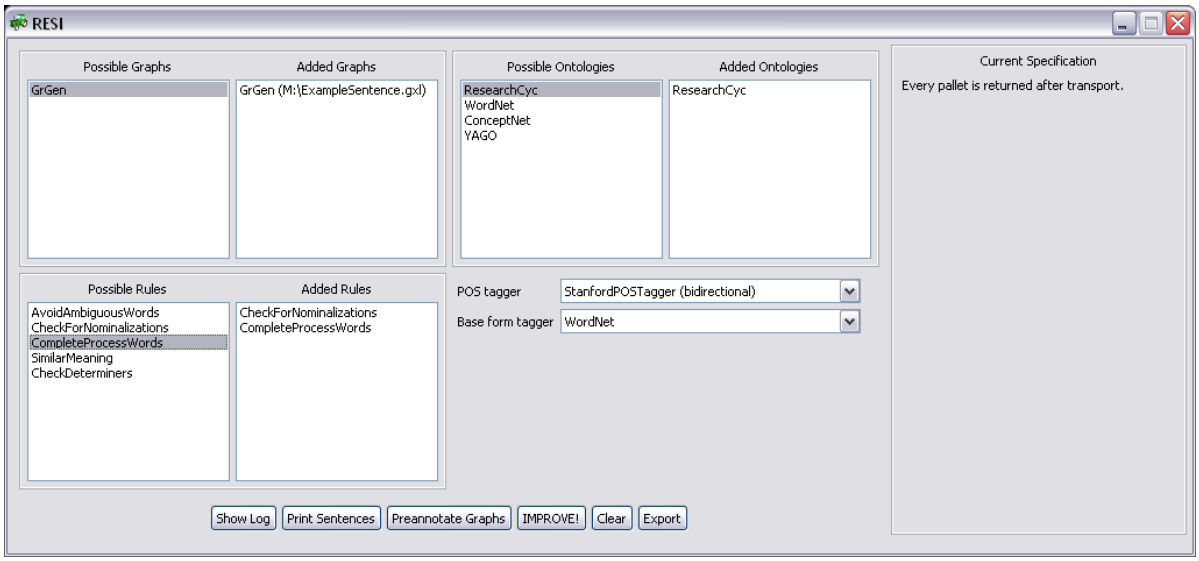

Abbildung 5.3.: Konfigurationsbildschirm von RESI

Außerdem beinhaltet die zentrale Anwendung die Präannotation (siehe Abschnitt 5.1.2). Die Tagger zur Annotation der Wortarten und Grundformen werden durch Schnittstellen definiert.

Der Benutzer importiert die zu verbessernde Spezifikation, indem er den entsprechenden Graphen auswählt. Nach der anschließenden Auswahl der Tagger stößt er die Präannotation an. Nachdem er die anzuwendenden Regeln und die dabei zu verwendenden Ontologien bestimmt hat, wendet RESI die Regeln auf den Graphen (also die importierte Spezifikation) an. Auch der Export der Spezifikation wird von der zentralen Anwendung für den Benutzer bereit gestellt.

5.3. Regelbeschreibung

In diesem Abschnitt werden die implementierten Regeln im Detail beschrieben. Für die meisten Regeln dient der folgende Beispielsatz zur Veranschaulichung:

```
Every pallet is returned after transport.
```

Des Weiteren wird für alle Beispiele die Ontologie ResearchCyc befragt. Im Pseudocode, der die Algorithmen der Regeln beschreibt, werden Interaktionen mit dem Graphen in rot, mit der Ontologie in blau und mit dem Benutzer in grün dargestellt, wie auch in Abbildung 5.4 ersichtlich.

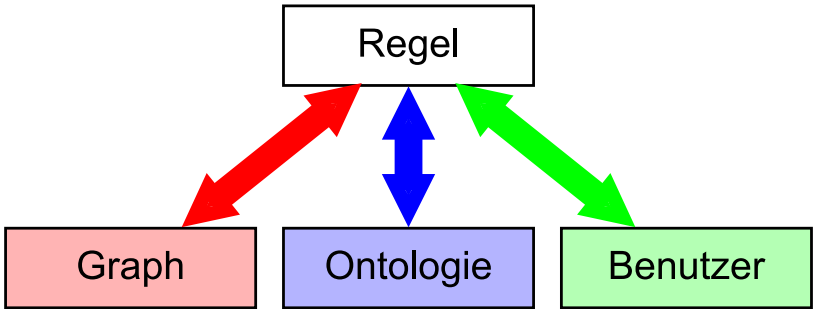

Abbildung 5.4.: Legende der Farbverwendung im Pseudocode

5.3.1. Überprüfe auf Nominalisierungen

Die Regel Überprüfe auf Nominalisierungen spürt Nominalisierungen auf und liefert Vollverben, die statt der Nominalisierung verwendet werden sollten (siehe Abschnitt 4.1.1). Dabei geht die Regel nach folgendem Algorithmus vor:

```
Lese alle Substantive aus dem Graphen
Für jedes Wort aus der Substantivliste {
        Ermittle aus der Ontologie eine Liste von Verben, die
           der möglichen Nominalisierung entsprechen
        Wenn kein Verb zurückkommt {
                Tue nichts, weil es keine Nominalisierung ist
                   oder kein passendes Verb gefunden wurde
        }
        Sonst {
```

```
                Lasse den Benutzer einen Kommentar zu der
                    Nominalisierung schreiben und füge
                    automatisch die gefundenen Verben in den
                    Kommentar ein
                Wenn der Benutzer einen Kommentar hinterlässt {
                        Schreibe den Kommentar in den Graphen
                            zurück
                }
        }
}
```

Die dazu gehörige Benutzerinteraktion ist auf dem Screenshot in Abbildung 5.5 zu sehen. Die dort vorhandene Nominalisierung `transport` sollte durch das Verb *transport* ersetzt werden, das dann detaillierter spezifiziert werden muss.

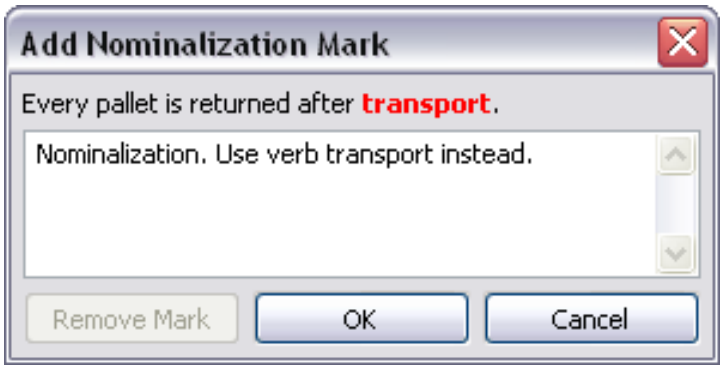

Abbildung 5.5.: Screenshot zur Regel Überprüfe auf Nominalisierungen

Die Kommentare werden im Graphen als Nominalisierungskommentare markiert, um sie von anderen Kommentaren unterscheiden zu können. Der Benutzer kann nach der Verwendung von RESI diese Kommentare in der Spezifikation wiederfinden, um die Sätze mit Hilfe der vorgeschlagenen Verben neu zu formulieren.

ResearchCyc ist die einzige Ontologie, die diese Regel unterstützt. Die Anfragen, die zur Ermittlung der Verben an ResearchCyc gestellt werden, sind in Anhang A.1.1 zu finden.

5.3.2. Vervollständige Prozesswörter

Die Regel Vervollständige Prozesswörter überprüft für jedes Prozesswort, welche Argumente notwendig sind und zeigt diese dem Benutzer an, damit er fehlende Elemente des Satzes ermitteln kann (siehe Abschnitt 4.1.2). Dabei wird der folgende Algorithmus angewandt:

```
Lese alle Prozesswörter aus dem Graphen
Für jedes Wort aus der Prozesswortliste {
        Ermittle die möglichen Argumentlisten aus der Ontologie
           (*)
        Zeige dem Benutzer die Argumentlisten
        Lasse den Benutzer wählen, welche Argumentliste die
           zutreffende ist
        Zeige für die ausgewählte Liste die Argumente an
        Lasse den Benutzer allen Argumenten Satzelemente
           zuweisen
        Lasse den Benutzer optional einen Kommentar angeben
        Schreibe die Verknüpfungen zwischen Prozesswörtern und
           ihren Argumenten im Satz sowie den Kommentar zurück
           in den Graphen
}

(*) In den Argumentlisten sind für jedes Argument semantische
   sowie syntaktische Rolle im Satz sowie das das Argument
   repräsentierende Satzelement angegeben
```

In unserem Beispiel ist `return` das Prozesswort. ResearchCyc liefert für dieses Wort, das in der Ontologie die Bedeutung *ReturningSomething* hat, zwei mögliche Argumentlisten mit syntaktischen Rollen (in Großbuchstaben) und semantischen Rollen (kursiv, werden in Anhang C erläutert):

ReturningSomething

- SUBJECT: *performedBy*
- OBJECT: *objectOfPossessionTransfer*

ReturningSomething 2

- OBJECT: *objectGiven*
- SUBJECT: *giver*
- OBLIQUE-OBJECT: *givee*

Der Benutzer entscheidet sich für die Argumentliste *ReturningSomething 2* (siehe Abbildung 5.6). Wie im Bild ersichtlich, wurde das zurückzugebende Objekt, das *objectGiven* von RESI korrekt als `pallet` identifiziert. Die anderen beiden nötigen Argumente (der *giver* und der *givee*, für den man die genaue Erklärung im Tooltip sehen kann) fehlen und sollten in der Spezifikation ergänzt werden. Im Graphen wird vermerkt, dass `pallet` das *objectGiven* ist und dass der *giver* und der *givee* fehlen. Optional kann der Benutzer zu dieser Markierung im Graphen noch einen Kommentar hinzufügen.

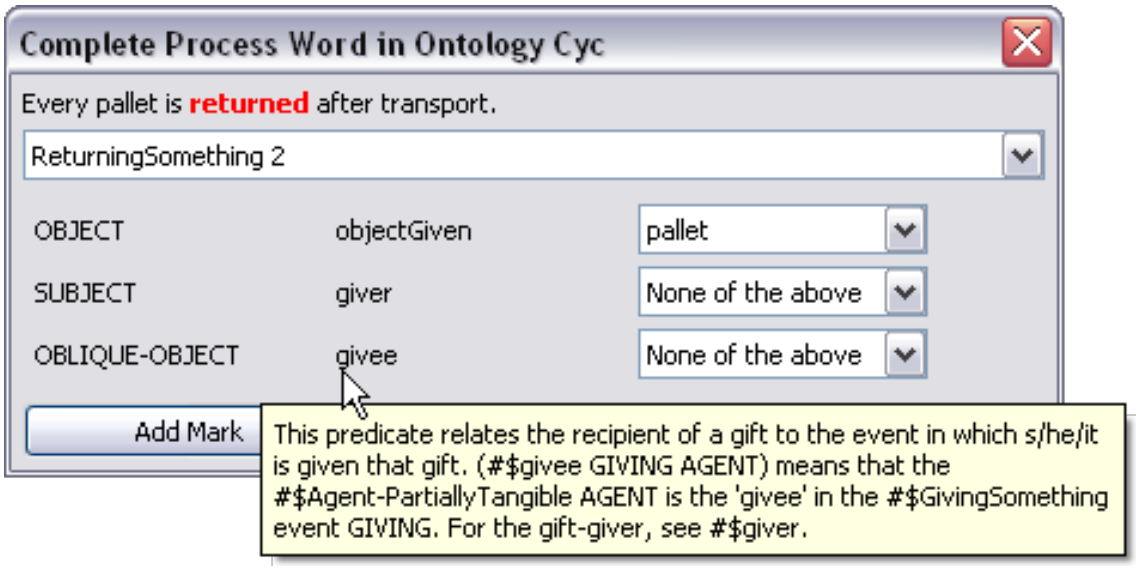

Abbildung 5.6.: Screenshot zur Regel Vervollständige Prozesswörter

Mit Hilfe dieser Informationen ist es später leichter, die Spezifikation zu ergänzen.

Auch diese Regel wird ausschließlich durch ResearchCyc unterstützt. Die zur Ermittlung der Argumentslisten notwendigen Anfragen befinden sich in Anhang A.1.2.

5.3.3. Überprüfe Artikel & Quantoren

Mit Hilfe der Regel Überprüfe Artikel & Quantoren wird überprüft, ob Artikel und Quantoren korrekt und entsprechend ihrer Bedeutung verwendet werden (siehe Abschnitt 4.1.3). Dazu wird dieser Algorithmus verwendet:

```
Lese alle Artikel und Quantoren aus dem Graphen
Für jedes Wort aus der Artikel-/Quantorenliste {
        Ermittle die Bedeutung des Wortes aus der Ontologie
        Zeige dem Benutzer die Bedeutung an
```

```
        Lasse den Benutzer bestätigen, dass er diese Bedeutung
            gemeint hat, oder ihn seine gemeinte Bedeutung
            angeben
        Lasse den Benutzer optional einen Kommentar angeben
        Schreibe die Bedeutung und den Kommentar zurück in den
            Graphen
}
```

Wie in Abbildung 5.7 ersichtlich ist, erkennt RESI, dass mit `Every pallet` jede Palette (ohne Ausnahme) gemeint ist. Wenn der Benutzer dies einschränken möchte (zum Beispiel, weil er klar machen möchte, dass nur die Paletten gemeint sind, die auch am Transport beteiligt waren), kann er das in einem Kommentar vermerken. Auf dem Screenshot sind auch die anderen Möglichkeiten für Quantoren zu erkennen. Bestimmte und unbestimmte Artikel werden von RESI mit Hilfe dieser Regel ebenfalls erkannt. Bei bestimmten Artikeln muss der Benutzer überprüfen, ob eindeutig klar ist, worauf referenziert wird. Bei unbestimmten Artikel ist zu überprüfen, ob nicht eigentlich ein Quantor gemeint ist (meistens bedeutet dies, dass *a* fälschlicherweise statt *one* verwendet wird). Die genaue Bedeutung des Quantors oder die Markierung als (un)bestimmter Artikel wird im Graphen zusammen mit dem Kommentar vermerkt.

