Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Untersuchen von Hintergrundwissen zur Verbesserung von semantischen Mappings

Bachelorarbeit

Marcel Jacob

Leipzig, den 24. September 2013

Betreuer:
Patrick Arnold
Prof. Dr. Erhard Rahm

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Arbeit unterstützt haben. Genannt seien dabei vor allem Herr Prof. Dr. Erhard Rahm und mein Betreuer M.Sc. Patrick Arnold, der mir bei der Auswahl des Themas, sowie bei der Bearbeitung immer zur Seite gestanden hat.

Inhaltsverzeichnis

1 Motivation

Seit jeher versuchen die Menschen Informationen zu speichern und für die Nachwelt in geeigneter Form festzuhalten. Da jedoch viele verschiedene Möglichkeiten existieren, um den gleichen Sachverhalt auszudrücken, hat jeder Mensch seine eigene Vorstellung wie diese Informationen repräsentiert werden sollten. In der Informatik haben sich Schemata etabliert, um die Struktur von Daten zu beschreiben. Verschiedene Beispiele für Schemata sind Datenbankschemata, XSD (XML Schema Definition) und Ontologien. Dabei kann die Struktur aufgrund der unterschiedlichen Herkunft und die Komplexität sehr unterschiedlich sein, je nach Domäne und dem verfolgten Ziel des Schemas. Dadurch entstehen viele verschiedene Schemata mit unterschiedlichem Aufbau. Dieses Problem wird als strukturelle Heterogenität bezeichnet. Es gilt dann die Schemata gegeneinander abzugleichen, um semantische Beziehungen zwischen diesen zu finden [9, 23].

Alle gefundenen Korrespondenzen ergeben ein Matching, welches im Weiteren beispielsweise genutzt werden kann, um die Schemata zu fusionieren. In erster Linie steht der Bereich der Datenintegration vor dieser Herausforderung. Dies beinhaltet unter anderem den Datenaustausch und die Schema Evolution. Ein manuelles Schema Mapping zu erstellen ist nicht nur aufwendig, sondern bei großen Schemata auch ineffizient. Es werden spezielle Domänenexperten benötigt, die diese Schemata potentiell schnell verstehen können. Dennoch verliert sich die beauftragte Person in solch einem Schema aufgrund tiefer Schachtelungen, vieler Tabellen und Attribute, die teilweise stark kryptisch durch Abkürzungen ausgedrückt werden [21]. In Folge dessen erkennt auch ein Experte nur einen Bruchteil der Korrespondenzen. Des Weiteren können gleich geschriebene Wörter mit unterschiedlicher Bedeutung (sogenannte Homonyme) zu fälschlich erkannten Korrespondenzen führen. Dies führt zu einem nicht zufriedenstellendem Ergebnis.

In Betracht dessen wurden automatische bzw. semi-automatische Ansätze erstellt, die die gesuchte Anfrage zur Datentransformation stellen und daraus resultierend, die Aufgabe des Menschen mindestens genauso gut lösen. Ein Problem, welches in diesem Zusammenhang auftritt, ist die Generierung der richtigen und effizienten Anfrage in Bezug auf die Schemata, so dass die transformierten Daten dem gesuchten Zielschema entsprechen [13]. Hierfür gibt es bereits zahlreiche Tools, die sich die linguistische und strukturelle Ähnlichkeit zwischen den Schemata zu Nutze machen, um die gewünschten Korrespondenzen zu erhalten. Beispiele für solche leistungsfähige Tools sind COMA 3.0 [8], welches an der Universität Leipzig entwickelt wurde und Clio [19], ein Projekt der Universität Toronto.

Bei einem Abgleich kann es vorkommen, dass syntaktisch verschiedene Wörter in den Schemata die gleiche Bedeutung haben und in Folge dessen diese Äquivalenz bei einem Matching erkannt werden sollte. Nicht weniger wichtig ist das Erkennen anderer Beziehungen wie Überbegriffe, Unterbegriffe und Teil-von Beziehungen, welche jedoch bislang von den meisten Tools nicht unterstützt werden.

Diese Bachelorarbeit soll eine Strategie testen, mit der es möglich ist, die Ergebnisse eines gegebenen Schema Mappings zu verbessern, indem die Defizite beim Erkennen von den genannten Beziehungen zumindest teilweise beseitigt werden. Damit dieses Ziel erreicht werden kann, sind verschiedene Schritte notwendig. Zunächst müssen die grundlegenden Begriffe definiert werden, die in unmittelbaren Zusammenhang mit der Vorbereitung, der praktischen Umsetzung und der Evaluation stehen und daher unerlässlich für das Verstehen der Gesamtproblematik sind. Diese Definitionen werden in Kapitel 2 behandelt. Es folgt in Kapitel 3 die Vorbereitung der Arbeit, welche eine Selektierung der Datenquellen vornimmt, um die Verbesserung des gegebenen Schema Mappings zu maximieren. In Kapitel 4 wird die Implementierung der ausgewählten Datenquellen beschrieben. Dies beinhaltet die Beschreibung der verschiedenen Ansätze zur Umsetzung und einen Test auf Effizienz durch Anfragen auf das fertige Programm. Die Evaluation in Kapitel 5 betrachtet die erzielten Ergebnisse für die Datenquellen auf verschiedenen Benchmarks. Die Zusammenfassung in Kapitel 6 bildet den Abschluss dieser Arbeit und gibt noch einmal einen kurzen Abriss über die einzelnen Kapitel.

2 Grundlagen

Der folgende Abschnitt gibt eine Einführung in grundlegende Begriffe, die im Rahmen der Bachelorarbeit benötigt werden, um die Problemstellung und dessen Lösung verstehen zu können.

2.1 Linguistische Grundlagen

Jede Sprache besteht aus unzähligen Substantiven, Verben, Adjektiven, Adverben und anderen Wortarten. Dabei stehen bestimmte Wörter untereinander in einem Zusammenhang, die in den Folgenden Abschitten genauer erläutert werden. Dabei liegt der Fokus auf den Substantiven. Für jede Beziehung zwischen den Wörtern wird ein Korrespondenztyp definiert, welcher in den weiteren Kapiteln als Repräsentant für die jeweilige Beziehung zwischen den Wörtern verwendet wird.

Synonyme

Synonyme sind syntaktisch verschiedene Wörter, aber mit semantisch gleicher Bedeutung. Für die deutsche Sprache bilden beispielsweise Person und Individuum Synonyme. Sind zwei Wörter Synonyme, so wird als Korrespondenztyp „equal“ angenommen. Alle Synonyme, die zu einem semantischen Konzept/Wort gehören, sind in der gleichen Synonymgruppe (engl. „synset“).

Für Synonyme kann man nach [17] weitere Unterscheidungen treffen:
Vollsynonyme stellen Wörter dar, die in einem Satz ersetzt werden können, ohne dessen Bedeutung oder Aussagekraft zu verändern. Hierzu zählen in erster Linie verkürzende Schreibweisen für einen Begriff oder beispielsweise die lateinische oder fachspezifische Bezeichnung gegenüber der Umgangssprachlichen für einen Begriff. Beispielsweise sind Kochsalz und Natriumchlorid oder Appendizitis und Blinddarmentzündung Vollsynonyme.

Wortpaare stellen Quasisynonyme dar, wenn sie den gleichen Bezug aufweisen, aber dennoch anders gedeutet werden können. PKW und Auto können hier als Quasisynonyme aufgefasst werden.

Teilsynonyme lockern die Bedingung der Übereinstimmung weiter auf:
Hier müssen nur noch wesentliche Teilbereiche übereinstimmen, die semantischen Relationen können unterschiedlich sein. Wenn wir von einem Ei sprechen, meinen wir intuitiv aus Erfahrung ein Hühnerei, auch wenn es noch andere Arten von Eiern gibt.

Homonyme

Ein Wort ist ein Homonym, wenn es bei syntaktisch gleicher Schreibweise mehrere Bedeutungen aufweist. Das klassische Beispiel für ein Homonym ist die Bank. Zum einen ist es möglich, dass die Sitzgelegenheit gemeint ist, zum anderen das Finanzinstitut. Auch das zunächst unscheinbare Wort Rentier stellt bei verschiedener Betonung ein Homonym dar. Je nach Interpretation und Themenbezug ist entweder das Tier oder der Rentner gemeint. In einem längeren Dokument kann die Bedeutung meist über den Kontext erschlossen werden, da das zugehörige Thema abgegrenzt werden kann. Jedoch stellen Homonyme ohne weitere Informationen ein Problem dar, wie dies in vielen Schemata der Fall ist. Es wird dann davon ausgegangen, dass diese Wörter eine hohe Wahrscheinlichkeit aufweisen zusammenzugehören und der Korrespondenztyp „ident“ vorgeschlagen.

Antonyme

Zwei verschiedene Wörter sind Antonyme, wenn sie gegensätzliche Bedeutungen aufweisen. Berg und Tal sind eindeutige Antonyme, wohingegen Hund und Katze nur im weiteren Sinne Antonyme bilden. Eine Korrespondenz soll bei solchen Beispielen nicht gefunden werden, ein „mismatch“ ist die Folge.

Hyper- und Hyponyme

Ein Wort w_1 ist ein Hypernym, wenn es ein allgemeineres Konzept als das Wort w_2 beschreibt. Die vorliegende Korrespondenz w_1 zu w_2 ist „inverse is-a“. Das Wort w_2 ist dann das Hyponym von w_1 und „is-a“ wird als Korrespondenztyp gesetzt. Verallgemeinerungen oder Spezifizierungen können über mehrere Stufen gehen, es entstehen bei n Stufen Hyper- und Hyponyme n-ten Grades. Beispielsweise ist ein Apfel eine Frucht. Der Apfel ist dann das Hyponym in Bezug auf die Frucht, welche wiederrum das Hypernym für Apfel darstellt.

Zu vielen Konzepten können Instanzen erstellt werden, die eine praktische Nähe aufweisen. Hier gilt dann auch die „is-a“-Beziehung für die erzeugte Instanz in Bezug auf das Konzept. Beispielsweise ist „Leipzig“ eine Instanz des Konzeptes „Stadt“ und es gilt: Leipzig „is-a“ Stadt.

Holo- und Meronyme

Holonyme und Meronyme stellen ebenfalls gegensätzliche Beziehungen dar. Beinhaltet das Wort w_1 bzw. das Objekt, welches durch das Wort repräsentiert wird, typischerweise das Wort w_2, so ist w_1 das Holonym von w_2. Liegt eine solche Situation vor, ist der Korrespondenztyp „has-a“. W_2 wird dann das Meronym von w_1 genannt, da w_2 ein Bestandteil von w_1 ist. Tritt dieser Fall ein, so wird „part-of“ als Korrepondenz festgelegt. Der menschliche Körper besteht aus vielen Einzelteilen, welche allesamt Teil des Körpers sind. Er ist in diesem Beispiel das Holonym, alle Körperteile würden Meronyme darstellen. Außerdem ist es möglich, den Studenten, der zwar kein direkter Bestandteil, aber dafür Mitglied der Universität ist, als Meronym zu Universität aufzufassen. Für den Student gilt dann eine „member-of“-Beziehung.

Assoziationen

Des Weiteren gibt es Wörter, die in keiner direkten Beziehungen zueinander stehen, jedoch trotzdem voneinander abhängig sind, da sie im gleichen Kontext erwähnt werden. Die Begriffe sind dann „related“ zueinander, da eine Assoziation zwischen ihnen besteht. Beispielweise führt für die beiden Wörter Sonnenmilch und Sonnenbrand keine der sonst genannten Beziehungen zu einem Ergebnis, dennoch treten beide Wörter oftmals im gleichen Kontext auf. Im konkreten Beispiel kann aus der Erfahrung ein Zusammenhang hergeleitet werden: Sonnenmilch verhindert Sonnenbrand.

Die nachfolgende Tabelle 1 fasst alle Relationstypen noch einmal mit ihren Beispielen zusammen.

Relation	**Beispiel**	**Korrespondenztyp**
Synonym	Person, Individuum	„equal“
Homonym	Bank, Bank	„ident“(gleiche Syntax)
Antonym	Berg, Tal	„mismatch“
Hypernym	Frucht, Apfel	„inverse is-a“
Hyponym	Apfel, Frucht	„is-a“
	Leipzig, Stadt	„is-a“
Holonym	Körper, Bein	„has-a“
	Universität, Student	„has-a“
Meronym	Bein, Körper	„part-of“
	Student, Universität	„member-of“
Assoziation	Sonnenmilch, Sonnenbrand	„related“

Tabelle 1: Übersicht über die linguistischen Relationstypen

2.2 Datenintegration

Der Bereich der Datenintegration, auch Informationsintegration, beschäftigt sich mit der Zusammenführung bestehender Daten von heterogenen Datenquellen zu einer einheitlichen und strukturierten Informationsmenge. Die Herausforderung bei der Integration der Daten ist es, diesen Prozess korrekt, vollständig und effizient umzusetzen. Durch die Kombination der Daten werden aussagekräftigere Ergebnisse erzielt, da mehrere Datenquellen verwendet werden. Außerdem ist es möglich, dass Anfragen beantwortet werden können, die eine Datenquelle allein nicht beantworten kann.

Die Datenintegration verschiedener Quellen setzt sich aus drei Hauptaufgaben zusammen: dem Data Matching, dem Schema Matching und der Datenfusion. Diese werden in den folgenden Abschnitten näher erläutert [2].

2.2.1 Data Matching

Im Bereich des Data Matchings liegt der Fokus beim Identifizieren, Matchen und Fusionieren von semantisch gleichen Entitäten zwischen verschiedenen Datenbanken. Dabei werden individuelle Datensätze gegeneinander gematcht, um jene zu erkennen, die sich auf das gleiche reale Objekt beziehen. Das Erkennen von Duplikaten innerhalb einer Datenbank wird unter anderem als Entity Resolution bezeichnet und ist ein Spezialfall des Data Matchings.

2.2.2 Schema Matching

Die zweite Aufgabe bei der Datenintegration ist das Schema Matching, welches in dieser Arbeit besondere Aufmerksamkeit erhält. Diese Problematik beschäftigt sich mit dem Identifizieren von Strukturen von Schemata, Tabellen und Attributen zwischen verschiedenen Datenbanken und sucht nach Daten mit gleichen oder ähnlichen semantischen Informationen.

Für das Schema Matching gibt es verschiedene Ansätze: zum einen auf Schemaebene und zum anderen auf Instanzebene. Im Folgenden wird eine Unterteilung in die verschiedenen Herangehensweisen nach [21, 6] vorgenommen.

Schemabasiert - Namen der Schemaelemente

Für den schemabasierten Ansatz auf Elementebene wird zunächst das Kreuzprodukt aus den Attributen der beiden vorhandenen Schemata gebildet. Anschließend wird für jedes Attributpaar eine Ähnlichkeit mit Hilfe einer Ähnlichkeitsfunktion berechnet. Dies kann zum Beispiel mit der Editierdistanz oder einem Test auf Gleichheit nach Durchführen des Stemmings, also dem Zurückführen auf den Wortstamm, geschehen. Nun wird ein Schwellwert bzw. eine Grenze festgelegt und mit dem berechneten Wert verglichen. Ist das durch die Ähnlichkeitsfunktion erhaltene Ergebnis höher als dieser Grenzwert, so wurde ein Match gefunden.

Die Zeitkomplexität kann hier bei großen Schemata ein Problem darstellen. Bei einer Anzahl von n bzw. m Attributen ergibt sich eine Zeitkomplexität von $O(n*m)$. Dennoch können nicht alle Korrespondenzen gefunden werden, wenn die Attribute Synonyme sind, sich jedoch syntaktisch vollkommen unterscheiden. Homonyme können gar nicht unterschieden werden. Sind zwei Attribute syntaktisch identisch, stellen sie für den Matcher das gleiche Wort dar. In Folge dessen kann es passieren, dass falsche Korrespondenztypen gefunden werden.

Schemabasiert - Struktur der Schemaelemente

Bei dieser Variante wird mit Hilfe der Struktur der Schemaelemente das Schema Matching durchgeführt. Hierzu werden unter anderem die Hierarchieebene, der Elementtyp und Nachbarschaftsbeziehungen genutzt, um neue Matches zu finden.

Instanzbasiert

Hierbei werden interessante Eigenschaften der Daten für jedes Attribut aus den Attributmengen der beiden Schemata extrahiert. Anschließend wird das Kreuzprodukt aller Attribute aus beiden Schemata gebildet und die Ähnlichkeit bezüglich der Eigenschaften für jedes Paar verglichen. Ein naheliegendes Problem ist die Frage nach den interessanten Eigenschaften und welche Gewichtung für diese vergeben werden.

Außerdem gibt es kombinierte Matcher, die verschiedene Herangehensweisen vereinen, um deren Vorteile zu kombinieren.

2.2.3 Datenfusion

Der dritte Teilbereich der Datenintegration behandelt die Fusion von Daten. Dabei werden Datensätze, die als Duplikate durch das Matching unter Zuhilfenahme von Ähnlichkeitsmaßen erkannt wurden, zu einem konsistenten Datensatz zusammengeführt. Dieser Prozess wird nicht aufgrund des Mehraufwands an Speicher durchgeführt, sondern hat ein wirtschaftliches Anliegen. Werden beispielsweise in einem Kundenmanagementsystem Duplikate nicht erkannt, steigen die Material- und Portokosten an. Möchte ein Kunde, der im System doppelt geführt wird, sein Konto nach einer Transaktion abfragen, kann dies zu Widersprüchen bzw. Inkonsistenzen und in Folge dessen zu Beschwerden führen. Außerdem werden Statistiken über die Kundenanzahl oder den Umsatz eines Kunden verfälscht. Diese negativen Effekte müssen soweit wie möglich vermieden werden.

2.3 Schema Mapping

Um das Schema Mapping durchführen zu können, wird das zuvor ermittelte Schema Matching-Ergebnis benötigt, welches als Eingabe für das Schema Mapping dient. Das Ziel beim Schema Mapping ist es, entsprechende Anfragen zu finden, die Daten eines Schemas in Daten eines anderen Schematas überführt. Dazu wird eine Transformationsanfrage abgeleitet, die Daten des Quellschemas in Daten des Zielschemas umformt. Hierbei sollen möglichst alle gefundenden Korrespondenzen beachtet werden, die Semantik erhalten bleiben und eventuelle Constraints nicht verletzt werden.