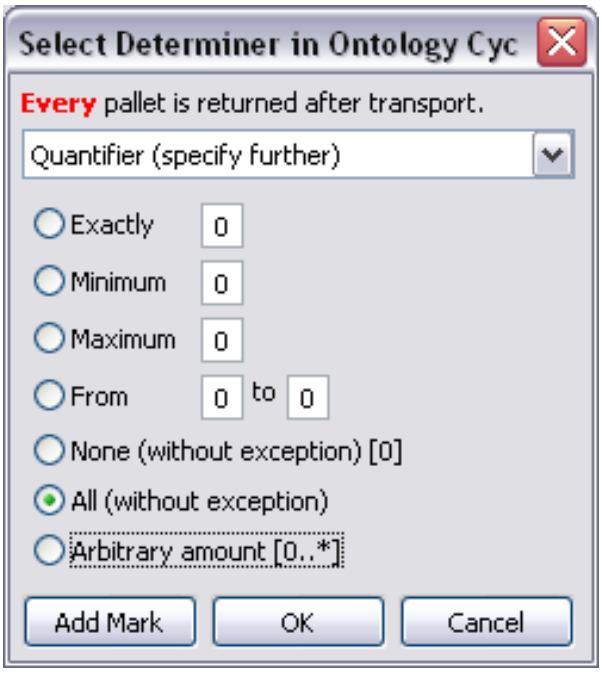

Abbildung 5.7.: Screenshot zur Regel Überprüfe Artikel & Quantoren

Durch die im Graphen gespeicherten Informationen können missverständliche Quantoren/Artikel in der Spezifikation ausgetauscht werden. Außerdem können insbesondere die genauen Details über Quantoren im weiteren Softwareentwicklungsprozess verwendet werden, zum Beispiel, um Kardinalitäten für UML-Diagramme zu ermitteln.

Nur ResearchCyc unterstützt diese Regel. In Anhang A.1.3 stehen die Anfragen, die zur Ermittlung der Wortbedeutung an ResearchCyc gestellt werden.

5.3.4. Überprüfe auf mehrdeutige Wörter

Die Regel Überprüfe auf mehrdeutige Wörter ermittelt für die Wörter der Spezifikation, welche Bedeutungen sie haben können, um Mehrdeutigkeiten zu verringern (siehe Abschnitt 4.2.1). Sie nutzt den folgenden Algorithmus:

```
Lese alle Wörter aus dem Graphen
Für jedes Wort aus der Wortliste {
        Ermittle die möglichen Bedeutungen des Wortes aus der
           Ontologie
        Zeige dem Benutzer die Bedeutungen an
        Lasse den Benutzer die korrekte Bedeutung auswählen
        Lasse den Benutzer optional einen Kommentar angeben
        Schreibe die Bedeutung und den Kommentar zurück in den
           Graphen
}
```

Wie bereits in Abschnitt 5.1.3 beschrieben, ergibt die Überprüfung des Wortes `transport`, dass es vier mögliche Bedeutungen (*Conveyance*, *TransmittingSomething*, *TransportationDevice* und *TransportationEvent*) gibt, die in Anhang C erklärt werden. Die Auswahl der korrekten Bedeutung *TransportationEvent* und die dazugehörige Erklärung ist in Abbildung 5.8 zu sehen. Die Bedeutung des Wortes wird im Graphen zusammen mit dem Kommentar gespeichert.

Die gespeicherten Bedeutungen werden auch von anderen Regeln verwendet, um mehrmaliges Nachfragen nach einer Wortbedeutung zu vermeiden. Des Weiteren können Leser der Spezifikation über die vermerkte Bedeutung (und den eventuell vorhandenen Kommentar) eindeutig herausfinden, was das Wort bedeuten soll. Sollte die Bedeutung nicht eindeutig vom Benutzer zu bestimmen sein, kann er durch den Kommentar später leicht das mehrdeutige Wort wiederfinden, um es zu ersetzen beziehungsweise genauer zu erklären. Wörter, denen bereits Bedeutungen zugewiesen wurden, werden bei einer erneuten Regelanwendung übersprungen.

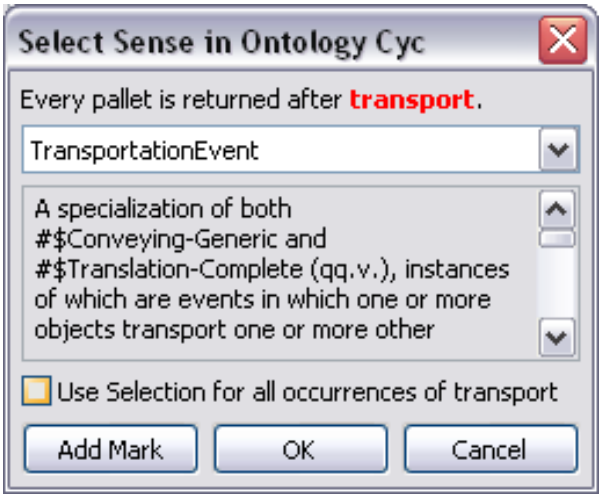

Abbildung 5.8.: Screenshot zur Regel Überprüfe auf mehrdeutige Wörter

Diese Regel kann ResearchCyc und WordNet verwenden, die passenden Anfragen befinden sich in Anhang A.1.4 und in Anhang A.2.1.

5.3.5. Überprüfe auf Wörter gleicher Bedeutung

Die Regel Überprüfe auf Wörter gleicher Bedeutung dient dazu, Objekte aufzuspüren, für die in der Spezifikation verschiedene Begriffe verwendet werden (siehe Abschnitt 4.2.2). Das Vorgehen entspricht dem folgenden Algorithmus:

```
Lese alle Substantive aus dem Graphen
Für jedes Wort wort1 aus der Substantivliste {
        Für jedes Wort wort2 aus der Substantivliste {
                Wenn wort2 lexikographisch nach wort1 kommt {
                        Ermittle die Ähnlichkeit von wort1 und
                           wort2
                        Wenn die Wörter ähnlich sind und wort1
                           ein Oberbegriff von wort2 ist {
                                Lasse Benutzer entscheiden, ob
                                   wort2 durch wort1 ersetzt
                                   werden soll
                                Wenn der Benutzer zustimmt {
                                        Schreibe wort1 an die
                                           Stelle von wort2 in
                                           den Graphen
```

```
                                    }
                                }
                                Sonst {
                                    Wenn die Wörter ähnlich sind
                                        und wort2 ein Oberbegriff
                                        von wort1 ist {
                                            Lasse Benutzer
                                                entscheiden, ob
                                                wort1 durch wort2
                                                ersetzt werden soll
                                            Wenn der Benutzer
                                                zustimmt {
                                                    Schreibe wort2
                                                        an die
                                                        Stelle von
                                                        wort1 in den
                                                        Graphen
                                            }
                                    }
                                }
                            }
                        }
}
```

In unserem Beispielsatz gibt es keine zwei Worte mit ähnlicher Bedeutung. Als weiteres Beispiel dient ein Text über einen Videoverleih, der im Rahmen der Evaluierung (Kapitel 6) untersucht wurde. Er enthält die Begriffe `tape` und `video`, die beide synonym als Begriff für eine Videokassette verwendet werden. Eine Überprüfung mit Hilfe von RESI ergibt, dass `tape` ein Oberbegriff für `video` ist. Der Bestätigungsdialog für die Ersetzung ist in Abbildung 5.9 zu sehen. Angezeigt wird auch der prozentuale Wert, wie sicher sich die Ontologie ist, dass `video` mit `tape` ersetzt werden sollte. Je nach Auswahl des Benutzers können alle oder keine Vorkommen von `video` mit `tape` ersetzt werden. Es ist auch möglich, dass nur dieses angezeigte Vorkommen von `video` ersetzt wird.

Diese Regel ist die einzige, die von allen an RESI angebundenen Ontologien (ResearchCyc, WordNet, ConceptNet und YAGO) unterstützt wird. Die Anfragen an die Ontologien zur Ermittlung der Ähnlichkeit finden sich in Anhang A.1.5, Anhang A.2.2, Anhang A.3.1 und Anhang A.4.1. Auf Grund der Struktur von ResearchCyc und YAGO kann für diese Ontologien der prozentuale Konfidenzwert nicht abgestuft werden, alle Ersetzungen werden mit einer Sicherheit von 100% vorgeschlagen. WordNet stuft die-

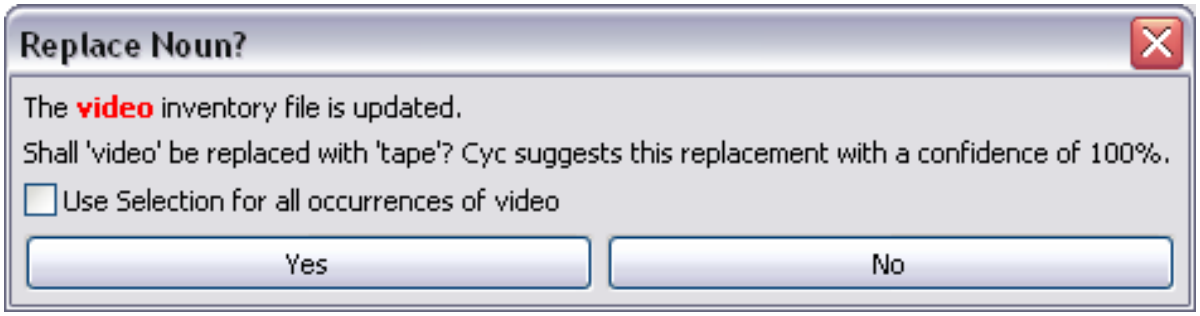

Abbildung 5.9.: Screenshot zur Regel Überprüfe auf Wörter gleicher Bedeutung

sen Wert über die Anzahl der möglichen Oberbegriffe des Unterbegriffs ab. Wenn zum Beispiel `video` vier Oberbegriffe hat, von denen einer `tape` ist, so liefert WordNet eine Sicherheit von 25%. ConceptNet funktioniert ähnlich bei der Ermittlung des Wertes. Hier fließt zusätzlich noch die Qualitätsmaßzahl der Verbindung Oberbegriff zu Unterbegriff ein. Diese Qualitätsmaßzahlen werden im Rahmen der Evaluierung der Ontologie vergeben (siehe auch Abschnitt 3.5). Wenn beispielsweise die Verbindung `video` zu `tape` die Qualitätsmaßzahl 3 hat und die anderen drei Verbindungen nur eine Qualitätsmaßzahl von jeweils 1, so wird ConceptNet eine Sicherheit von 50% liefern.

5.4. Implementierungsdetails und Erweiterbarkeit

RESI ist größtenteils in Java [65] geschrieben. Für die Benutzeroberfläche wurde das Standard Widget Toolkit (SWT) [66] verwendet.

Neue Regeln – und damit neue Funktionalität, um Probleme in Spezifikationen zu beheben – müssen außer ihrem Algorithmus nur Schnittstellen definieren, über die sie mit Graphen und Ontologien kommunizieren wollen. Sobald diese Schnittstellen von Graph- und Ontologieklassen implementiert werden, kann die Regel auf Spezifikationen angewandt werden.

Bis jetzt unterstützt RESI nur das Importieren einer Spezifikation, die mit Hilfe von SAL$_E$MX [30] in einen GrGen-Graphen [28] umgewandelt wurde. Dies bedeutet auch, dass die Anbindung an den Graphen bis jetzt in C# [22] geschrieben ist, weil GrGen nur eine C#-Programmierschnittstelle anbietet. Die Verbindung zwischen Java und C# wird über IKVM.NET [26] realisiert. IKVM.NET ist eine virtuelle Maschine für Java, die in .NET [51] implementiert ist. Da sie keine 64-Bit-Version von SWT anbietet, ist RESI nur auf 32-Bit-Systemen lauffähig.

Die Informationen, die im Rahmen von SAL$_E$MX durch thematische Rollen zu der Spezifikation hinzugefügt wurden (siehe Abschnitt 3.1), sind für RESI nicht zwingend erforderlich. Dementsprechend wäre es möglich, andere Repräsentationen von Spezifikationen zu im- und exportieren, wenn eine Schnittstelle in Form einer Graphenklasse dafür implementiert wird. Sobald diese Klasse die Schnittstelle einer Regel korrekt implementiert, unterstützt sie diese Regel.

Weitere Ontologien – wie zum Beispiel domänenspezifische Ontologien – können analog durch Erstellen einer Ontologieklasse, die die Ontologieschnittstellen der zu unterstützenden Regeln implementiert, angebunden werden. Die bisher angebundenen externen Ontologien kommunizieren mit den entsprechenden Ontologieklassen über TCP.

ResearchCyc enthält bereits einen Server, der über eine Java-Schnittstelle, die im Rahmen von OpenCyc [15] veröffentlicht wurde, angesprochen werden kann. YAGO nutzt einen rein datenbankbasierten Zugriff, der aber durch eine mitgelieferte Java-Schnittstelle stark vereinfacht wird. Für ConceptNet wurde ein eigener Server in Python [56] geschrieben, der Anfragen über XML-RPC [68] annimmt und an die mit ConceptNet mitgelieferte Python-Schnittstelle weiterreicht.