Eine für diese Aufgabe geeignete Anfragesprache ist SchemaSQL, eine Erweiterung von SQL, mit der Anfragen über mehrere Datenbanken realisiert werden. Dabei ist es möglich, sowohl auf Daten als auch auf Metadaten zuzugreifen [21].

2.4 Verfeinern des Schema Mappings

Der folgende Abschnitt bezieht sich auf [4] und stellt verschiedene Möglichkeiten vor, die nach einem Schema Mapping eingesetzt werden können, um eventuell entstandene Fehler auszuschließen und dadurch die Qualität des Schema Mappings zu verbessern.

Background Knowledge

Das Ausnutzen von Hintergrundwissen stellt ein mächtiges Werkzeug für das Bestimmen von Beziehungen zwischen Wörtern dar. In solch einem semantischen Wortnetz können die verschiedenen linguistischen Beziehungen, die in Abschnitt 2.1 vorgestellt wurden, gespeichert werden. Eine Auswahl der wichtigsten Datenquellen folgt in Abschnitt 3.1. Durch das Einbringen von zusätzlichem Wissen soll die semantische Heterogenität zwischen den Attributen der Schemata zumindest teilweise überwunden werden.

Compound-Strategie

Ein Compound ist ein Wort w, welches aus mehr als einem Wortstamm besteht. Dies bedeutet, dass ein Compound in mindestens zwei unabhängige Wörter zerlegt werden kann. Dieses Wort w wird in den „Kopf“, w_H der am Ende des Wortes steht und den „Modifier“ w_M unterteilt. Der Modifier spezifiziert den Kopf.
Diese Strategie überprüft, ob ein Wort w_1 ein anderes Wort w_2 enthält. W_1 muss mindestens drei Buchstaben länger sein als w_2, um falschen Compounds, den Pseudo-Compounds, vorzubeugen. Oftmals ist dann das Wort w_1 spezieller als w_2 und eine „is-a“ bzw. „inverse is-a“ Beziehung kann zwischen den Wörtern festgelegt werden. Beispielweise enthält das Wort Apfelbaum das Wort Baum, daher ist der Apfelbaum ein Baum. Jedoch funktioniert die Methode nicht immer, denn es kann vorkommen, dass gleiche w_H nur ein Hinweis auf einen ähnlichen Aufbau des Objektes sind. Auch wenn ein Sägezahn im weitesten Sinne einem Zahn ähnelt, so trifft hier nicht die „is-a“ Beziehung zwischen beiden zu, da sie aus unterschiedlichen Domänen stammen.

Itemization

Diese Strategie kommt zum Einsatz, wenn mindestens eines der beiden Konzepte mehrere Wörter enthält, die durch „und“ oder „oder“ miteinander verbunden sind. Hierbei kommt ein Algorithmus zum Einsatz, der zuerst Synonyme und Hyponyme in jeder Menge entfernt. Anschließend werden Synonyme zwischen den Mengen entfernt und zuletzt alle Hyponyme in Menge 1 entfernt, wenn es ein Hypernym in Menge 2 gibt. Nun wird eine Unterscheidung der Mengen getroffen:

1. Sind beide Mengen leer, sind die Ausgangsmengen identisch zueinander. Es folgt der Korrespondenztyp „equal“.

2. Ist Menge 1 leer, Menge 2 enthält jedoch mindestens noch ein Element, liegt eine „is-a“ -Beziehung vor.

3. Enthält Menge 1 mindestens ein Element, Menge 2 ist jedoch die leere Menge, wird „inverse is-a“ als Korrespondenz vergeben.

4. Ist keine der beiden Mengen leer, kann keine Entscheidung getroffen werden.

Structure

Diese Strategie macht sich die interne Struktur der Schemata zu Nutze, um neue Korrespondenzen durch logische Schlussfolgerungen ableiten zu können. Besteht eine „is-a“ Relation zwischen Y und X, können mit zusätzlichen Bedingungen folgende Schlüsse, wie in Abbildung 1 illustriert, gezogen werden:

1. Sind X und Z Synonyme, besteht eine „is-a“ Beziehung zwischen Y und Z.

2. Besteht eine „is-a“ Beziehung zwischen X und Z, kann eine „is-a“ Beziehung zwischen Y und Z hergeleitet werden.

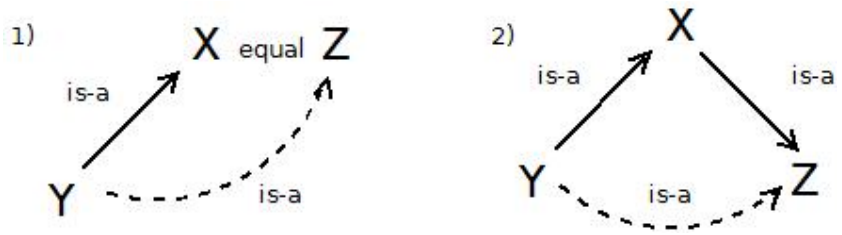

Abbildung 1: Regeln für die Structure-Strategie

Diese Regeln können analog für „part-of“ Beziehungen angewandt werden.

Alle vorgestellten Verfahren zur nachträglichen Verbesserung des Schema Mapping haben ihre Vor- und Nachteile. Die Tabelle 2 zeigt zusammenfassend, welche Korrespondenzen von den jeweiligen Strategien erkannt werden können.

Strategie	„equal“	„is-a“	„part-of“	„related“
Background Knowledge	Ja	Ja	Ja	Ja
Compound	Nein	Ja	Nein	Nein
Itemization	Ja	Ja	Nein	Nein
Structure	Nein	Ja	Ja	Nein

Tabelle 2: Strategien zur Verbesserung des Schema Matchings

Da eine Datenquelle mit Hintergrundwissen alle Korrespondenzen beinhalten kann, eignet sich hier eine Überprüfung am besten. Hierzu werden verschiedene Wissensspeicher in Kapitel 3.1 betrachtet.

2.5 Effektivitätsmaße

Um die Ergebnisse der Implementierung auswerten zu können, werden im Folgenden die drei verschiedenen Maße Precision, Recall und F-Measure, die sich im Bereich der Evaluation etabliert haben, in Bezug auf das Schema Mapping eingeführt.

Ausgangspunkt ist ein gegebener Benchmark mit verschiedenen Korrespondenzen zwischen den Attributen zweier Schemata. Dieser ist in der Regel manuell erstellt, um Fehler so gering wie möglich zu halten und wird als Goldstandard bezeichnet. Wird dieser nun mit dem implementierten Verfahren verglichen, ergeben sich vier Möglichkeiten, die in Tabelle 3 mit ihren Abkürzungen dargestellt sind. Diese werden im Folgenden für die Formeln für Precision, Recall und F-Measure verwendet.

		Verfahren	
		Ja	Nein
Benchmark	Ja	#VerBen	#Ben
	Nein	#Ver	nicht relevant

Tabelle 3: Mögliche Ergebnisse beim Vergleich von Verfahren mit dem Benchmark

Precision

Die Precision p ist ein Maß, welches die Zielgenauigkeit des Verfahrens beschreibt. Dies meint den Anteil richtig gefundener Korrespondenzen gemessen an allen durch das Verfahren gefundenen Korrespondenzen. Sie lässt sich wie folgt berechnen:

$$p = \frac{\#VerBen}{\#VerBen + \#Ver}$$

Recall

Der Recall r beschreibt den Erschöpfungsgrad des Verfahrens, das heißt er ist derjenige Anteil, der die richtig gefundenen Korrespondenzen ins Verhältnis zu allen Korrespondenzen aus dem Benchmark setzt. Folgende Formel kann zur Berechnung des Recalls verwendet werden:

$$r = \frac{\#VerBen}{\#VerBen + \#Ben}$$

F-Measure

Eine Kombination aus den beiden Maßen Precision und Recall ergibt das F-Measure f. Hierzu wird das harmonische Mittel wie folgt berechnet:

$$f = \frac{2 * p * r}{p + r}$$

Alle beschriebenen Maße liegen im Bereich von 0 bis 1, wobei der Wert von 1, also 100% den Zielwert des Verfahrens meint, der jedoch so gut wie nie erreicht wird. Dies liegt unter anderem an der Tatsache, dass sich Precision und Recall gegenseitig beeinflussen. Die große Herausforderung ist es, alle richtigen Korrespondenzen zu finden, ohne dabei falsche Korrespondenzen zu erkennen.

Der in dieser Bachelorarbeit verfolgte Ansatz der Nutzung von Hintergrundwissen konzentriert sich vorrangig auf den Recall, da dieser bei aktuellen Ergebnissen, wie in [4] nachzulesen, noch sehr gering ist.

3 Vorbereitung

Es gibt bereits unzählige Projekte, die sich das ehrgeizige Ziel gesetzt haben, ein vollständiges Wörterbuch mit ihren Beziehungen zu erstellen. Diese wurden für die verschiedensten Zwecke konzipiert und sind in der Regel online verfügbar. Dabei können sie sich in diversen Merkmalen unterscheiden und die folgenden Fragenstellungen sind zu betrachten:

1. Welche Domäne wird behandelt?
2. Welchen Umfang hat die Datenquelle?
3. In welcher Sprache oder welchen Sprachen ist die Datenquelle verfügbar?
4. Welche Korrespondenzen werden gespeichert?

Eine Datenquelle sollte mindestens Synonyme, Hypernyme und Hyponyme speichern, damit sie sinnvoll für das Verbessern des Schema Mappings genutzt werden kann.

Eine kleine Auswahl bekannter und bereits etablierter Projekte wird im Folgenden näher betrachtet, deren Vor- und Nachteile diskutiert und zwei dieser Datenquellen für die praktische Umsetzung ausgewählt.

3.1 Auswahl der Datenquelle

Die nachfolgende Tabelle 4 enthält eine Auflistung verschiedener Ansätze für Wörterbücher mit ihren wichtigsten Merkmalen, welche im Weiteren besprochen und gegeneinander abgewägt werden. Dabei beziehen sich die Angaben der Tabelle auf die jeweils aktuellen Versionen (Stand September 2013). Die eingetragenen Beziehungen in der Spalte Verknüpfungen kann unvollständig sein, dennoch ist es möglich einen groben Überblick zu erhalten.

Viele semantische Netze erklären die verschiedenen Bedeutungen von Wörtern und stellen einen Bezug auf eine Relation zwischen den Wörtern her ohne diese Beziehung jedoch genauer zu spezifizieren. Datenquellen, die nur solche Relationen aufweisen, werden in Tabelle 4 nicht berücksichtigt.

Datenquelle	Kosten	Domäne	Umfang	Sprache	Korrespondenz
Wordnet	Nein	allgemein	ca. 117000 Synsets	Englisch	„equal“, „is-a“, „inverse is-a“, „part-of“, „related“
Germanet	Ja/Nein	allgemein	ca. 85000 Synsets	Deutsch	„equal“, „is-a“, „inverse is-a“, „part-of“, „related“
EuroWordnet	Ja	allgemein	k.A.	verschiedene	„equal“, „is-a“, „inverse is-a“, „part-of“, „related“
OpenThesaurus	Nein	allgemein	ca. 25000 Synsets	Deutsch	„equal“, „mismatch“, „is-a“, „inverse is -a“, „related“
UNESCO Thesaurus	Ja	allgemein	ca. 31200 Konzepte	verschiedene	„equal“, „is-a“, „inverse is-a“, „related“
Gene Ontology	Nein	Biologie	ca. 62000 Konzepte	Englisch	„equal“, „is-a“, „inverse is-a“
Unified Verb Index	Nein	allgemein	ca. 8500 Verben	Englisch	„equal“
BabelNet	Nein	allgemein	ca. 5,5 Mio. Konzepte	verschiedene	„is-a“, „inverse is-a“, „part-of“, „related“
UMLS Metathesaurus	Nein	Biomedizin	ca. 157000 Konzepte	verschiedene	„equal“, „is-a“, „inverse is-a“
SUMO	Nein	allgemein	ca. 28000 Konzepte	Englisch	„equal“, „is-a“, „inverse is-a“
Moby Thesaurus	Nein	allgemein	ca. 30000 Konzepte	Englisch	„equal“, „related“

Tabelle 4: Auflistung einiger Datenquellen mit ihren Merkmalen

Da es bereits eine Validierung mit WordNet [16] in [4] gegeben hat, wird eine andere Datenquelle als Grundlage genommen. Vielversprechend wäre der WordNet-Ansatz für die deutsche Sprache, das GermaNet [18], welches jedoch nur erhältlich ist, wenn ein spezieller Antrag für das Erhalten einer Lizenz gestellt wurde. Andere Ansätze wie das EuroWordnet [1], welches für mehrere Sprachen konzipiert wurde, oder der UNESCO Thesaurus [20] sind generell kostenpflichtig und scheiden daher aus. Domänenspezifische Ansätze, wie die Gene Ontology [5] oder der UMLS Metathesarus, der Teil des Unified Medical Language System (UMLS) [10] ist, erfordern entweder Fachwissen aus dem jeweiligen Fachbereich oder einen hohen Rechercheaufwand, um die Korrektheit beim Matchen durch einen Benchmark überprüfen zu können. Die erwartete Verbesserung für Schema Mappings aus dieser Domäne wären aber entsprechend hoch.

Bei der Spalte Umfang ist zwischen Konzepten und Synsets zu unterscheiden, weshalb ein Vergleich hier nur eingeschränkt möglich ist. Die Anzahl der Konzepte einer Quelle ist immer größer gleich der Anzahl der Synsets. Eine ungefähre Abschätzung über die Größe der Datenquelle ist anhand der verschiedenen Zahlenwerte dennoch möglich. Datenquellen, die ebenfalls die allgemeine Domäne darstellen, in Englisch gespeichert sind, aber einen geringeren Umfang als WordNet aufweisen, entfallen als mögliche Datenquellen, da diese keine oder nur wenige bislang unbekannte Verknüpfungen von Wortpaaren besitzen werden. Dies betrifft SUMO [15] und den MobyThesaurus [22].

Die Erstellung eines Wörterbuchs, einer Ontologie oder einer Taxonomie erfolgt meist manuell, da eine Sprache als solches sehr komplex aufgebaut ist und es daher schwierig ist ohne Hintergrundwissen ein zufriedenstellendes Ergebnis zu erhalten. Eine Ausnahme in der Tabelle bildet BabelNet [14], welches automatisch erzeugt wurde. BabelNet birgt daher Gefahr falsche Ergebnisse zu liefern, welche auf Kosten der Precision gehen würden.

Der Unified Verb Index [3], ein Projekt, welches von der Universität von Colorado geleitet wird, fährt hingegen einen anderen Ansatz. Hierbei hat man sich nur auf Verben festgelegt, welche jedoch einen Nachteil mit sich bringen: im Allgemeinen basieren Schemata nur auf Substantiven, weshalb sich dieser Ansatz für das Matchen zwischen Schemata bzw. dem nachträglichen Verbessern eines Mappings nicht gut eignet. Zudem lassen sich Verben nicht zielgerichtet zum Beispiel in Über- und Unterbegriffe einteilen.

Alle vorhandenen, manuell erstellten Datenquellen verfolgen das Ziel, bei einem Einsatz im Schema Matching- und Schema Mapping-Bereich den Recall zu erhöhen. Ebenso können potentiell falsche Matches erkannt werden und die Precision geringfügig anheben. Ist mehr Wissen vorhanden, wird auch mit hoher Wahrscheinlichkeit mehr erkannt, welches eine positive Auswirkung auf den Recall hat. Falsche Einträge bleiben bei manuell erstellten Lexika relativ gesehen konstant.

Eine Auflistung weiterer Wörterbücher für spezielle Domänen und vieler weiterer Sprachen kann im Thesaurusportal [7] näher betrachtet werden. Nur wenige Wortnetze enthalten Korrespondenzen, die über Synonyme und „is-a“ Beziehungen hinausgehen. Jedoch sollten diese Beziehungen bereits ausreichen, um das Ergebnis, gemessen in Recall und Precision, des Schema Mappings zu verbessern.

Letztendlich kristallisiert sich als Erstes der OpenThesaurus [12] als frei verfügbarer Thesaurus für die deutsche Sprache heraus, da er viele Konzepte und Beziehungen zwischen diesen beinhaltet und als erste deutsche Datenquelle im EnrichmentCenter ein breites Spektrum der deutschen Sprache abdeckt. Es ist zu erwarten, dass Benchmarks, welche deutsche Korrespondenzen enthalten, erheblich verbessert werden, da bislang keine Strategie die deutsche Sprache ausreichend behandelt. Eine zweite Datenquelle, die alle Anforderungen erfüllt ist das Unified Medical Language System (UMLS). Da dieses System aus mehreren Komponenten und vielen einzelnen Datenquellen besteht, wird eine spezielle Datenquelle ausgewählt, der UMLS-Metathesaurus. Die letzte Datenquelle, die alle Anforderungen erfüllt ist die Gene Ontology, die jedoch auf Grund ihres geringeren Umfang im Vergleich mit dem UMLS-Metathesaurus ausscheidet.

OpenThesaurus und der UMLS-Metathesaurus bilden somit eine gute Grundlage für die Bachelorarbeit. Wie stark diese beiden Datenquellen Precision und Recall steigern können, soll im Rahmen dieser Bachelorarbeit herausgefunden werden.

3.2 Allgemeines zum OpenThesaurus

Der OpenThesaurus [12] ist ein 2002 gegründetes Projekt, welches Synonymgruppen oder auch Synsets genannt, Hypernyme, Hyponyme und Assoziationen zwischen Wörtern erfasst und somit ein Wortnetz der deutschen Sprache bildet. Durch die aktive Mitarbeit der Community an diesem OpenSource Projekt erscheinen täglich neue Versionen mit Änderungen am Vokabular oder den Korrespondenzen. Dabei ist der komplette Datensatz kostenlos zum Download erhältlich und eine eigenständige Änderung der Daten möglich und auch gewünscht, um eine Verbreitung zu gewährleisten. Die derzeitige Version des OpenThesaurus enthält ungefähr 28000 Synonymgruppen und 104000 verschiedene Einzelwörter (Stand 17. September 2013).

3.2.1 Struktur

Die Version des OpenThesaurus vom 17. September 2013 enthält 21 Tabellen, hiervon sind jedoch viele nicht aussagekräftig, da sie keine Datensätze enthalten. Einige Tabellen haben Informationen gespeichert, die jedoch für das Verfeinern des Schema Mappings nicht relevant sind. Beispielsweise ist es für das Mapping egal, welcher Domäne die einzelnen Wörter angehören. Alle diese nicht relevanten Tabellen werden in dieser Bachelorarbeit nicht weiter betrachtet und bei der Implementierung entfernt, um die Übersichtlichkeit beizubehalten. Die Datenbank enthält die vier Tabellen *term*, *term_link*, *link_type* und *synset_link*, welche alle notwendigen Informationen für das Testen des

OpenThesaurus-Verfahrens enthalten. Dabei enthält die Tabelle term alle notwendigen Informationen über ein Wort, zum Beispiel welcher Synonymgruppe es angehört (*synset_id*). Die Relation *term_link* enthält Antonyme, die in *term_id* und *target_term_id* hinterlegt sind.