Auch für WordNet wurde ein eigener Server entwickelt. Er basiert auf JWordNet [63], das um Funktionalität erweitert wurde. So wurde unter anderem die Funktion zum Finden von Grundformen mit Hilfe von WordNet gemäß der WordNet-Spezifikation verbessert. Durch diese Funktionalität ist es möglich, WordNet als Tagger für die Grundformen der Substantive und Verben einzusetzen (siehe Abschnitt 5.1.2). Bis jetzt unterstützt nur WordNet diese Funktionalität. Es ist aber möglich, über die durch die zentrale Anwendung definierte Schnittstelle einen anderen Tagger zu implementieren. Das Gleiche gilt für den Wortarten-Tagger, der bis jetzt nur über den *Stanford Log-linear Part-Of-Speech Tagger* [67] realisiert wird.

5.5. Externer Server

Mit Hilfe der Serversoftware der Ontologien können diese auf einem anderen Rechner laufen. Da externe Ontologien sehr ressourcenintensiv sind (insbesondere ResearchCyc, was mehrere Gigabyte Hauptspeicher benötigt), konnte so ein System mit den Ontologien auf einem an der Universität stationierten Rechner installiert werden. Dieses System dient ausschließlich als Ontologieserver für RESI und soll in Zukunft das gesamte AUTOMODEL-Projekt [45] unterstützen.

Das System läuft als virtuelle Maschine mit Debian GNU/Linux [62] innerhalb eines VM-

ware ESX Servers [69], wobei der virtuellen Maschine 4 Gigabyte Arbeitsspeicher zugesichert werden und als Prozessor ein AMD-Opteron-Prozessor dient.

Von jedem Ort mit Internetanschluss kann durch einen SSH-Tunnel [73] auf die auf dem System laufenden Ontologien zugegriffen werden als ob sie auf einem Rechner im lokalen Netzwerk laufen würden.

6. Evaluierung

Zur Überprüfung der Funktionalität von RESI wurden 2 Spezifikationstexte mit Hilfe von RESI bearbeitet. Zur Anwendung der Regeln wurde ausschließlich die Ontologie ResearchCyc verwendet. Die Ergebnisse aus dieser Evaluierung sowie der Vergleich mit anderen Arbeiten, die die gleichen Texte untersucht haben, findet sich in diesem Kapitel. Im Anschluss daran wird die Laufzeit von RESI sowie die Verwendung von anderen Ontologien beleuchtet und abgegrenzt, was RESI nicht kann.

6.1. Erster Spezifikationstext: ABC Video Rental

Der Spezifikationstext wurde von Kiyavitskaya et al. verwendet, um die Funktionalität ihres Werkzeugs zu zeigen [44]. Zusätzlich listen sie ausführlich weitere Probleme auf, die ihr Werkzeug nicht finden kann, die sie aber trotzdem in den Sätzen sehen. Der Text beschreibt Prozesse, die zu der fiktiven Videothek ABC gehören und ist in Quelltext 6.1 zu finden. Für die Verwendung mit RESI mussten Bindestriche durch Unterstriche ersetzt und Zahlen ausgeschrieben werden.

Quelltext 6.1: Erster Spezifikationstext: ABC Video Rental [44]

```
Customers select at least one video for rental.
The maximal number of tapes that a customer can have
   outstanding on rental is 20.
The customer's account number is entered to retrieve
   customer data and create an order.
Each customer gets an id card from ABC for identification
   purposes.
This id card has a bar code that can be read with the bar
   code reader.
Bar code Ids for each tape are entered and video information
   from inventory is displayed.
The video inventory file is updated.
When all tape Ids are entered, the system computes the total
```

```
   bill.
Money is collected and the amount is entered into the
   system.
Change is computed and displayed.
The rental transaction is created, printed and stored.
The customer signs the rental form, takes the tape(s) and
   leaves.
To return a tape, the video bar code ID is entered into the
   system.
The rental transaction is displayed and the tape is marked
   with the date of return.
If past-due amounts are owed they can be paid at this time;
   or the clerk can select an option which updates the
   rental with the return date and calculates past-due fees.
Any outstanding video rentals are displayed with the amount
   due on each tape and the total amount due.
Any past-due amount must be paid before new tapes can be
   rented.
```

Welche Probleme von RESI gefunden werden und welche nicht gefunden werden, obwohl sie von Kiyavitskaya et al. manuell als Probleme identifiziert wurden, wird nachfolgend nach Regeln geordnet aufgelistet. Die Farblegende der dabei zur Veranschaulichung verwendeten Grafiken befindet sich in Abbildung 6.1.

Abbildung 6.1.: Farblegende der Evaluierungsgrafiken

6.1.1. Überprüfe auf Nominalisierungen

RESI findet 4 Nominalisierungen: `order`, `identification` und 2 Mal `return`. Der Prozess der Rückgabe wird ausführlich beschrieben, die Verwendung von `return` ist in diesem Fall nicht problematisch. `order` wird nicht genauer definiert, an dieser Stelle ist Nachbesserungsbedarf. Dass ein Problem bei `identification` existiert, erwähnen auch Kiyavitskaya et al. und schlagen ebenfalls vor, an der Stelle das Vollverb zu verwenden, wie es auch RESI tut. Das Problem des nicht ausreichend spezifizierten `order` erwähnen sie nicht. Dargestellt wird die jeweils gefundene Anzahl an Nominalisierungen in Abbildung 6.2.

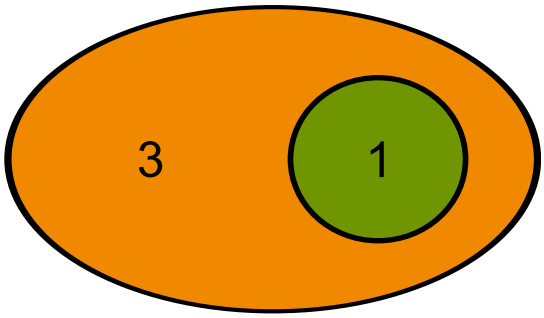

Abbildung 6.2.: Anzahl der gefundenen Nominalisierungen

6.1.2. Vervollständige Prozesswörter

Von RESI werden 14 Prozesswörter gefundenen, denen Argumente fehlen. Einigen fehlen sogar mehrere Argumente, sodass insgesamt 18 fehlende Argumente von RESI bemängelt werden. Kiyavitskaya et al. konzentrieren sich in ihrer manuellen Überprüfung auf den Spezialfall der Passivkonstruktion, der insgesamt 20 Mal im Text vorhanden ist. Von diesen findet RESI 11, zu den anderen 9 (wobei 5 davon allein auf das Verb `enter` fallen) befinden sich in der Ontologie ResearchCyc keine Argumentslisten. Für 3 der 11 gefundenen unvollständig spezifizierten Prozesswörter wird dabei ein zweites fehlendes Argument gefunden, das von Kiyavitskaya et al. nicht erkannt wurde. Zusätzlich findet RESI 3 weitere Prozesswörter, denen Argumente fehlen und die somit nicht vollständig spezifiziert sind. Diese sind nicht im Passiv formuliert, sodass Kiyavitskaya et al. sie nicht als Problem erkannt haben. Die Anzahl der von RESI beziehungsweise Kiyavitskaya et al. gefundenen unvollständig spezifizierten Prozesswörter wird in Abbildung 6.3 dargestellt.

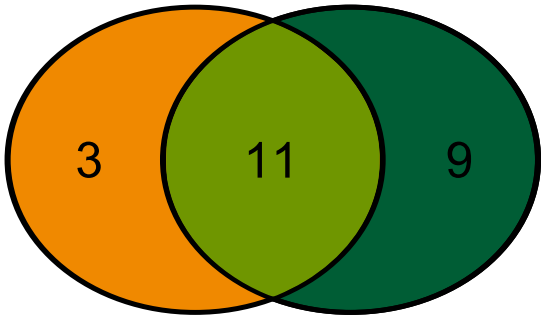

Abbildung 6.3.: Anzahl der gefundenen unvollständig spezifizierten Prozesswörter

6.1.3. Überprüfe Artikel & Quantoren

Kiyavitskaya et al. bemängeln 6 bestimmte Artikel, deren zugehöriges Substantiv keinen Bezugsindex hat. RESI weist explizit auf alle 24 bestimmten Artikel hin, sodass auch die 6 problematischen darunter sind (siehe auch Abbildung 6.4). Zusätzlich werden alle 6 unbestimmten Artikel erkannt, die in jedem Fall *genau 1* bedeuten und dementsprechend durch *one* ersetzt werden sollten. Von Kiyavitskaya et al. wird nur einer der unbestimmten Artikel als Problem angesehen (siehe auch Abbildung 6.5). Die 8 enthaltenen Quantoren werden auch von der Regel erkannt, wobei allerdings die Bedeutung von `at least one` nicht korrekt wiedergegeben wird, weil nur einzelne Wörter betrachtet werden und dementsprechend `one` als *genau 1* erkannt wird. Die Bedeutung von `each` ist nicht korrekt in der Ontologie ResearchCyc mit dem Wort verknüpft, sodass auch hierfür nicht die korrekte Bedeutung automatisch angezeigt werden kann.

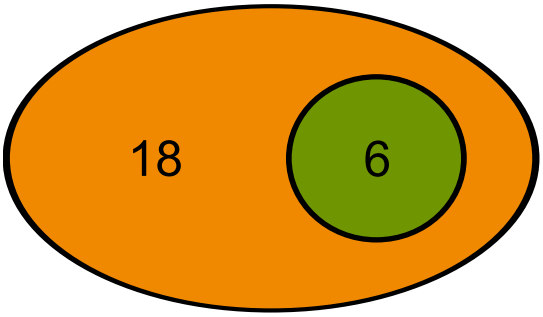

Abbildung 6.4.: Anzahl der gefundenen bestimmten Artikel

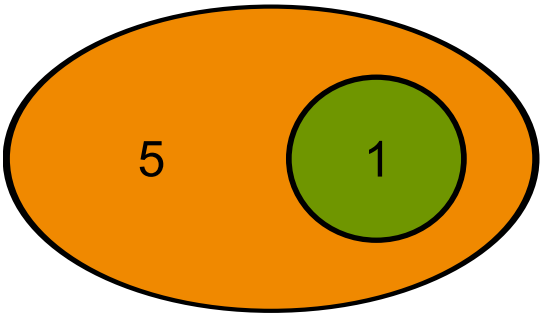

Abbildung 6.5.: Anzahl der gefundenen unbestimmten Artikel

6.1.4. Überprüfe auf mehrdeutige Wörter

Von den 222 Wörtern im Text befindet RESI 44 für mehrdeutig, wobei sie insgesamt 198 Bedeutungen außer den gemeinten aufzeigt. Für 12 Wörter wird durch die Beschreibung

aus der Ontologie die Bedeutung genauer spezifiziert. Diese Regel funktioniert ähnlich wie das Werkzeug von Kiyavitskaya et al. und unterliegt dementsprechend auch dem gleichen Problem. Nahezu alle Bedeutungen sind für den menschlichen Leser eindeutig, obwohl das Werkzeug viele Probleme entdeckt. Dies liegt daran, dass der menschliche Leser (insbesondere, wenn er Domänenwissen hat) den Kontext wahrnimmt, in dem das mehrdeutige Wort steht, und somit automatisch die in diesem Kontext korrekte Bedeutung aus der Liste der möglichen Bedeutungen auswählt. Es kann aber sinnvoll sein, Fragen zu stellen, auf die ein Domänenexperte nie kommen würde, um Probleme in Spezifikationen aufzudecken [4]. Deshalb lässt RESI den Benutzer die genaue Bedeutung für jeden Begriff auswählen und speichert diese Bedeutung auch, obwohl es für einen Domänenexperten eventuell überflüssig erscheint. Somit können andere Menschen, die mit dieser Spezifikation arbeiten müssen, überprüfen, ob sie wirklich die richtige Bedeutung interpretiert haben.

6.1.5. Überprüfe auf Wörter gleicher Bedeutung

Für den vorliegenden Text schlägt RESI 11 Begriffe zur Ersetzung vor. Diese sollten durch einen anderen Begriff aus dem Text ersetzt werden, um mehrere Begriffe für das gleiche Objekt zu vermeiden. Davon ist nur die Ersetzung von `video` mit `tape` sinnvoll, die anderen Vorschläge sind entweder falsch (wie zum Beispiel die Ersetzung von `clerk` mit `system`) oder nicht notwendig (wie zum Beispiel `number` mit `amount` zu ersetzen). Außer dem Paar `video`/`tape` finden Kiyavitskaya et al. noch 4 weitere Synonyme, für die jeweils der gleiche Begriff verwendet werden sollte (siehe auch Abbildung 6.6). Neben den vielen falschen Vorschlägen hat diese Regel noch ein weiteres Problem: die lange Ausführungszeit. Da jedes Substantiv aus der Spezifikation mit jedem anderen verglichen wird, steigt die Laufzeit der Regel quadratisch mit der Anzahl der Substantive im Text, was schon bei diesem kurzen Text zu für den Benutzer unangenehmen Laufzeiten führt.

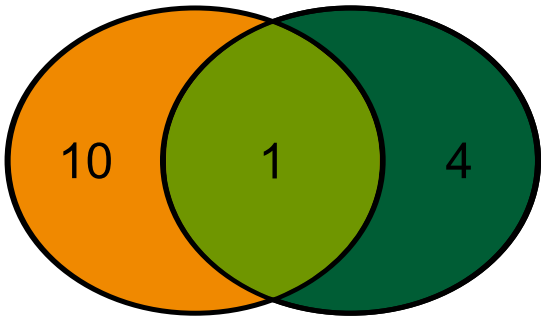

Abbildung 6.6.: Anzahl der gefundenen Synonympaare

6.1.6. Weitere Probleme und Zusammenfassung

Kiyavitskaya et al. listen noch 7 weitere Probleme auf, die sie in dem Text gefunden haben. Diese Probleme beziehen sich hauptsächlich auf die Verwendung von Plural und die nicht genau spezifizierte Verbindung von verschiedenen Begriffen (wie den Zusammenhang der unterschiedlichen Gebührenzahlen).