Der OpenThesaurus enthält fast ausschließlich Wörter in ihrer Grundform. Insgesamt enthält der OpenThesaurus nicht nur Substantive, die meist für das Schema Matching benötigt werden, sondern auch andere Wortarten wie Verben, Adjektive, Adverben und auch Phrasen sind hier gespeichert. Diese zusätzlichen Informationen beeinträchtigen jedoch nicht das Ergebnis, sondern wirken sich lediglich geringfügig auf die Performanz der Anfragen aus.
Die Klassifikation der Begriffe von dem allgemeinsten Begriff zu spezielleren Begriffen in OpenThesaurus ist in Abbildung 2 angedeutet:

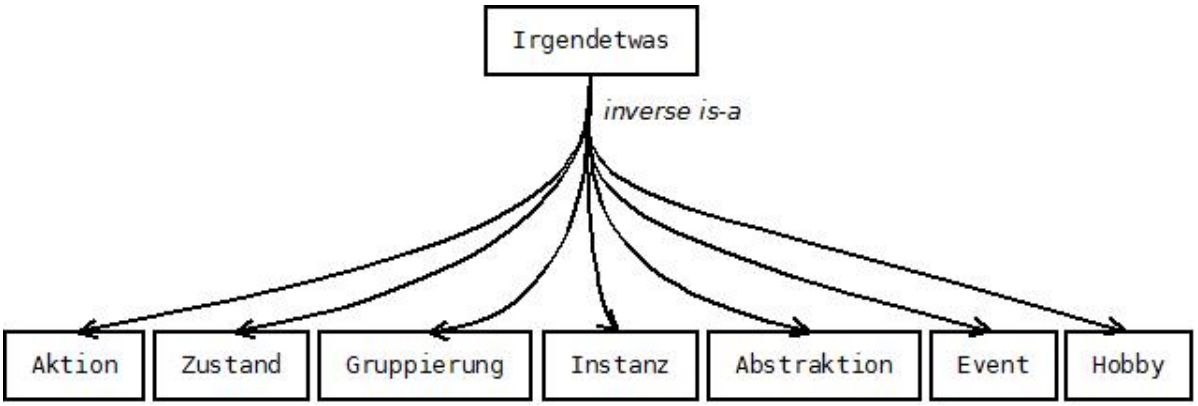

Abbildung 2: Angedeuteter Hierarchiebaum des OpenThesaurus - Wurzel und erste Ebene

Hierbei wurden die *synset_ids* in Wörter aufgelöst, jedoch nur das aussagekräftigste Wort für jede *synset_id* als Repräsentant ausgewählt, um die Übersichtlichkeit zu wahren.

3.2.2 Einsatzgebiete

Der OpenThesaurus kann unter anderem in der Textverarbeitung eingesetzt werden, um die Qualität von Texten durch das Finden von Synonymen zu einzelnen Wörtern zu erhöhen. Im Bereich des Information Retrieval werden Thesauri allgemein genutzt, um eine Suchanfrage mit gefundenen Synonymen gegebenenfalls zu verändern [11]. Beim Schema Matching werden Thesauri benötigt, um nach Möglichkeit alle vorhandenen Korrespondenzen zwischen Schemata ausfindig zu machen. Ist bereits ein Schema Mapping gegeben, besteht die Aufgabe darin, falsche Korrespondenzen ausfindig zu machen und die Fehler zu beheben.

3.3 Allgemeines zu UMLS

Die nachfolgenden Informationen zu diesem Kapitel entstammen größtenteils aus [10]. Das Unified Medical Language System (UMLS) ist ein bereits Ende der 80er Jahre gegründetes Projekt, welches Konzepte und Relationen verschiedener Datenquellen aus der Medizin und Biomedizin miteinander in Beziehung setzt und zusammenführt. Dabei kommen Software-Tools zum Einsatz, die beispielsweise den Metathesaurus, den Hauptbestandteil des Projektes, modifizieren können oder aus linguistischer Sicht Hilfestellung leisten, um die Arbeit mit den Daten zu erleichtern. Des Weiteren bietet UMLS Möglichkeiten für das Natural Language Processing (NLP), also das Textverstehen mit Hilfe des Computers, an.

3.3.1 Komponenten

Dabei besteht UMLS aus drei Hauptkomponenten: dem Metathesaurus, dem semantischen Netzwerk (Semantic Network) und dem SPECIALIST Lexikon, die im Folgenden näher betrachtet werden sollen.

Der komplette Metathesaurus ist ein mehrere Millionen biomedizinische und gesundsheitsbezogene Konzepte enthaltender, mehrsprachiger Thesaurus. Dieser beinhaltet Synonyme und Beziehungen zwischen den einzelnen Konzepten, wie Hypernyme-, Hyponyme und Assoziationen, und stellt somit die Datengrundlage für die Strategie mit UMLS dar. Für die Verbesserung des Schema Mappings ist es jedoch notwendig, eine entsprechende Auswahl aus den über 150 Wortschätzen zu treffen, um die Datenmenge einzugrenzen und damit die Übersicht zu behalten. Außerdem kann nur dadurch eine schnelle Performance gewährleistet werden. Zu den 150 vorhandenen Datenquellen gehören unter anderem: Medical Subject Headings, UMLS-Metathesaurus, LOINC, die NCBI Taxonomy und viele mehr. Die Auswahl fällt hier auf den „UMLS-Metathesaurus“, da dieser eine große Anzahl von Konzepten (157571) aufweist und die Annahme nahelegt eine hohe Relevanz für den Metathesaurus von UMLS, aufgrund seines Namens zu besitzen.

Das semantische Netzwerk besteht zum einen aus den verschiedenen Kategorien, die in ihrer Gesamtheit alle Konzepte abdecken und zum anderen aus Beziehungen, die zwischen den Kategorien bestehen.

Das SPECIALIST Lexikon stellt die Informationen bereit, die nötig sind, um Natural Language Processing durchführen zu können. Es beinhaltet häufig vorkommende Begriffe, sowie Vokabular aus der Biomedizin.

Struktur UMLS-Metathesaurus

In dem Original-Datenbankschema des UMLS-Metathesaurus gibt es 48 Tabellen, von denen viele keine Datensätze enthalten, weil nur der UMLS-Metathesaurus der UMLS-Komponente Metathesaurus in die Datenbank eingefügt wurde. Andere Tabellen enthalten eine Vielzahl von Informationen über gelöschte oder zusammengefügte Kozepte, welche keine Bedeutung für die Verbesserung des Schema Mappings besitzen. Letztendlich beinhalten nur die beiden Tabellen *mrconso* und *mrrel* relevante Informationen. Dabei speichert die Tabelle *mrconso* unter anderem eine spezielle Konzept-Id und den eigentlichen Konzeptnamen. Bei der Tabelle *mrrel* können die gespeicherten Relationen nachvollzogen werden. Diese haben die Form Relations-Id, Konzept-Id1, Relation, Konzept-Id2. In der Implementierung werden nur die notwendigen Tabellen, Spalten und Datensätze behalten. Ein Beispiel einer gespeicherten Hierarchiekette der Relationen von allgemein zu speziell ist die folgende:

Disease „inverse is-a“ Gastrointestinal Diseases,
Gastrointestinal Diseases „inverse is-a“ Ileus,
Ileus „inverse is-a“ Paralytic Ileus.

Dies unterstreicht nochmal die medizinische Domäne des UMLS-Metathesaurus anhand sehr spezifischer Krankheiten.

Eine Klassifikation der Begriffe wie bei dem OpenThesaurus ist nicht direkt möglich, da der UMLS-Metathesaurus kein allgemeinstes Konzept besitzt, welches als Oberbegriff für den gesamten Wissensspeicher festgelegt wurde. Beispielsweise „Disease“ oder „Acids“ können keinem allgemeineren Konzept zugewiesen werden.

4 Implementierung

Die grundsätzliche Idee für die Datenquellen ist die Gleiche und soll im Folgenden kurz erläutert werden: Zuerst wird die Datenquelle heruntergeladen und die vorhandene Struktur betrachtet. Nicht benötigte Daten werden gelöscht und der Rest der Daten in ein geeignetes Format zur Datenanfrage gebracht. Anschließend werden die Anfragen ausgeführt und die Ergebnisse angezeigt.

Grundsätzlich sollten die verschiedenen Methoden bzw. Herangehensweisen für die jeweils betrachtete Datenquelle die gleichen Ergebnisse liefern, jedoch ist aus Effizienzgründen diejenige zu bevorzugen, welche bei der Ausführung der Anfrage schneller ein Ergebnis liefert und nicht an eine Limitierung der Anfragen auf eine bestimmte Zeitspanne angewiesen ist.