Die von ihnen als am häufigsten auftretend identifizierten Probleme (Passivkonstruktionen, bestimmte Artikel ohne Referenz und Gebrauch von Synonymen) werden von RESI angegangen und zu einem Großteil auch erkannt. Eine Zusammenfassung der von RESI gefundenen Probleme (inklusive der fälschlicherweise gefundenen Probleme) befindet sich in Tabelle 6.1.

6.2. Zweiter Spezifikationstext: ESFAS

Dieser Spezifikationstext beschreibt die Funktionalität einer Drucküberwachung namens ESFAS (Engineered Safety Feature Actuation System). Er wurde von Berry et al. formuliert [9] und basiert auf einem Text von Courtois und Parnas [13]. Er ist vollständig in Quelltext 6.2 abgedruckt.

Quelltext 6.2: Zweiter Spezifikationstext: ESFAS [9, 13]

```
The system monitors the pressure and sends the safety
   injection signal when the pressurizer's pressure falls
   below a 'low' threshold.
The human operator can override system actions by turning on
   a 'Block' button and resets the manual block by pushing
   on a 'Reset' button.
A manual block is permitted if and only if the pressure is
   below a 'permit' threshold.
The manual block must be automatically reset by the system.
A manual block is effective if and only if it is executed
   before the safety injection signal is sent.
The 'Reset' button has higher priority than the 'Block'
   button.
```

Kamsties et al. haben diesen Text im Rahmen ihrer Tests zur Wirksamkeit von Inspektionen verwendet [41]. In ihrer Veröffentlichung zeigen sie nur ein Problem exemplarisch im Detail auf, das ihre Methode gefunden hat. Für sie muss der Begriff `manual block`

genauer spezifiziert werden, da sonst nicht eindeutig klar ist, ob damit nur der Zustand des `'Block' button` (gedrückt) oder der Systemzustand (der Benutzer setzt sich über das System hinweg) oder beides gemeint ist. RESI erkennt `block` als Nominalisierung, die behoben werden sollte.

Eine Übersicht über die weiteren von RESI im Text gefundenen Probleme steht in Tabelle 6.1, wobei auch hierbei fälschlicherweise erkannte Probleme einbezogen werden.

Tabelle 6.1.: Von RESI gefundene Probleme

Statistik	**Text 1**	**Text 2**
# Sätze	17	6
# Wörter	222	99
# Zeichen (ohne Leerzeichen)	1036	486
# Substantive	76	27
# Verben	36	14
	Text 1	**Text 2**
# Mehrdeutige Wörter	44	15
# Zusätzliche Bedeutungen	198	42
# Genauere Bedeutungen	12	6
# Nominalisierungen	4	4
# Unvollständig spezifizierte Prozesswörter	14	3
# Fehlende Argumente	18	5
# Gefundene Synonyme	11	2
# Quantoren	8	0
# Bestimmte Artikel	24	12
# Unbestimmte Artikel	6	6

6.3. Laufzeit

Um die Geschwindigkeit von RESI zu testen, wurde der erste Spezifikationstext (*ABC Video Rental*) automatisch präannotiert und alle Regeln auf ihn angewandt. Dabei wurde gemessen, wie lange die automatisierten Prozesse dauern, um festzustellen, ob ein

Benutzer bei der Verwendung von RESI durch die Zugriffe auf die Ontologien ausgebremst wird. Gemessen wurde auf einem Rechner mit einem Pentium-D-Prozessor (Taktfrequenz 3 Gigahertz), 3 Gigabyte Arbeitsspeicher und Windows XP als Betriebssystem.

Bei der Präannotation wurden die Annotierung der Wortarten und die Annotierung der Grundformen getrennt voneinander gemessen. Die Dauer der Präannotation der einzelnen Sätze wird durch die Boxplots in Abbildung 6.7 dargestellt. Die gesamte Präannotation dauerte für die 17 Sätze des Spezifikationstexts auf dem Testsystem weniger als 30 Sekunden. Da die Präannotation für jeden Satz unabhängig von der der anderen Sätze ist, steigt die Laufzeit der Präannotation linear mit der Länge der Spezifikation.

Bei der Regelanwendung wurde gemessen, wie lange die jeweiligen Ontologieabfragen dauern. Bei der Regel Überprüfe auf Nominalisierungen wurde demnach überprüft, wie lange es dauert, festzustellen, ob ein Substantiv eine Nominalisierung ist, und – sofern nötig – das dazugehörige Vollverb zu ermitteln. In Abbildung 6.8 wird die Dauer dieser Abfrage sowohl für alle Substantive der Spezifikation als auch nur für die als Nominalisierung erkannten Substantive gezeigt. Für die Regel Vervollständige Prozesswörter wird in Abbildung 6.9 dargestellt, wie lange es für die einzelnen Prozesswörter dauerte, ihre mit Vorschlägen gefüllten Argumentlisten aus der Ontologie zu ermitteln. In Abbildung 6.10 befindet sich die Darstellung der Dauer der Abfragen, um die Bedeutung des jeweiligen Artikels oder Quantors (Regel Überprüfe Artikel & Quantoren) mit Hilfe der Ontologie zu gewinnen. Die Regel Überprüfe auf mehrdeutige Wörter nutzt Ontologiewissen, um mögliche Bedeutungen der Wörter der Spezifikation zu erhalten. Die Dauer dieser Abfragen wird in Abbildung 6.11 gezeigt. Die Abbildung 6.12 stellt dar, wie lange es dauerte, für zwei Substantive zu überprüfen, ob sie Synonyme sind (Regel Überprüfe auf Wörter gleicher Bedeutung).

Die Dauer einer einzelnen Abfrage ist unabhängig von der Länge der Spezifikation. Diese Dauer ist bei jeder Regel so kurz, dass sich der Benutzer nicht von RESI aufgehalten fühlt, wenn er mit RESI arbeitet. Für die meisten Regeln erhöht sich die Anzahl der Abfragen linear mit der Länge der Spezifikation. Dementsprechend muss ein Benutzer auch bei längeren Spezifikationen nicht länger zwischen zwei Interaktionen warten als dies bei kürzeren Spezifikationen der Fall ist. Die Gesamtdauer der Bearbeitung einer Spezifikation mit RESI erhöht sich also linear mit der Länge der Spezifikation.

Einzige Ausnahme hiervon bildet die Regel Überprüfe auf Wörter gleicher Bedeutung. Zwar ist die einzelne Abfrage sehr schnell (siehe Abbildung 6.12), allerdings erhöht sich bei dieser Regel die Anzahl der Abfragen quadratisch mit der Länge der Spezifikation, da jedes Substantiv mit jedem anderen Substantiv aus der Spezifikation

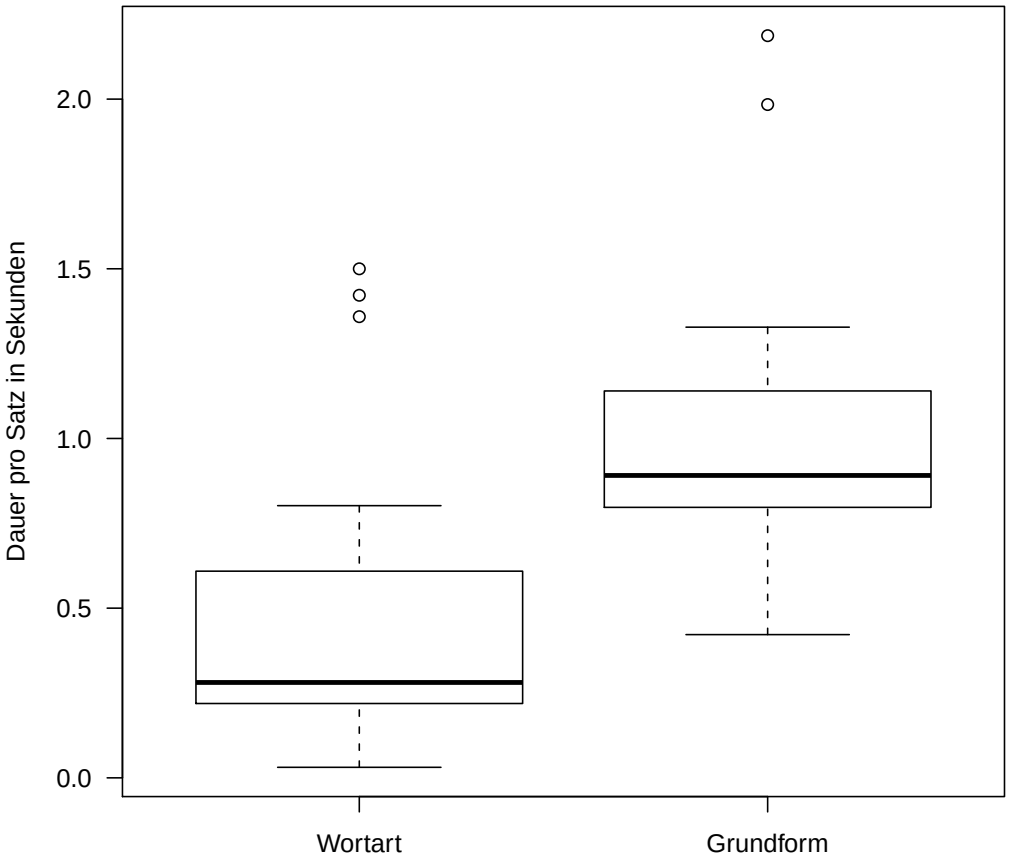

Abbildung 6.7.: Dauer der Präannotation für einzelne Sätze

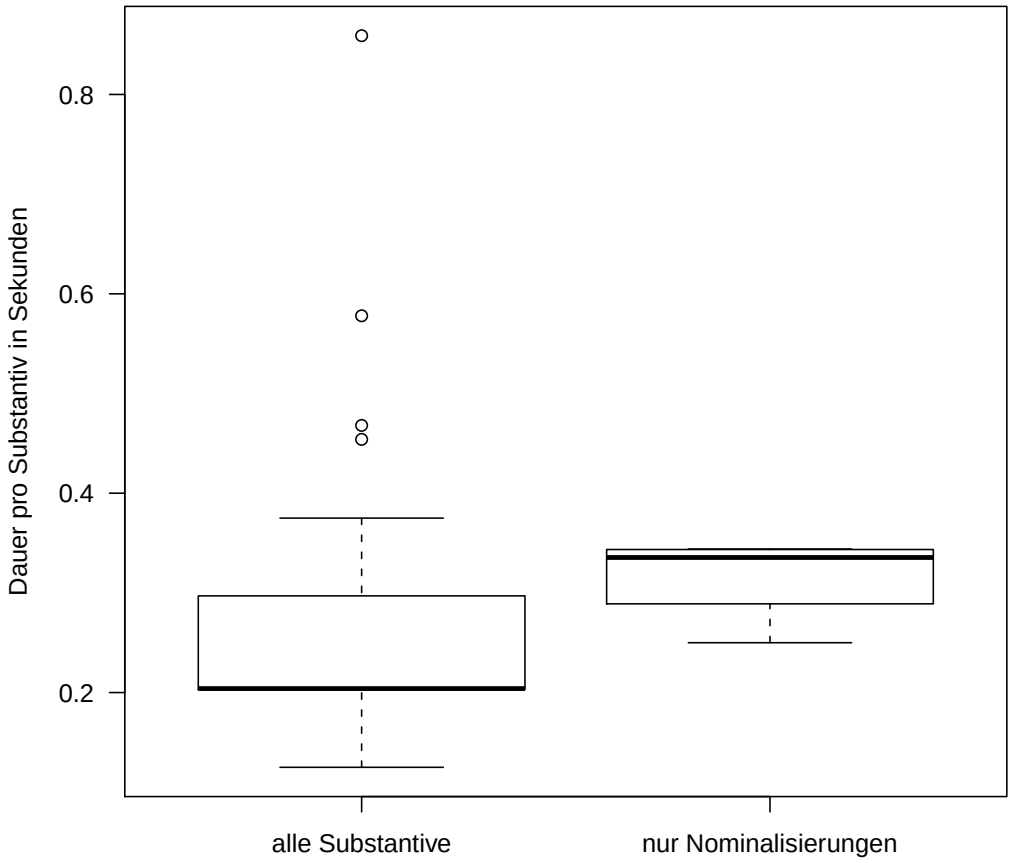

Abbildung 6.8.: Dauer der Ontologieabfrage zur Regel Überprüfe auf Nominalisierungen

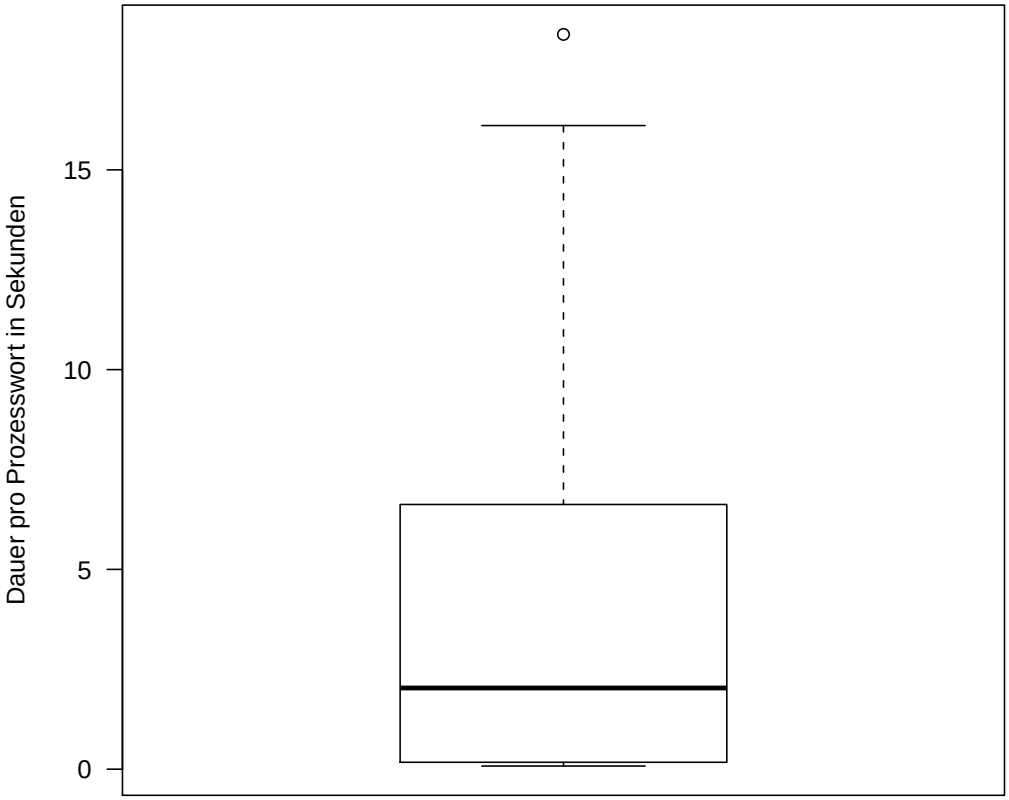

Abbildung 6.9.: Dauer der Ontologieabfrage zur Regel Vervollständige Prozesswörter

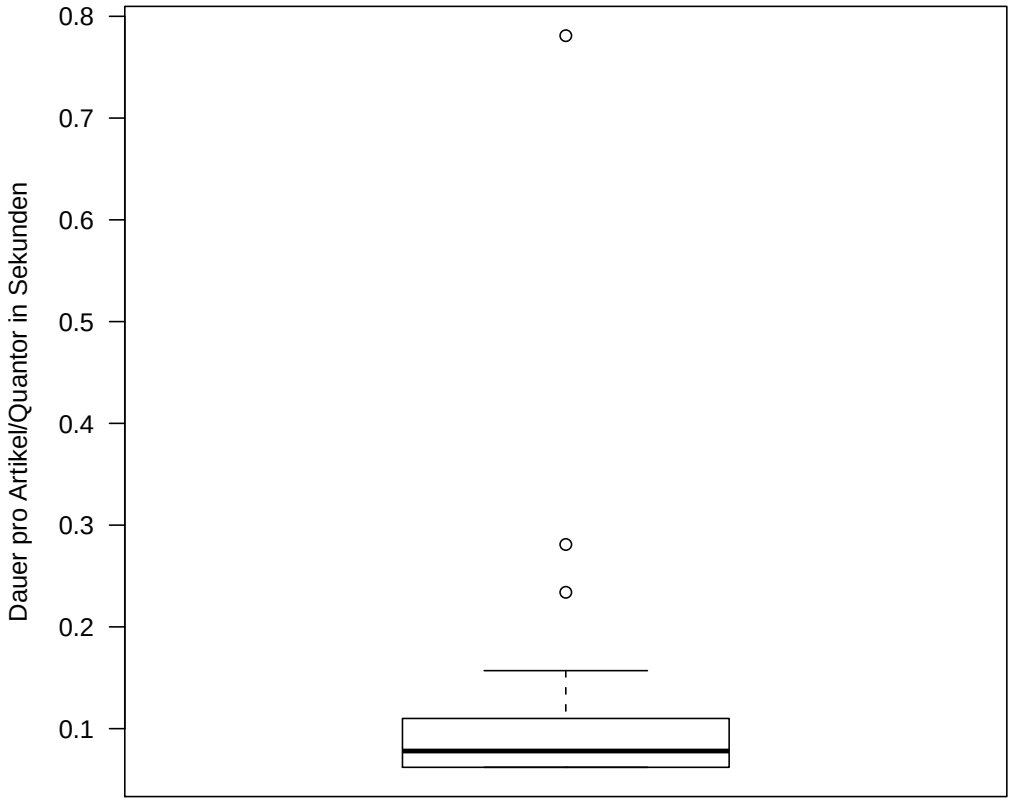

Abbildung 6.10.: Dauer der Ontologieabfrage zur Regel Überprüfe Artikel & Quantoren

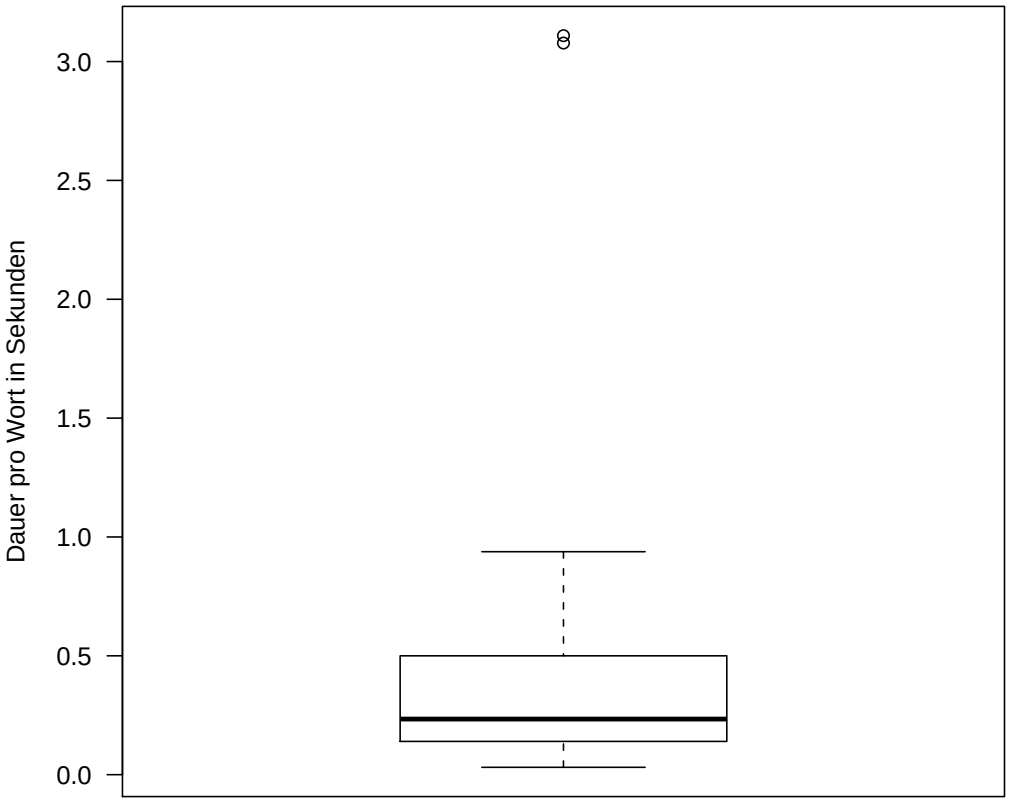

Abbildung 6.11.: Dauer der Ontologieabfrage zur Regel Überprüfe auf mehrdeutige Wörter

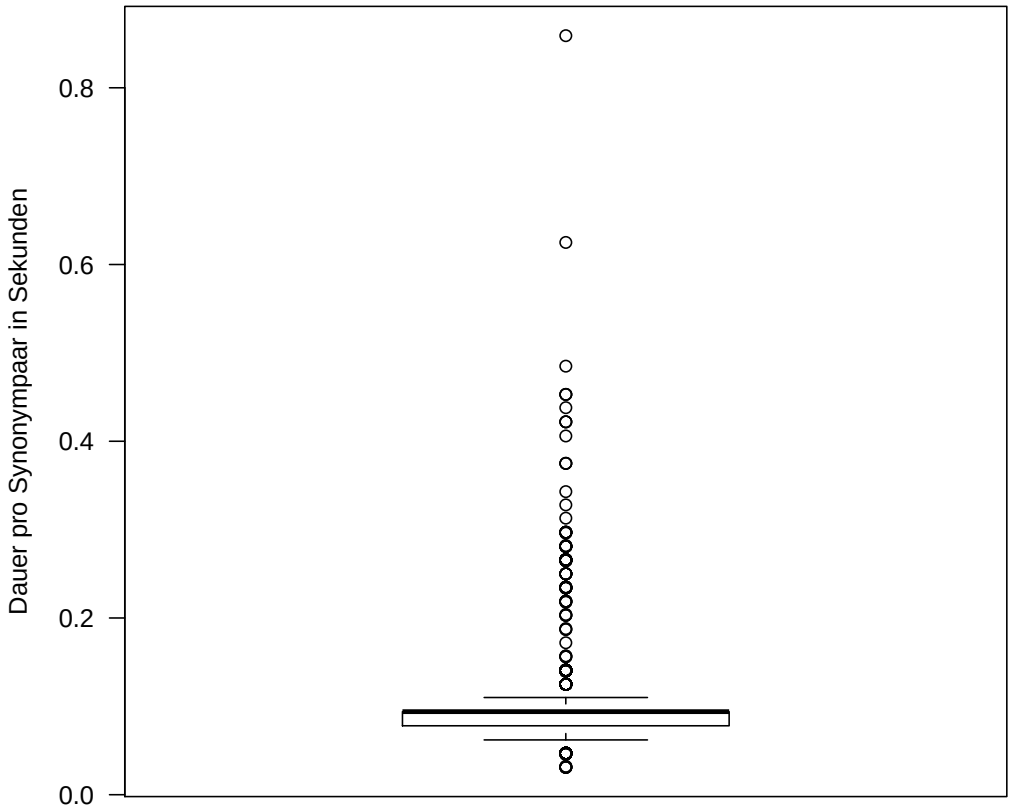

Abbildung 6.12.: Dauer der Ontologieabfrage zur Regel Überprüfe auf Wörter gleicher Bedeutung

verglichen wird. Schon bei der relativ kurzen untersuchten Beispielspezifikation mit 76 Substantiven werden theoretisch 2850 Vergleiche durchgeführt. In der Praxis sind es 2440, weil einige doppelte Vergleiche weggelassen werden. Dennoch ist diese Zahl schon hier sehr hoch und der Benutzer muss demnach zwischen zwei Interaktionen mehrere Sekunden warten. Bei längeren Spezifikationen würde sich diese Zeit noch deutlich erhöhen.

6.4. Andere Ontologien

Auf Grund ihres konzeptuellen Aufbaus und ihres Inhalts eignen sich die angebundenen Ontologien bis auf ResearchCyc nur sehr eingeschränkt für die Verwendung mit RESI. Nur WordNet unterstützt noch die Regel Überprüfe auf mehrdeutige Wörter, unterliegt dabei aber dem gleichen Problem wie auch ResearchCyc, weil es Mehrdeutigkeiten entdeckt, wo ein menschlicher Leser keine sieht. Bereits anhand des kurzen ESFAS-Textes wird deutlich, dass hier das Problem deutlich ausgeprägter ist: Mit Hilfe von WordNet werden 27 mehrdeutige Wörter entdeckt (ResearchCyc: 15), die insgesamt 191 zusätzliche Bedeutungen außer der gemeinten haben (ResearchCyc: 42).

Die Regel Überprüfe auf Wörter gleicher Bedeutung wird von allen Ontologien unterstützt. WordNet schlägt für den ersten Spezifikationstext 3 Ersetzungen und für den zweiten 1 Ersetzung vor. Keiner dieser Vorschläge ist sinnvoll. YAGO braucht selbst bei einer Suchtiefe von 2 (zur genaueren Erklärung siehe Anhang A.4.1) zur Überprüfung des ersten Spezifikationstexts je nach Rechnergeschwindigkeit ca. 10–20 Minuten, ohne dabei einen sinnvollen Vorschlag zu finden. Auch ConceptNet liefert kein vernünftiges Ergebnis. Hinzu kommt, dass mit der momentanen Implementierung die Überprüfung des ersten Spezifikationstexts mehrere Stunden dauert, was für einen Benutzer nicht zu akzeptieren ist.

Dementsprechend sind von den verwendeten Ontologien nur ResearchCyc und stark eingeschränkt WordNet produktiv verwendbar, die anderen beiden getesteten Ontologien liefern keine brauchbaren Ergebnisse und sind außerdem deutlich langsamer als ResearchCyc. Es ist möglich, dass andere Ontologien bessere Ergebnisse als die getesteten liefern, was man in der zukünftigen Entwicklung von RESI berücksichtigen sollte.

6.5. Abgrenzung

Sowohl im Vorfeld als auch im Rahmen der Evaluierung wurde klar, dass RESI nicht uneingeschränkt funktioniert. In großem Maße hängt es von der Art und der Qualität der angeschlossenen Ontologien ab. Zum einen verwenden die großen Ontologien nur die englische Sprache in großem Umfang, weshalb RESI bis jetzt ausschließlich englischsprachige Spezifikationen unterstützt. Sobald entsprechende anderssprachige Ontologien verfügbar sind und an RESI angebunden werden, unterstützt RESI auch andere Sprachen. Die Qualität der Regelanwendungen hängt direkt mit der Qualität der verwendeten Ontologie zusammen. Beispielsweise könnte die Regel Vervollständige Prozesswörter alle unvollständig spezifizierten Prozesswörter einer Spezifikation erkennen, wenn die Ontologie die entsprechenden Argumentlisten beinhaltet. Da im Rahmen dieser Arbeit aber nur eine begrenzte Anzahl von Ontologien untersucht werden konnten, ist die Anbindung weiterer Ontologien Teil der zukünftigen Entwicklung von RESI. Insbesondere die Anbindung von domänenspezifischen Ontologien ist hierbei ein interessanter Aspekt.

Des Weiteren ist es für RESI kaum möglich, direkt Manipulationen an der Spezifikation vorzunehmen. Die meisten Probleme erfordern eine Umformulierung des gesamten Satzes und können nicht durch Austauschen eines einzelnen Wortes behoben werden. Deshalb bietet RESI hauptsächlich die Möglichkeit, Informationen zur Spezifikation hinzuzufügen. Mit Hilfe dieser Informationen kann die Spezifikation anschließend vom Experten in Zusammenarbeit mit dem Kunden umformuliert werden.