Ist die eigentliche Implementierung abgeschlossen, so werden die geschriebenen Java-Klassen in das Mapping-Enrichtment-Modul des Projektes EnrichtmentCenter integriert. Dies ist ein Projekt, welches alle bisherigen Strategien zur Verbesserung des Schema Mappings beinhaltet. Hierzu gehören der WordNet-Ansatz und die in Abschnitt 2.4 vorgestellten Strategien, sowie Klassen für die Evaluation.

4.1 OpenThesaurus

Der folgende Abschnitt beschreibt die Umsetzung der beschriebenen Idee zur Implementierung am Beispiel von OpenThesaurus detaillierter.

4.1.1 Anfragemöglichkeiten

Für die auszuführenden Anfragen stehen mehrere Optionen zur Auswahl:

1. Webservice von OpenThesaurus / API- Zugriff
   Die Nutzung des Webservices ist jedoch an verschiedene Bedingungen geknüpft. Zunächst muss der Webservice immer Online sein, damit Anfragen gestellt werden können. Ebenfalls wird stets eine Internetverbindung vom Nutzer vorausgesetzt. Des Weiteren können nicht mehr als 60 Anfragen pro Minute gestellt werden, was für Testzwecke und die spätere Ausführung von vielen Anfrangen beim Abgleich von Attributen zwischen Schemata eine erhebliche Einschränkung darstellt. Daher wird auf diese Art des Zugriffs auf die Daten nicht weiter eingegangen.

2. Datenbank
   Hierbei werden alle Anfragen im MySQL-spezifischen SQL formuliert und mit Hilfe des JDBC-Treibers auf der Datenbank ausgeführt, welche eine Ergebnismenge zurückliefert.

Beim Formulieren der SQL-Statements muss darauf geachtet werden, dass diese schnell ein korrektes Ergebnis zurückliefern. Beispielsweise kann nach Synonymen mit Hilfe einer Subquery gesucht werden oder unter Verwendung eines Joins. Zweiteres liefert viel schneller das gleiche Ergebnis zurück. In der Tabelle 5 werden diese beiden Statements anhand des Beispielwortes „Mensch“ auf dem Originalschema veglichen.

	Subquery	Join
SQL-Statement	`select t1.word from term t1 where t1.synset_id = any (select t2.synset_id from term t2 where t2.word = 'Mensch') and t1.word <> 'Mensch' group by t1.word order by t1.word asc`	`select t1.word from term t1, (select t3.synset_id from term t3 where t3.word = 'Mensch') as t2 where t1.synset_id = t2.synset_id and t1.word <> 'Mensch' group by t1.word order by t1.word asc`
Zeit	6,5 Sekunden	0,02 Sekunden

Tabelle 5: Vergleich von zwei SQL-Statements für Synonyme am Beispiel des Wortes Mensch

Bei der Subquery-Variante ist zu beachten, dass das zu suchende Wort ein Homonym sein kann und daher die Subquery mehr als eine *synset_id* zurückliefern kann, weshalb hier das Keyword „any“ zum Einsatz kommt. Die äußere Query ermittelt aus den *synset_ids* die Wörter. Ein erheblich schnelleres Resultat der Abfrage kann erreicht werden, indem die Subquery durch einen Join ersetzt wird.
Wird die gleiche Anfrage mit dem Wort „Test“ durchgeführt, wird unter anderem das Wort „belletristisch“ als Synonym zurückgeliefert (Stand 17. September 2013). Dies ist wahrscheinlich ein falsches Synonym, welches eventuell nur aus Testgründen in der Datenbank vorhanden ist oder noch nicht auf Korrektheit geprüft wurde. In Folge dessen geht die Precision in geringem Maße zurück. Solche Wörter werden jedoch nur extrem selten auftreten. Jedoch wird hieraus ein Nachteil von OpenSource Projekten deutlich: die Mitarbeit der Community macht Überprüfungen unerlässlich. Dies erfordert Zeit, weshalb beim regelmäßigen Veröffentlichen von neuen Versionen, im Falle von OpenThesaurus jeden Tag, falsche bzw. stark fragwürdige Relationen auftreten können.
Da die Anfragen für Antonyme, Assoziationen, Hyper- und Hyponyme bereits recht komplex werden, erscheint eine Änderung des Datenbankschemas sinnvoll, um die Anfragen zumindest geringfügig „intuitiver“ zu gestalten.

4.1.2 Änderungen am OpenThesaurus

Beim ersten Start des Programms wird der SQL-Dump aufgesetzt und anschließend das Original-Schema in das geänderte Schema überführt und die aktuelle Version in der OpenThesaurusVersionTimestamp.txt gespeichert. Bei erneutem Ausführen des Programms liegt bereits das korrekte Schema vor, insofern keine neue Version im entsprechenden Verzeichnis vorliegt und unter der Annahme, dass weder eine Änderung an der Struktur der Datenbank oder der OpenThesaurusVersionTimestamp.txt, vorgenommen wurde. Daher ist es sinnvoll, beim Ausführen des Programms zuerst zu überprüfen, ob eine Änderung des Datenbankschemas notwendig ist. Dadurch wird Zeit gespart, wenn bereits das korrekte Schema für die Datenbank vorliegt.
Nehmen wir an, es wurde eine neue Version oder ein falsches Datenbankschema entdeckt, dann werden folgende Schritte durchgeführt:

1. Löschen nicht benötigter Daten
   a) Tabellen: Dies umfasst unter anderem Tabellen zur Kategorisierung von Wörtern, Metadaten über den OpenThesaurus oder eine Einteilung der Wörter in ihren Sprachgebrauch. Am Ende verbleiben nur die Tabellen *link_type*, *synset_link*, *term* und *term_link*.
   b) Spalten: Nun werden nicht benötigte Spalten entfernt. Das Ergebnis nach diesem Schritt ist in Abbildung 3 dargestellt.

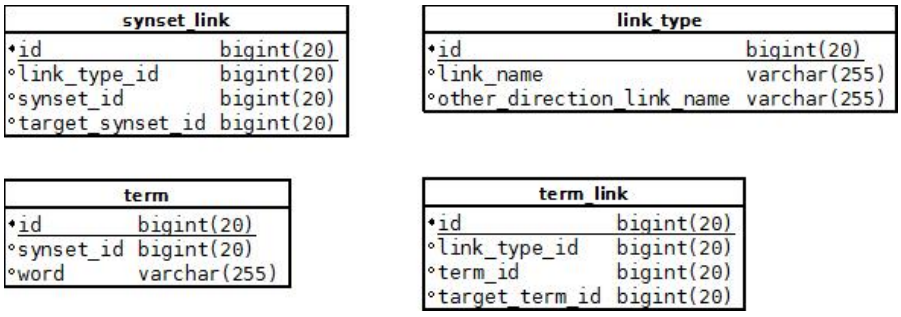

Abbildung 3: Vier Tabellen mit minimaler Anzahl Spalten

2. Hinzufügen des Datensatzes für Antonyme in der Tabelle *link_type*.
3. Update *link_type_id* auf id des Antonym-Datensatzes in *link_type*.
4. Füge *synset_id* und *target_synset_id* in *term_link* hinzu.
5. a) Finde zu *target_term_id* die *synset_id* und Update *target_synset_id*.
   b) Finde zu *term_id* die *synset_id* und Update *synset_id* in *term_link*.
   c) Entferne eventuell entstandene Duplikate.
6. Führe *term_link* und *synset_link* zusammen mit fortlaufender id durch Inserts in *synset_link*.

7. Löschen der Tabelle *term_ link*.

8. Umbenennen der Tabelle *synset_ link* in *link*.

Der erste Schritt wird direkt in Java ausgeführt, da der Datenbankname nur hier bekannt ist. Alle folgenden Schritte werden in der Datei DatabaseSchemaParser.sql gespeichert und bei Bedarf in Java per ScriptRunner ausgeführt. Auf eine Anpassung der Datentypen des OpenThesaurus auf realistische Wertebereiche wird verzichtet, da keine erhebliche Verbesserung der Laufzeit bei Anfragen zu erwarten ist.
Das finale Datenbankschema für den OpenThesaurus in jeder Version sollte die in Abbildung 4 dargestellte Struktur besitzen:

Abbildung 4: Finales Datenbankschema

4.1.3 Anfragen am fertigen Datenbankschema

Zum Testen der Effizienz der Anfragen wurde eine Testdatei angelegt. Diese enthält pro Zeile zwei Wörter bzw. Phrasen, die durch ein Trennzeichen voneinander gekapselt sind. Um ein realistisches Szenario zu modellieren, wurden 1000 Wort- bzw. Phrasenpaarungen getestet. Diese setzen sich aus ca. 200 Synonymen, 200 Hyper- und Hyponyme, 200 Assoziationen, 50 Antonymen und 350 zufälligen Paarungen zusammen. Bei der Implementierung wurde eine Reihenfolge zur Überprüfung der Paarungen in der Datenbank festgelegt. Als erstes wird geprüft, ob beide Wörter identisch sind. Ist dies nicht der Fall, wird nach Synonymen, dann nach Antonymen, Assoziationen, Hyponymen und zuletzt nach Hypernymen gesucht. Da Hypo- und Hypernyme meistens mehr Zeit benötigen als alle anderen Korrespondenzen, werden diese erst am Schluss geprüft. Wurde vorher bereits eine Korrespondenz gefunden, wird diese zurückgegeben. Eine Suche nach weiteren Korrespondenztypen ist dann nicht nötig.
Bei den Anfragen werden die jeweiligen *synset_ ids* der Wörter bzw. Phrasen in die Anfrage mit einbezogen, das heißt eine Gesamtabfrage, die den Durchschnitt über die jeweils gefundenen Korrespondenzen bildet, wird ausgeführt. Dabei ist die Idee, die Anfrage nicht bis zu ihren Wörtern herunterzubrechen, sondern die resultierenden *synset_ ids* der Wörter gegeneinander abzugleichen. Wenn die Ergebnismenge nicht leer ist, wurde eine entsprechende Korrespondenz bezüglich der gestellten Anfrage gefunden. Da die Anfragen komplett in SQL gestellt und nicht mit Javacode vermischt werden, um ein entsprechendes Resultat zu erhalten, wird ein besseres Verständnis gewährleistet. Die Anfragen können dementsprechend ohne Kenntnis der Java-Quellcode Dateien nachvollzogen werden. Dennoch sind die Anfragen für Assoziationen und Antonyme mit 16 Zeilen bereits recht komplex.