RESI ist kein Werkzeug, das der Kunde anwenden soll. Es dient dazu, den Experten bei seiner Arbeit zu unterstützen und ihm Probleme aufzuzeigen, die er allein übersehen hätte. Außerdem soll es ihm erlauben, Spezifikationen schneller untersuchen zu können als durch rein manuelle Kontrolle.

7. Zusammenfassung und Ausblick

Gute Software steht und fällt mit der Qualität der Spezifikation. In textbasierten Spezifikationen treten verschiedene Arten von Problemen auf, die sich in sprachlichen Defekten manifestieren. Um einen Experten beim Finden und Beheben dieser Probleme zu unterstützen, benötigt eine entsprechende Software semantisches Wissen über die Texte. Dieses Wissen kann von Ontologien bereit gestellt werden.

RESI nutzt Ontologiewissen von verschiedenen Ontologien, um die mit der höchsten Priorität zu beseitigenden Probleme zu finden und Verbesserungsvorschläge zu liefern, wobei eine Regel von RESI sich jeweils einer Problemart annimmt. Dabei wurden vier verschiedene allgemeine Ontologien angebunden und passende Anfragen an diese erarbeitet, um das in den Ontologien gespeicherte Wissen zu verwenden.

Bei der Evaluierung zeigte sich, dass die Qualität der Ergebnisse hauptsächlich von der Qualität der befragten Ontologie abhängt. ResearchCyc lieferte dabei die besten Ergebnisse, sodass viele Probleme automatisch identifiziert und mit Verbesserungsvorschlägen versehen werden konnten. Es zeigte sich aber auch, dass RESI einen Experten nicht ersetzen, sondern lediglich unterstützen kann.

Durch die leichte Erweiterbarkeit von RESI ist es möglich, weitere als die in dieser Arbeit vorgestellten Probleme zu identifizieren und zu beseitigen, indem weitere Regeln implementiert werden. Durch die Anbindung von anderen Ontologien kann der Wissensbestand von RESI erweitert werden. Andere Darstellungsformen von Spezifikationen – wie zum Beispiel einfacher Fließtext – können ebenfalls angebunden werden.

In der Zukunft ist es interessant zu untersuchen, wie gut andere Ontologien für die Verwendung mit RESI geeignet sind. Insbesondere Domänenontologien, die sich nur der Spezifikationsdomäne annehmen, sind dabei von hohem Interesse. Auch wie sich RESI auf die Qualität von Softwarespezifikationen im Vergleich zu anderen Techniken auswirkt, gilt es zu untersuchen.

A. Ontologieanfragen

In diesem Anhang werden die Anfragen vorgestellt, die von RESI an die externen Ontologien gestellt werden, unterteilt nach Ontologien und Regeln. Die Anfragen werden anhand von konkreten Beispielen gezeigt, wobei die beispielhaft eingefügten Werte in blau dargestellt werden. Als Spezifikation, aus dem die Werte stammen, dient unser Beispielsatz aus Abschnitt 5.3:

```
Every pallet is returned after transport.
```

A.1. ResearchCyc

Anfragen an ResearchCyc werden in CycL [17] gestellt. Dabei werden feste Begriffe beziehungsweise Prädikate durch ein vorgestelltes `#$` und Variablen durch ein vorgestelltes `?` gekennzeichnet. Das in ResearchCyc gespeicherte Wissen ist in Domänen aufgeteilt, sogenannte Microtheories. Alle für RESI relevanten Anfragen werden in der Microtheory *GeneralEnglishMt* gestellt. Für einige Anfragen wird die Bedeutung der Wörter aus dem Satz benötigt. Falls die Bedeutung noch nicht ermittelt wurde, wird sie vorher durch Benutzerauswahl aus einer Liste von Bedeutungen bestimmt. Die Liste wird nach der Methode ermittelt, die in Abschnitt A.1.4 beschrieben wird.

A.1.1. Ermittlung von Verben zu Nominalisierungen

Zur Ermittlung von Nominalisierungen wird zuerst überprüft, ob es ein Wort gibt, für das die potenzielle Nominalisierung (im folgenden Beispiel `transport`) die Singularform ist:

```
(#$singular ?THEWORD "transport")
```

Dadurch erfährt man, dass `transport` die Singularform von `#$Transport-TheWord` ist.

Zusätzlich wird überprüft, ob ein Wort existiert, für das die potenzielle Nominalisierung ein Singularetantum[1] ist:

```
(#$massNumber ?THEWORD "transport")
```

Für unser Beispiel liefert ResearchCyc keine Ergebnisse.

Für die ermittelten Wörter wird überprüft, ob es ein Verb mit der Bedeutung gibt, die die potenzielle Nominalisierung hat (im Beispiel `#$TransportationEvent`):

```
(#$denotation #$Transport-TheWord #$Verb ?SENSECOUNTER
   #$TransportationEvent)
```

Wenn die zurückgegebene Liste nicht leer ist, kann der Infinitiv des Vollverbs ermittelt werden, der dann zurückgeliefert wird:

```
(#$infinitive #$Transport-TheWord ?RESULT)
```

A.1.2. Ermittlung von Argumentlisten zu Prozesswörtern

Die Argumentlisten können über eine einzige Anfrage ermittelt werden. Allerdings muss dafür vorher aus dem Prozesswort (im Beispiel `return`) eine Wortkonstante gebaut werden, indem nur der erste Buchstabe groß geschrieben und `-TheWord` angehängt wird:

```
(#$verbSemTrans #$Return-TheWord ?SENSECOUNTER ?FRAMETYPE
   ?FRAME)
```

In der Variable `?FRAME` steht dann die Argumentliste im folgenden Format:

```
(#$and
   (#$objectGiven :ACTION :OBJECT)
   (#$isa :ACTION #$ReturningSomething)
   (#$giver :ACTION :SUBJECT)
   (#$givee :ACTION :OBLIQUE-OBJECT))
```

[1] Ein Singularetantum ist ein Substantiv, das ausschließlich im Singular gebräuchlich ist wie zum Beispiel Müll oder Laub.

Die Zeile mit dem Prädikat `#$isa` gibt die Bedeutung an, die das Wort in diesem Zusammenhang hat. Nur wenn diese Bedeutung mit der Bedeutung des Prozessworts übereinstimmt, wird die Argumentsliste zurückgegeben. In den anderen Zeilen stehen die Argumente mit ihren semantischen (zum Beispiel `#$objectGiven`) und ihren syntaktischen (zum Beispiel `:OBJECT`) Rollen.

Um die semantischen Rollen für den Benutzer von RESI verständlicher zu machen, werden auch die Erklärungen für die semantischen Rollen ermittelt:

```
(#$comment #$giver ?COMMENT)
```

Um die aus diesen Informationen generierten Argumentlisten noch mit Vorschlägen aus dem entsprechenden Satz zu füllen, wird für jedes Argument ermittelt, womit es gefüllt werden kann:

```
(#$arg2Isa #$objectGiven ?WHATFITS)
```

Dadurch erfährt man zum Beispiel, dass ein `#$objectGiven` jedes `#$SomethingExisting` sein kann.

Abschließend wird für jedes Wort des Satzes getestet, ob es als eines der Argumente dienen kann. Dazu wird überprüft, ob das, was als Argument verwendet werden kann (`#$SomethingExisting`) eine Verallgemeinerung der Bedeutung des Wortes aus dem Satz (`#$Pallet-TransportationConstruct` für `pallet`) ist:

```
(#$genls #$Pallet-TransportationConstruct
   #$SomethingExisting)
```

Wenn diese Abfrage als wahr angegeben wird, wird das entsprechende Wort als Vorschlag in die Argumentliste eingetragen. Nach Überprüfung aller Argumente und aller Wörter wird die Liste zurückgeliefert.

A.1.3. Ermittlung von Bedeutungen von Artikeln und Quantoren

Nach Bildung einer Wortkonstante (ersten Buchstaben groß schreiben und `-TheWord` anhängen) aus dem Artikel oder Quantor (im Beispiel `Every`) kann durch folgende Anfrage die Bedeutung ermittelt werden:

```
(#$denotation #$Every-TheWord ?DETERMINERTYPE ?SENSECOUNTER
   ?CONSTANTORNUMBERVALUE)
```

In der Variable `?DETERMINERTYPE` steht anschließend, von welchem Typ der Artikel oder Quantor ist:

- `#$Determiner-Definite` für bestimmte Artikel
- `#$Determiner-Indefinite` für unbestimmte Artikel
- `#$Determiner-Central` für Quantoren, wobei hier in `?CONSTANTORNUMBERVALUE` die Bedeutung angegeben wird:
  - `#$No-NLAttr` für *0*
  - `#$Each-NLAttr` oder `#$Each-NLAttr` für *Alle*
  - sonstige Konstanten für *Beliebig*
- `#$Number-SP` für Zahlen, wobei hier in `?CONSTANTORNUMBERVALUE` der Wert als Ganzzahl angegeben wird

A.1.4. Ermittlung von möglichen Wortbedeutungen

Zur Ermittlung aller möglichen Wortbedeutungen wird eine normale Suchanfrage an ResearchCyc gestellt, wie sie auch über das Suchformular gestellt wird, wenn mit einem Browser auf ResearchCyc zugegriffen wird. Für jede gefundene Bedeutung wird noch die Beschreibung über folgende Anfrage ermittelt (im Beispiel für `#$Transportation-Event`):

```
(#$comment #$TransportationEvent ?COMMENT)
```

A.1.5. Ermittlung von Wortähnlichkeiten

Wortähnlichkeit wird ermittelt durch die Fragestellung, ob das eine Wort eine Verallgemeinerung des anderen Wortes ist. Dazu dient folgende Anfrage mit den Bedeutungen der beiden Wörter (im Beispiel `#$Pallet-TransportationConstruct` für `pallet` und `#$TransportationEvent` für `transport`):

```
(#$genls #$Pallet-TransportationConstruct
   #$TransportationEvent)
```

Wenn diese Abfrage als wahr zurückgegeben wird, wird eine Ersetzung des spezifischeren Begriffs mit dem allgemeineren mit einem Konfidenzwert von 100% vorgeschlagen. Ansonsten wird die Abfrage mit den gleichen Worten in umgekehrter Reihenfolge wiederholt, um herauszufinden, ob eine Verallgemeinerung in der anderen Richtung vorliegt.

A.2. WordNet

WordNet wird über eine Java-Schnittstelle angesprochen. Nachfolgend werden die Anfragen an das Kommandozeilenprogramm angegeben, die die Ergebnisse liefern, die denen der Anfragen an die Java-Schnittstelle in RESI entsprechen.

A.2.1. Ermittlung von möglichen Wortbedeutungen

Zur Ermittlung der Wortbedeutungen werden alle möglichen Bedeutungen mit Beschreibungen angefragt. Dabei wird nach der passenden Wortart gefiltert. Das Kommandozeilenprogramm kann diese Filterung nicht automatisch vornehmen, aber der folgende Befehl liefert alle Bedeutungen für ein Wort mit Beschreibungen (hier für das Wort `transport`):

```
wordnet transport -over
```

A.2.2. Ermittlung von Wortähnlichkeiten

Die Wortähnlichkeit wird in WordNet darüber ermittelt, ob ein Wort ein Oberbegriff des anderen ist. Um alle Oberbegriffe eines Wortes zu ermitteln, kann man den folgenden Kommandozeilenbefehl verwenden (für das Wort `pallet`):

```
wordnet pallet -hypen
```

Wenn das andere Wort in dieser Liste als Oberbegriff auftaucht, wird vorgeschlagen, den Unterbegriff mit dem Oberbegriff zu ersetzen. Der Konfidenzwert wird dabei über die „Verzweigung“ bestimmt. Auf dem Weg vom Unterbegriff zum Oberbegriff wird auf jeder Ebene überprüft, wie viele direkte Oberbegriffe dieses Wort hat. Der Prozentwert

dieser Ebene wird durch

$$\frac{1}{\text{Anzahl der Bedeutungen}}$$

berechnet. Der gesamte Konfidenzwert ergibt sich durch Multiplikation der Prozentwerte der einzelnen Ebenen.

A.3. ConceptNet

Auf ConceptNet wird ausschließlich über den eigenen Server zugegriffen, der die Anfragen an die mit ConceptNet mitgelieferte Python-Schnittstelle weiterreicht. Der Quelltext des Servers befindet sich in Quelltext B.3.

A.3.1. Ermittlung von Wortähnlichkeiten

Zur Ermittlung der Wortähnlichkeiten wird überprüft, ob ein Wort der Oberbegriff des anderen ist. Dazu wird die Funktion `get_fwd_relations` wiederholt mit `IsA` als zweitem Argument aufgerufen. Als erstes Argument dient das Wort, das der potentielle Unterbegriff ist. Die zurückgegebenen Verbindungen werden dann weiter verwendet. Sie zeigen auf jeweils einen direkten Oberbegriff, der wieder als erstes Argument für die Funktion verwendet wird. Dies wird immer wiederholt, bis entweder ein direkter Oberbegriff mit dem potentiellen Oberbegriff übereinstimmt oder die maximale Suchtiefe (die konfigurierbar ist) erreicht wurde. Bei Erfolg wird auf jeder Ebene ein Prozentwert durch

$$\frac{\text{Qualitätsmaßzahl der richtigen Verbindung}}{\sum \text{Qualitätsmaßzahlen aller Verbindungen}}$$

berechnet. Das Produkt dieser Prozentwerte ist der zurückgegebene Konfidenzwert.

A.4. YAGO

Zur Kommunikation mit YAGO wird die mitgelieferte Java-Schnittstelle verwendet. Dabei besteht eine Anfrage aus einem oder mehreren sogenannter Templates, die einer Zeile der Anfrage entsprechen. Ein Template besteht aus vier Teilen: der Template-ID (einer Variable), einem Prädikat und zwei Argumenten. Variablen werden mit vorangestelltem `?` gekennzeichnet und gelten templateübergreifend. Nach einer erfolgreichen Abfrage ist jede Variable mit einem Wert gefüllt.