Synonyme
Zuerst werden die jeweiligen *synset_ids* der Wörter bzw. Phrasen gefunden und anschließend mit Hilfe der Tabelle *term* verglichen. Enthält die Schnittmenge der *synset_ids* mindestens eine *synset_id*, sind die Eingabewörter bzw. -phrasen Synonyme. Die folgende SQL-Query überprüft, ob die Wörter „apfelsine“ und „orange“ Synonyme sind.

```
select t.synset_id
from term t,     (select t3.synset_id
                  from term t3
                  where t3.word = 'apfelsine') as word1,
                  (select t3.synset_id
                  from term t3
                  where t3.word = 'orange') as word2
where    word2.synset_id = word1.synset_id
        and t.synset_id = word2.synset_id
        and t.synset_id = word1.synset_id
group by t.synset_id
```

Antonyme
Beim Test auf Antonyme werden zuerst die *synset_ids* von Wort1 herausgefunden. Diese werden mit der Tabelle *link* abgeglichen. Die entsprechende *link_type_id* für Antonyme muss ebenfalls gelten. Da Antonyme nur in eine Richtung gespeichert werden, muss ebenfalls auch die Spalte *target_synset_id* überprüft werden, die Ergebnismengen werden vereinigt. Anschließend wird diese mit der Menge der *synset_ids* von Wort2 geschnitten, Duplikate entfernt und das Ergebnis angezeigt. Auf weitere SQL-Statements wird an dieser Stelle verzichtet. Diese können in der Quellcode-Datei „Relations.java“ nochvollzogen werden. Wird eine Korrespondenz zwischen den Wörtern festgestellt, so kann ein Match unter Garantie ausgeschlossen werden. Korrespondenzen, die durch andere Strategien falsch erkannt wurden, können ausfindig gemacht und der Fehler behoben werden. Somit ist ein positiver Einfluss auf das Mapping und damit die Precision gegeben.

Assoziationen
Für Assoziationen ist die Anfrage die gleiche wie bei den Antonymen, jedoch mit einer anderen *link_type_id*.

Hypernyme und Hyponyme
Die Anfrage für direkte Hyper- und Hyponyme erfolgt in folgendem Schema:
Finde zu Wort1 die *synset_ids* aus der Tabelle *term* und teste in der Tabelle *link* die Oberbegriff/Unterbegriff-Beziehung mit der entsprechenden *link_type_id*. Dabei steht in der Spalte *target_synset_id* der allgemeinere Begriff, in *synset_id* der Speziellere. Von der Ergebnismenge wird wieder der Durchschnitt gebildet und anschließend ausgegeben.

Es gibt jedoch auch Wortpaarungen, die keine direkten Hyper- bzw. Hyponyme in der Hierarchie der Datenquelle darstellen. Im Folgenden soll erklärt werden, wie Hypernyme und Hyponyme bei tieferer Schachtelung gefunden werden und wie die sich anbietende Rekursion zum Finden von Hyper- und Hyponymen n-ten Grades prinzipiell erfolgt. Hierzu wird das folgende Beispiel betrachtet:

Mensch „is-a“ Lebewesen,
Berufstätiger „is-a“ Mensch,
Informatiker „is-a“ Berufstätiger,
Datenbankadministrator „is-a“ Informatiker.

Ausgehend von dem Wort Mensch werden bei einer Rekursionstiefe von eins die Wörter Lebewesen als direktes Hypernym und Berufstätiger als direktes Hyponym von Mensch gefunden. Bei einer Rekursionstiefe von zwei wird zusätzlich der Informatiker als Hyponym zweiten Grades zu Mensch gefunden. Schließlich bei einer Rekursionstiefe von drei wird auch der Datenbankadministrator als Teilmenge der Informatiker gefunden und ist ebenso ein Hyponym zu Mensch.

Da die einzelnen Terme jedoch einer Synonymgruppe zugeordnet werden und daher eine *synset_ id* erhalten, läuft die Rekursion über diese ab. Die nachfolgende Tabelle 6 zeigt noch einmal anhand eines einfachen Beispiels, in welcher Rekursionstiefe die Hyper- bzw. Hyponyme gefunden werden, falls eine Korrespondenz gefunden werden kann. Ausgangspunkt ist hier die Suche nach einem Wort, dessen *synset_ id* = 2 ist.

synset_ id		Rekursionstiefe	
Hypernym	Hyponym	1	2
1	2	Ja	Ja
2	3	Ja	Ja
3	4	Nein	Ja
5	1	Nein	Ja
6	2	Ja	Ja
7	8	Nein	Nein

Tabelle 6: Suche nach Hyper- und Hyponymen ausgehend von *synset _id* = 2

Hierbei werden die gefundenen *synset_ ids* weiter in die jeweilige Richtung geprüft, das heißt für Begriffe (bzw. *synset_ ids*), die allgemeiner sind, werden weitere Oberbegriffe gesucht und für speziellere Begriffe analog weitere Unterbriffe. Die Abfrage setzt sich also fort, jedoch nur in eine Richtung, da sonst *synset_ ids* doppelt gefunden würden und was noch verheerender wäre: eine Endlosschleife würde entstehen. Werden keine neuen *synset_ ids* gefunden oder ist die maximale Rekursionstiefe erreicht, wird die Rekursion beendet. Im konkreten Beispiel wird bei einer gleichzeitigen Suche nach Hyper- und Hyponymen im ersten Rekursionsschritt, also Rekursionstiefe 1, das Wort mit der *synset_ id* = 1 als Hypernym zu *synset_ id* = 2 und das Wort mit der *synset_ id* = 3 als Hyponym

zu $synset\_id = 2$ erkannt. Bei Rekursionstiefe 2 wird nun nach den korrespondierenden $synset\_ids$ des Datensatzes aus Rekursionstiefe 1 gesucht. Im Beispiel wird noch ein Hypernym und ein Hyponym gefunden, welche in Rekursionstiefe 1 noch nicht erkannt wurden. In der implementierten Version wird nicht parallel, sondern hintereinander nach Hypernymen und Hyponymen gesucht.

Durch die Annahme, dass ein Wort im Schnitt mehr Unterbegriffe als Oberbegriffe besitzt und für Hyponyme alle Hypernyme von Wort1 gefunden werden müssen, um einen Abgleich mit Wort2 durchzuführen, werden bei der Reihenfolge der Abfrage im Java-Programm Hyponyme vor den Hypernymen überprüft.

Wurde keine passende Korrespondenz gefunden, so wird nicht wie bei der Webschnittstelle des OpenThesaurus von einem Tippfehler ausgegangen, bei der mittels eines Ähnlichkeitsmaßes dennoch passende Wörter ermittelt werden. Hier wird durch eine geringfügige Anpassung am Wortende versucht die Grundform nachzubilden und das geänderte Wort erneut in der Datenbank gesucht. Mehr Änderungen werden nicht vorgenommen, denn es wird davon ausgegangen, dass die Schemata frei von Rechtschreibfehlern sind. In Folge dessen kann keine Korrespondenz zwischen den Wörtern gefunden werden.

4.1.4 Überblick über den Quellcode

Für die Implementierung wurde die eigentliche Aufgabe in kleinere Teilaufgaben zerlegt, gelöst und anschließend die Ergebnisse zusammengeführt. Für die Lösung einzelner Teilaufgaben wurde je eine Java-Klasse angelegt. In Abbildung 5 ist eine Übersicht in Form eines UML-Diagramms über den Quellcode anhand einzelner Klassen und deren Funktionen dargestellt.