A.4.1. Ermittlung von Wortähnlichkeiten

Auch mit Hilfe von YAGO wird Wortähnlichkeit über Oberbegriffe ermittelt. Dabei findet folgendes Abfrageschema Anwendung (im Beispiel ist `tape` ein potentieller Oberbegriff von `video`):

```
?id0 IsA  video ?x1
?id1 subClassOf  ?x1 ?x2
?id2 subClassOf  ?x2 tape
```

Es wird begonnen ohne Hilfsvariablen (im Beispiel beginnen sie mit `?x`) und nur mit einem Template (ohne die Templates mit dem Prädikat `subClassOf`). Solange kein Ergebnis gefunden wird, wird die Anfrage wiederholt mit steigender Zahl von Hilfsvariablen und Templates mit dem Prädikat `subClassOf` bis zu einer konfigurierbaren maximalen Suchtiefe. In jedem Durchlauf wird dabei das Prädikat des ersten Templates variiert mit den Werten `isA`, `type` und `subClassOf`, das Prädikat der anderen Templates bleibt `subClassOf`.

B. Quelltexte

Quelltext B.1: *GraphSimilarMeaning* (Graphschnittstelle für die Regel Überprüfe auf Wörter gleicher Bedeutung)

```
package de.uka.ipd.resi.graphinterface;

import java.util.Set;

import de.uka.ipd.resi.Word;
import de.uka.ipd.resi.exceptions.SyntaxException;
import de.uka.ipd.resi.exceptions.WordNotFoundException;

/**
 * Graph interface for RuleCompareNouns.
 *
 * @see de.uka.ipd.resi.ruleimpl.RuleSimilarMeaning
 * @author Torben Brumm
 */
public interface GraphSimilarMeaning {

        /**
         * Returns all nouns in the graph.
         *
         * @return All nouns.
         * @throws SyntaxException
         */
        public Set<Word> getAllNouns() throws SyntaxException;

        /**
         * Replaces the old noun with the new noun.
         *
         * @param oldNoun Noun which is replaced.
         * @param newNoun Noun which replaces the other noun.
         * @throws WordNotFoundException
         */
```

```
        public void replaceNoun(Word oldNoun, Word newNoun)
            throws WordNotFoundException;
}
```

Quelltext B.2: *OntologySimilarMeaning* (Ontologieschnittstelle für die Regel Überprüfe auf Wörter gleicher Bedeutung)

```
package de.uka.ipd.resi.ontologyinterface;

import de.uka.ipd.resi.Word;
import de.uka.ipd.resi.exceptions.NotConnectedException;
import de.uka.ipd.resi.exceptions.WordNotFoundException;

/**
 * Ontology interface for RuleSimilarMeaning.
 *
 * @see de.uka.ipd.resi.ruleimpl.RuleSimilarMeaning
 * @author Torben Brumm
 */
public interface OntologySimilarMeaning {

        /**
         * Returns the similarity of two nouns. The more
            similar they are, the greater the absolute value of
             the returned
         * value is (max is 1). If the returned value is
            greater than 0, the first noun is more general, if
            it is less than
         * 0, the second noun is more general.
         *
         * @param noun1 First noun.
         * @param noun2 Second noun.
         * @return Similarity (on a scale from -1 to 1).
         * @throws WordNotFoundException
         * @throws NotConnectedException
         */
        public float getSimilarity(Word noun1, Word noun2)
            throws WordNotFoundException, NotConnectedException;
}
```

Quelltext B.3: ConceptNet-XMLRPC-Server, in Python geschrieben

```
import DocXMLRPCServer
from csamoa.conceptnet.models import *
from collections import deque

class StemBean:
    def __init__(self, stem):
        self.language = stem.language.id
        self.name = stem.text

class RelationBean:
    def __init__(self, relation):
        self.predicateType = relation.predtype.name
        self.stem1 = StemBean(relation.stem1)
        self.stem2 = StemBean(relation.stem2)
        self.polarity = relation.polarity
        self.modality = relation.modality
        self.score = relation.score


def get_fwd_relations(word, type='All', minimumscore=3):
        lang_en = Language.objects.get(id='en')
        stem = Concept.get(word, lang_en)
        rel = deque();
        for a in stem.get_fwd_relations():
                if (type == 'All') or (a.predtype.name == type)
                    :
                        if (a.score >= minimumscore):
                                rel.append(a)
        return [RelationBean(a) for a in rel]

def get_rev_relations(word, type='All', minimumscore=3):
        lang_en = Language.objects.get(id='en')
        stem = Concept.get(word, lang_en)
        rel = deque();
        for a in stem.get_rev_relations():
                if (type == 'All') or (a.predtype.name == type)
                    :
                        if (a.score >= minimumscore):
                                rel.append(a)
        return [RelationBean(a) for a in rel]
```

```
print "Starting XML-RPC Server"
port = 9854
xmlrpc = DocXMLRPCServer.DocXMLRPCServer(('',port))
print "Now serving on localhost port %s!"%str(port)
xmlrpc.register_introspection_functions()
xmlrpc.register_function(get_fwd_relations, 'ConceptNet.
    getFwdRelations')
xmlrpc.register_function(get_rev_relations, 'ConceptNet.
    getRevRelations')
xmlrpc.serve_forever()
```

C. ResearchCyc-Konstanten

In diesem Anhang befinden sich die Erklärungen zu Begriffen aus der ResearchCyc-Datenbank. [16]

Conveyance

A collection of solid tangible objects each instance of which is used for moving partially tangible things. A Conveyance could be a car, ship, plane, or other vehicle for transporting people; it could be a conveyor belt or a grocery bag for moving goods; it could be a gun, a bow, or a cannon for launching projectiles. Notable specializations include TransportationDevice, whose instances actually move along with the things they transport, and Conveyance-Stationary, whose instances remain stationary while moving other things. Note that not all conveyances are artifacts, as (e.g.) horses and rivers can be used to convey things. See Conveying-Generic, TransportationEvent, and Conveying-Stationary for the different kinds of conveying events.

givee

This predicate relates the recipient of a gift to the event in which s/he/it is given that gift. (givee GIVING AGENT) means that the Agent-PartiallyTangible AGENT is the 'givee' in the GivingSomething event GIVING. For the gift-giver, see giver.

giver

This predicate relates a giver to the event in which s/he/it gives. (giver GIVING AGENT) means that the Agent-PartiallyTangible AGENT is the giver in the Giving-Something event GIVING. For the recipient, see givee.

objectGiven

(objectGiven EVENT OBJECT) holds if the SomethingExisting OBJECT is given by the giver to the givee during a GivingSomething.

objectOfPossessionTransfer

objectOfPossessionTransfer relates an event (more specifically, an instance of ChangeInUserRights) to an object (an instance of SomethingExisting) the user rights to which are transferred during that event. (objectOfPossessionTransfer EVENT OBJECT) means that in EVENT, all or some rights to use OBJECT are transferred from one agent (the fromPossessor) to another (the toPossessor). Since EVENT is an instance of ChangeInUserRights, it could be (among other things) a buying, renting, lending, or repossessing.

performedBy

An AgentiveRole (q.v.) predicate that relates an action (see Action) to an agent (see Agent-PartiallyTangible) who performs it deliberately, i.e. intentionally and volitionally. (performedBy ACT DOER) means that DOER deliberately does ACT. For example, (performedBy #$AssassinationOfPresidentLincoln JohnWilkesBooth) holds. Note that an action can have multiple deliberate performers. See also the generalizations deliberateActors and doneBy.

ReturningSomething

A specialization of GivingSomething. In each instance of ReturningSomething, a tangible thing or object (see objectOfPossessionTransfer) is returned by an Agent-PartiallyTangible to another Agent-PartiallyTangible who is the original possessor/owner of the object. An instance of ReturningSomething is generally an event following an instance of Renting or BorrowingSomething where the same object is involved. It could also take place after events which are instances of Buying, Bartering, etc.

TransmittingSomething

A collection of events; a subcollection of CausingAnotherObjectsTranslationalMotion. In each TransmittingSomething, something sends something else from one person or place to another.

TransportationDevice

A specialization of both Conveyance and PhysicalDevice. Each instance of TransportationDevice is an artifact designed to move an object from one location to another, by (for example) carrying, pulling, or pushing the transported object. Instances of this collection may or may not have their own power source (see SelfPoweredDevice). Those that do, such as automobiles and speedboats, constitute the specialization TransportationDevice-Vehicle. Other transportation devices (for example, instances of Wheelbarrow or Bicycle) require an external motive force. Because transporter and transportees are specializations of objectMoving, it follows that any object in the role of transporter moves as a whole with those objects playing the role of transportees. Consequently, since any instance of TransportationDevice has playing the role of transporter as its intended function, stationary objects which cause motion, such as conveyor belts, escalators, rocket launchers, and slingshots, are excluded from the collection TransportationDevice. Although they facilitate travel, ice skates, shoes, skis and other instances of WearableConveyance are also excluded from the collection TransportationDevice, since they are devices which are worn rather than ridden on, ridden with, or ridden in.

TransportationEvent

A specialization of both Conveying-Generic and Translation-Complete (qq.v.), instances of which are events in which one or more objects transport one or more other objects. Each instance of TransportationEvent is an event in which an object (in the role of transporter) aids in the translational movement of another object (having the role of transportees), so that both objects move together along the same complete pathway (see motionPathway-Complete). Optionally, one of these objects, or some third object moving along with them, provides the force to make the movement happen (see providerOfMotiveForce).

Examples of transportation events include automobile transportation, riding a bicycle, dogs pulling goods on a sled, a wagon with groceries rolling down a hill, a person carrying

clothes in a suitcase, etc. In that last case, note that the transporter is the suitcase, not the person.

Note that the transporter in a transportation event need not be in motion relative to its destination throughout the transportation event; an automobile transportation is a single transportation event even if it has sub-events in which the driver and all of the passengers disembark while the car is parked and refueled. A single transportation event may also have more than one transporter. For example, a sofa may be transported accross a living room floor by two people working together.

Events which are not TransportationEvents include a river conveying some flotsam, the wind blowing a leaf, a conveyor belt moving a widget to the next person on an assembly line, or a walking beast of burden that is carrying nothing. The first three of these negative examples are instances of Conveying-Stationary, since the would-be transporter doesn't actually change its overall location (see conveyor-Stationary); in the fourth case the unburdened beast has no transportees and the event is an instance of AnimalWalkingProcess.

Literaturverzeichnis

[1] IEEE recommended practice for software requirements specifications. In: *IEEE Std 830-1998* (1998), Okt. 3, 7

[2] Ackerman, A. ; Buchwald, L. ; Lewski, F. : Software inspections: an effective verification process. In: *Software, IEEE* 6 (1989), Mai, Nr. 3, S. 31–36. http://dx.doi.org/10.1109/52.28121. – DOI 10.1109/52.28121. – ISSN 0740–7459 9

[3] Bandler, R. : *Metasprache und Psychotherapie: Die Struktur der Magie I.* Junfermann, 1994. – ISBN 3–87387–186–6 8

[4] Berry, D. M.: The importance of ignorance in requirements engineering: An earlier sighting and a revisitation. In: *Journal of Systems and Software* 60 (2002), Nr. 1, 83 - 85. http://dx.doi.org/10.1016/S0164-1212(01)00103-0. – DOI 10.1016/S0164–1212(01)00103–0. – ISSN 0164–1212 45

[5] Berry, D. M. ; Bucchiarone, A. ; Gnesi, S. ; Trentanni, G. : A New Quality Model for Natural Language Requirements Specifications. (2008). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.5268. – [abgerufen am 24.07.2009] 11

[6] Berry, D. M. ; Kamsties, E. : The Dangerous 'All' in Specifications. In: *IWSSD '00: Proceedings of the 10th International Workshop on Software Specification and Design.* Washington, DC, USA : IEEE Computer Society, 2000. – ISBN 0–7695–0884–7, S. 191 9, 17

[7] *Kapitel* Ambiguity in Requirements Specification. In: Berry, D. M. ; Kamsties, E. : *The Springer International Series in Engineering and Computer Science.* Bd. 753: *Perspectives on Software Requirements.* Springer, 2003. – ISBN 978–1–4020–7625–1, S. 7–44 8

[8] Berry, D. M. ; Kamsties, E. : The Syntactically Dangerous All and Plural in Specifications. In: *IEEE Softw.* 22 (2005), Nr. 1, S. 55–57. http://dx.doi.org/10.1109/MS.2005.22. – DOI 10.1109/MS.2005.22. – ISSN 0740–7459 9

[9] Berry, D. M. ; Kamsties, E. ; Krieger, M. M.: *From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity - A Handbook*. http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf. Version: Nov. 2003. – [abgerufen am 24.07.2009] xiii, 8, 46

[10] Brooks, F. P. Jr.: No Silver Bullet: Essence and Accidents of Software Engineering. In: *Computer* 20 (1987), Nr. 4, S. 10–19. http://dx.doi.org/10.1109/MC.1987.1663532. – DOI 10.1109/MC.1987.1663532. – ISSN 0018–9162 3

[11] Chantree, F. ; Nuseibeh, B. ; Roeck, A. de ; Willis, A. : Identifying Nocuous Ambiguities in Natural Language Requirements. In: *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 0–7695–2555–5, S. 56–65 11

[12] Cheng, B. H. C. ; Atlee, J. M.: Research Directions in Requirements Engineering. In: *ICSE '07: Proceedings of the 29th International Conference on Software Engineering (Future of Software Engineering)*, 2007. – ISBN 0–7695–2829–5, S. 285–303 ix, 5

[13] Courtois, P.-J. ; Parnas, D. L.: Documentation for safety critical software. In: *ICSE '93: Proceedings of the 15th international Conference on Software Engineering*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1993. – ISBN 0–89791–588–7, S. 315–323 xiii, 46

[14] Cycorp Inc.: *Cyc*. http://www.cyc.com/. – [abgerufen am 24.07.2009] 13

[15] Cycorp Inc.: *OpenCyc*. http://www.opencyc.org/. – [abgerufen am 24.07.2009] 13, 38

[16] Cycorp Inc.: *ResearchCyc*. http://research.cyc.com/. – [abgerufen am 24.07.2009] 13, 27, 71

[17] Cycorp Inc.: *The Syntax of CycL*. http://www.cyc.com/doc/handbook/oe/02-the-syntax-of-cycl.html. – [abgerufen am 24.07.2009] 13, 59

[18] Davis, A. ; Overmyer, S. ; Jordan, K. ; Caruso, J. ; Dandashi, F. ; Dinh, A. ; Kincaid, G. ; Ledeboer, G. ; Reynolds, P. ; Sitaram, P. ; Ta, A. ; Theofanos, M. : Identifying and measuring quality in a software requirements specification. In: *Proceedings of the First International Software Metrics Symposium*, 1993. – ISBN 0–8186–3740–4, S. 141–152 10

[19] Dawson, L. ; Swatman, P. : The use of object-oriented models in requirements

engineering: a field study. In: *ICIS '99: Proceeding of the 20th international conference on Information Systems*. Atlanta, GA, USA : Association for Information Systems, 1999. – ISBN ICIS1999–X, S. 260–273 3

[20] Denger, C. ; Berry, D. M. ; Kamsties, E. : Higher Quality Requirements Specifications through Natural Language Patterns. In: *SWSTE '03: Proceedings of the IEEE International Conference on Software-Science, Technology & Engineering*. Washington, DC, USA : IEEE Computer Society, 2003. – ISBN 0–7695–2047–2, S. 80 10

[21] Ebert, C. : *Systematisches Requirements Engineering und Management*. 1. Auflage. dpunkt.verlag, 2005. – ISBN 978–3–89864–546–1 20

[22] Ecma International: *Standard ECMA-334: C# Language Specification*. http://www.ecma-international.org/publications/standards/Ecma-334.htm. – [abgerufen am 24.07.2009] 37

[23] Fabbrini, F. ; Fusani, M. ; Gnesi, S. ; Lami, G. : The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool. In: *SEW '01: Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*. Washington, DC, USA : IEEE Computer Society, 2001. – ISBN 0–7695–1456–1, S. 97 11

[24] Fagan, M. E.: Design and Code Inspections to Reduce Errors in Program Development. In: *IBM Systems Journal* 15 (1976), Nr. 3, S. 182–211 9

[25] Fantechi, A. ; Gnesi, S. ; Lami, G. ; Maccari, A. : Application of Linguistic Techniques for Use Case Analysis. In: *RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1465–0, S. 157–164 12

[26] Frijters, J. : *IKVM.NET*. http://www.ikvm.net/. – [abgerufen am 24.07.2009] 37

[27] Fuchs, N. E. ; Schwertel, U. ; Schwitter, R. : Attempto Controlled English — Not Just Another Logic Specification Language. In: *Lecture Notes in Computer Science* 1559 (1999), S. 1–20. http://dx.doi.org/10.1007/3-540-48958-4_1. – DOI 10.1007/3–540–48958–4_1. – ISSN 0302–9743 10

[28] Geiss, R. ; Batz, G. V. ; Grund, D. ; Hack, S. ; Szalkowski, A. M.: GrGen: A Fast SPO-Based Graph Rewriting Tool. In: Corradini, A. (Hrsg.) ; Ehrig, H. (Hrsg.) ; Montanari, U. (Hrsg.) ; Ribeiro, L. (Hrsg.) ; Rozenberg, G. (Hrsg.):

Graph Transformations - ICGT 2006 Bd. 4178, Springer. – ISBN 3–540–38870–2, 383 – 397 27, 37

[29] Gelhausen, T. ; Derre, B. ; Geiss, R. : Customizing GrGen.NET for Model Transformation. In: *GRaMoT '08: Proceedings of the 3rd International Workshop on Graph and Model Transformation.* ACM. – ISBN 978–1–60558–033–3, 17-24 7

[30] Gelhausen, T. ; Derre, B. ; Landhäusser, M. ; Brumm, T. ; Körner, S. J.: *SaleMX project website.* http://svn.ipd.uni-karlsruhe.de/trac/mx/wiki. Version: 2009. – [abgerufen am 24.07.2009] 5, 37

[31] Gelhausen, T. ; Derre, B. ; Landhäusser, M. ; Brumm, T. ; Körner, S. J.: *SaleMX project website - The thematic roles of SALE.* http://svn.ipd.uni-karlsruhe.de/trac/mx/wiki/MX/SALE/ThematicRoles. Version: 2009. – [abgerufen am 24.07.2009] 6

[32] Gelhausen, T. ; Landhäusser, M. ; Körner, S. : Automatic Checklist Creation for the Assessment of UML Models. (2008), Sept. http://www.ipd.uka.de/Tichy/uploads/publikationen/191/EduSympGLK2008.pdf. – [abgerufen am 24.07.2009] 7

[33] Gelhausen, T. ; Tichy, W. F.: Thematic Role Based Generation of UML Models from Real World Requirements. In: *ICSC 2007: Proceedings of the International Conference on Semantic Computing*, 2007. – ISBN 978–0–7695–2997–4, S. 282–289 6

[34] Havasi, C. ; Speer, R. ; Alonso, J. : ConceptNet 3: a Flexible, Multilingual Semantic Network for Common Sense Knowledge. In: *RANLP'2007: Proceedings of the International Conference on Recent Advances in Natural Language Processing.* INCOMA Ltd.. – ISBN 978–954–91743–7–3 14, 27

[35] Heath, C. H. . D.: *Made to Stick: Why Some Ideas Survive and Others Die.* 1. Auflage. Random House, 2007. – ISBN 978–1400064281 4

[36] Heitmeyer, C. L.: Software Cost Reduction. In: *Encyclopedia of Software Engineering* 2 (2002), Jan. ISBN 978–0471377375 3

[37] ISO: *ISO 8807: Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour.* http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=16258. Version: 1989. – [abgerufen am 24.07.2009] 3

[38] ITU-T: *Recommendation Z.100 (08/2002): Specification and Description Language (SDL)*. http://www.itu.int/ITU-T/studygroups/com17/languages/Z100.pdf. Version: 2002. – [abgerufen am 24.07.2009] 3

[39] Kaiya, H. ; Saeki, M. : Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach. In: *QSIC 2005: Proceedings of the Fifth International Conference on Quality Software*, 2005. – ISBN 0–7695–2472–9, S. 223–230 12

[40] Kaiya, H. ; Saeki, M. : Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In: *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference*, 2006. – ISBN 0–7695–2555–5, S. 189–198 12

[41] Kamsties, E. ; Berry, D. M. ; Paech, B. : Detecting Ambiguities in Requirements Documents Using Inspections. In: Lawford, M. (Hrsg.) ; Parnas, D. L. (Hrsg.): *WISE'01: Proceedings of the First Workshop on Inspection in Software Engineering*, 2001, S. 68–80 9, 46

[42] Kamsties, E. ; Knethen, A. V. ; Philipps, J. ; Schätz, B. : An Empirical Investigation of the Defect Detection Capabilities of Requirements Specification Languages. In: *EMMSAD'01: Proceedings of the Sixth CAiSE/IFIP8.1 International Workshop on Evaluation of Modelling Methods in Systems Analysis and Design*, 2001 10

[43] Kamsties, E. ; Paech, B. : Taming Ambiguity in Natural Language Requirements. In: *ICSSEA 2000: Proceedings of the 13th International Conference Software and Systems Engineering and their Applications* Bd. 2, 2000 10

[44] Kiyavitskaya, N. ; Zeni, N. ; Mich, L. ; Berry, D. M.: Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. In: *Requirements Engineering* 13 (2008), Nr. 3, S. 207–239. – ISSN 0947–3602 xiii, 12, 20, 41

[45] Körner, S. J.: *AutoModel*. http://www.ipd.uka.de/~koerner/. Version: 2009. – [abgerufen am 24.07.2009] 5, 38

[46] Körner, S. J. ; Gelhausen, T. : Improving Automatic Model Creation using Ontologies. In: Knowledge Systems Institute (Hrsg.): *SEKE'2008: Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering*. – ISBN 1–891706–22–5, 691-696 6

[47] Lamsweerde, A. van: *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009. – ISBN 0470012706 3

[48] Leffingwell, D. ; Widrig, D. : *Managing Software Requirements: A Use Case Approach*. Pearson Education, 2003. – ISBN 032112247X 3

[49] Luisa, M. ; Mariangela, F. ; Pierluigi, I. : Market research for requirements analysis using linguistic tools. In: *Requirements Engineering* 9 (2004), Nr. 1, S. 40–56. http://dx.doi.org/http://dx.doi.org/10.1007/s00766-003-0179-8. – DOI http://dx.doi.org/10.1007/s00766–003–0179–8. – ISSN 0947–3602 3

[50] Marcus, M. P. ; Marcinkiewicz, M. A. ; Santorini, B. : Building a large annotated corpus of English: the Penn Treebank. In: *Computational Linguistics* 19 (1993), Nr. 2, S. 313–330. – ISSN 0891–2017 24

[51] Microsoft Corporation: *.NET*. http://www.microsoft.de/net. – [abgerufen am 24.07.2009] 37

[52] Miller, G. A. ; Fellbaum, C. ; Tengi, R. ; Wakefield, P. ; Langone, H. ; Haskell, B. R.: *WordNet*. http://wordnet.princeton.edu/. – [abgerufen am 24.07.2009] 13, 14, 27

[53] Pisan, Y. : Extending requirement specifications using analogy. In: *ICSE '00: Proceedings of the 22nd international conference on Software engineering*. New York, NY, USA : ACM, 2000. – ISBN 1–58113–206–9, S. 70–76 12

[54] Porter, A. ; Votta, L. : Comparing Detection Methods For Software Requirements Inspections: A Replication Using Professional Subjects. In: *IEEE Transactions on Software Engineering* 21 (1995), S. 563–575. – ISSN 0098–5589 9

[55] Power, N. : Variety and Quality in Requirements Documentation. In: *REFSQ'01: Proceedings of the Seventh International Workshop on Requirements Engineering : Foundation for Software Quality*, 2001. – ISSN 0163–5948, S. 165–170 3

[56] Python Software Foundation: *Python*. http://www.python.org. – [abgerufen am 24.07.2009] 38

[57] Robertson, S. ; Robertson, J. : *Mastering the requirements process*. 1. Auflage. Addison-Wesley, 1999. – ISBN 0–321–41949–9 20

[58] *Kapitel* Das SOPHIST-REgelwerk – Psychotherapie für Anforderungen. In: Rupp, C. : *Requirements-Engineering und -Management*. 4. Carl Hanser Verlag, 2007. – ISBN 3–446–40509–7, S. 139–176 1, 7, 15, 16, 17, 18, 19

[59] Rupp, C. ; Goetz, R. : Psychotherapy for System Requirements. In: *ICCI '03:*

Proceedings of the Second IEEE International Conference on Cognitive Informatics, 2003. – ISBN 0–7695–1986–5, S. 75–80 4

[60] Schwertel, U. : Controlling Plural Ambiguities in Attempto Controlled English (ACE). In: *Proceedings of the 3rd International Workshop on Controlled Language Applications*, 2000 10

[61] Smith, R. ; Avrunin, G. ; Clarke, L. ; Osterweil, L. : PROPEL: an approach supporting property elucidation. In: *ICSE 2002: Proceedings of the 24rd International Conference on Software Engineering*, 2002. – ISBN 1–58113–472–X, S. 11–21 10

[62] Software in the Public Interest, Inc.: *Debian GNU/Linux*. http://www.debian.org/. – [abgerufen am 24.07.2009] 38

[63] Steele, O. : *JWordNet*. http://sourceforge.net/projects/yawni. – [abgerufen am 24.07.2009] 38

[64] Suchanek, F. M. ; Kasneci, G. ; Weikum, G. : Yago: A Core of Semantic Knowledge. (2007). http://suchanek.name/work/publications/www2007.pdf. – [abgerufen am 24.07.2009] 14, 28

[65] Sun Microsystems, Inc.: *Java*. http://java.sun.com/. – [abgerufen am 24.07.2009] 37

[66] The Eclipse Foundation: *SWT: The Standard Widget Toolkit*. http://www.eclipse.org/swt/. – [abgerufen am 24.07.2009] 37

[67] The Stanford Natural Language Processing Group: *Stanford Log-linear Part-Of-Speech Tagger*. http://nlp.stanford.edu/software/tagger.shtml. – [abgerufen am 24.07.2009] 38

[68] UserLand Software, Inc.: *XML-RPC*. http://www.xmlrpc.com/. – [abgerufen am 24.07.2009] 38

[69] VMware, I. : *VMware ESX Server*. http://www.vmware.com/de/products/vi/esx/. – [abgerufen am 24.07.2009] 39

[70] Volere: *Requirements Tools*. http://www.volere.co.uk/tools.htm. Version: 2009. – [abgerufen am 24.07.2009] 4

[71] Wikimedia Foundation Inc.: *Wikipedia*. http://www.wikipedia.org. – [abgerufen am 24.07.2009] 14

[72] Wilson, W. ; Rosenberg, L. ; Hyatt, L. : Automated Analysis of Requirement Specifications. In: *ICSE '97: Proceedings of the 19th International Conference on Software Engineering*, 1997. – ISBN 0–89791–914–9, S. 161–171 11

[73] Ylonen, T. ; C. Lonvick, E. : *RFC 4251: The Secure Shell (SSH) Protocol Architecture*. http://tools.ietf.org/html/rfc4251. Version: Jan. 2006. – [abgerufen am 24.07.2009] 39