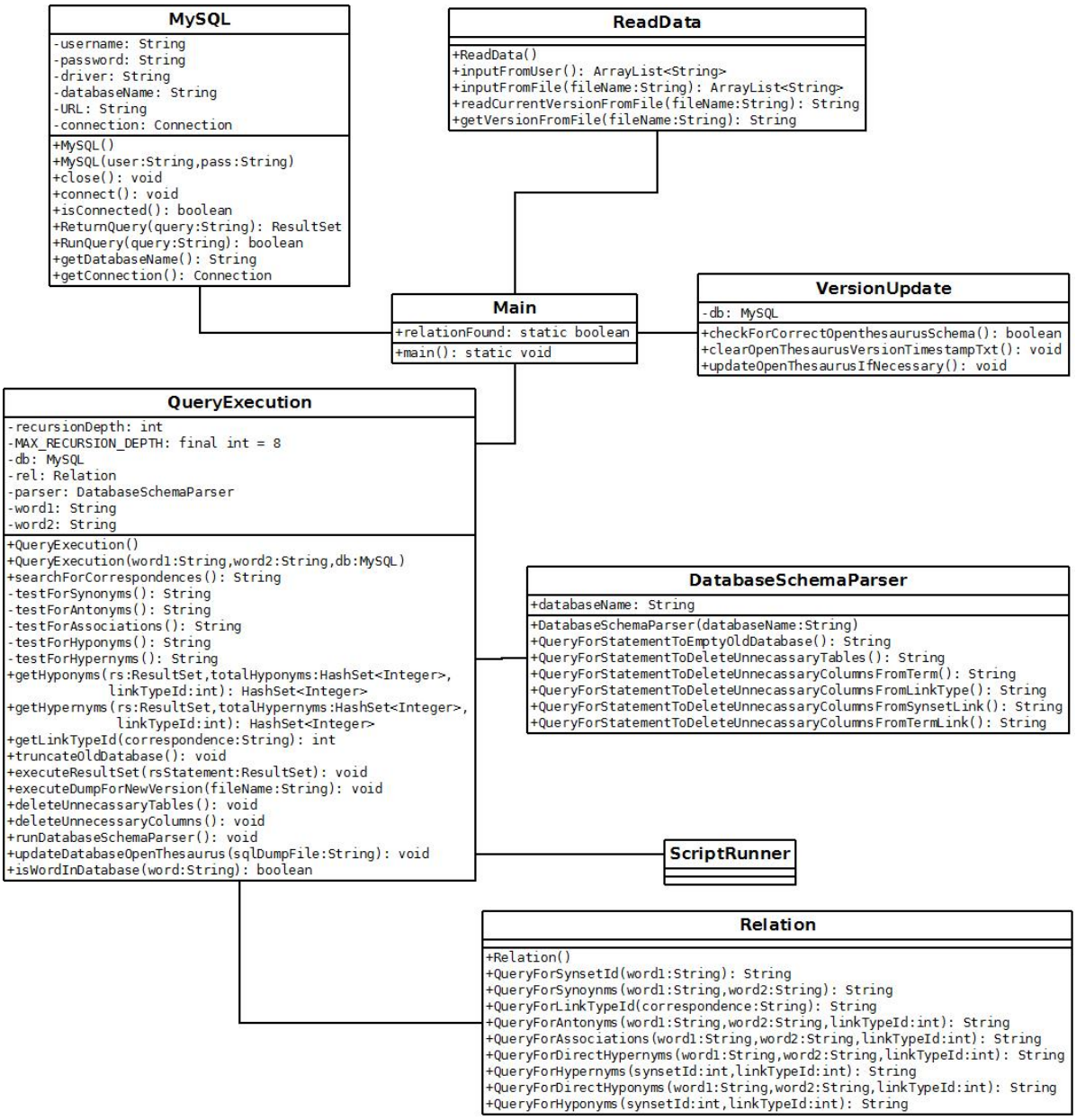

Abbildung 5: UML-Diagramm: Implementierte Klassen für den OpenThesaurus

Hierbei löst die Klasse *Main* die eigentliche Aufgabe des Suchens nach Beziehungen in der Datenbank. Zuerst wird mit *checkForCorrectOpenThesaurusSchema* überprüft, ob das Datenbankschema geändert wurde. Ist dies der Fall, so wird die *OpenThesaurusVersionTimestamp.txt* geleert. Anschließend wird getestet, ob eine neue Version des OpenThesaurus vorliegt. Hierzu wird der Inhalt der obigen Textdatei mit dem String, der die Version des vorliegenden SQL-Dumps enthält, verglichen. Stimmen diese beiden Strings nicht überein, wird die alte Datenbank geleert und der korrekte bzw. neue SQL-Dump mit Hilfe des *ScriptRunners*, der beliebig viele SQL-Statements hintereinander ausführen kann, eingespielt. Die Textdatei wird auf die Version des OpenThesaurus SQL-Dumps aktualisiert. Nun kommt der *DatabaseSchemaParser* zum Einsatz. Die vorliegende Struktur muss nun wie in 4.1.2 beschrieben, angepasst werden. Hierfür werden die Methoden des DatabaseSchemaParser und anschließend die *DatabaseSchemaParser.sql* Datei benötigt, die wieder mit dem *ScriptRunner* ausgeführt wird.

Ist dieser Prozess abgeschlossen, können die Anfragen an die Datenbank beginnen. Dazu wird eine Instanz der Klasse *ReadData* erstellt, die beispielsweise die Datei *openthesaurusTestGleichverteilt.txt* einliest und deren Wort- bzw. Phrasenpaarungen zeilenweise extrahiert. Um auf die Datenbank zugreifen zu können, wird eine Instanz der Klasse *MySQL* erzeugt. Für jede Wortpaarung wird nun ein Objekt von *QueryExecution* erzeugt und mit der Methode *searchForCorrespondences* nach Relationstypen in der Datenbank gesucht. Diese Methode ruft demnach alle vorhandenen Anfragen für die Korrespondenzen auf. Hierzu gehören Synonyme, Antonyme, Assoziationen, Hypernyme und Hyponyme. Es wird ein Ergebnis mit einer gefundenen Relation oder null zurückgeliefert. Dies entspricht dem Ergebnis für diese Wortpaarung und kann ausgegeben werden.

4.1.5 Entschachtelung der Relationen

Aufgrund der teils langen Anfragedauer für Hypernyme und Hyponyme erscheint es sinnvoll, die rekursiven Relationen zu entschachteln. Betrachten wir das zu folgende Beispiel:

Maurer „is-a“ Berufstätiger,
Berufstätiger „is-a“ Mensch.

Diese beiden Korrespondenzen sind in einer Tabelle gespeichert. Bei einer Entschachtelung würde zusätzlich die Korrespondenz Maurer „is-a“ Mensch hinzukommen. Die ursprüngliche Schachtelung wurde aufgelöst. Eine Entschachtelung der Relationen erfordert mehr Speicher, jedoch kann dadurch Zeit bei der Anfrage von ursprünglich tiefgeschachtelten Relationen gespeichert werden. Alle anderen Anfragen werden geringfügig langsamer, da nun mehr Datensätze in der Datenbank gespeichert werden. Insgesamt ist die Zeitperformance also vom Benchmark abhängig. Bei einer relativ gleichmäßigen Verteilung der Korrespondenztypen im Benchmark steigt die Zeit in der Summe für die Anfragen um einige Sekunden an. Da diese Idee außerdem zusätzlichen Speicher benötigt, ist sie für den OpenThesaurus nicht von Vorteil und die zuerst vorgestellte Idee wird in das Mapping-Enrichment-Modul eingebunden.

4.1.6 Limitierungen

Da der OpenThesaurus die allgemeine Domäne repräsentiert, weshalb große „Sprünge“ bei der Verallgemeinerung und Spezifizierung vorkommen können, lässt sich auch über eine „sinnvolle“ Rekursionstiefe diskutieren. Beispielsweise ist es höchst fraglich, ob die Attribute „Irgendetwas“ und „Maurer“ als eine bestimmte Korrespondenz erkannt werden sollten, denn diese Erkenntnis ist nichtssagend.

Bei einer Datenquelle mit einer speziellen Domäne ist dies anders. Da hier die allgemeinsten Konzepte noch immer spezifisch sind, kann man die maximale Rekursionstiefe hochsetzen. Die gefundenen Korrespondenzen sind immer noch aussagekräftig. Die einzige Einschränkung wäre hier bei einer rekursiven Speicherung der benötigte Zeitaufwand für die gesamte Rekursion. Bei einer entschachtelten Speicherung der Relationen ist der benötigte Speicher zu beachten.

Ein weiteres Problem, welches auch auf den UMLS-Metathesaurus zutrifft ist die Einschränkung auf die Grundform der Wörter. Da jedoch in den Schemata auch der Plural von Wörter vorhanden sein kann, muss in einem zusätzlichen Schritt eine „künstliche“ Einzahl aus den Attributen erzeugt werden, die wiederum im Wissensspeicher angefragt wird. Dies birgt jedoch aufgrund von Umlauten und unregelmäßigen Wörtern die Gefahr von Fehlern.

4.2 UMLS-Metathesaurus

Die Idee bei der Vorgehensweise ist grundsätzlich die Gleiche wie beim OpenThesaurus. Dies beinhaltet als erstes das Löschen nicht benötigter Tabellen und Spalten und die anschließende Entfernung von Duplikaten, die aufgrund dieser Reduktion entstanden sein können. Zusätzlich werden jedoch weitere Schritte durchgeführt, die nicht benötigte Daten verwerfen, um die Datenmenge so gering wie möglich zu halten. Hierzu gehört das Zusammenführen der vorhandenen Relationen mit den vorhandenen Konzepten. Hierbei werden sowohl Konzepte, die in der Tabelle mit den Relationen nicht vorkommen, als auch Relationen, denen kein Konzept zugeordnet werden kann, entfernt. Da die Tabelle *mrrel* sowohl Hypernyme als auch Hyponyme als eigenständige Relationen speichert, jedoch das gleiche Ergebnis mit nur einer Relation durch Überprüfen beider Seiten bei der Abfrage erhalten werden kann, kann ein Relationstyp komplett gelöscht werden. Die Wahl fiel auf die Relation der spezielleren Begriffe.

Diese Strategie führt die Anfragen genauso aus wie der geänderte OpenThesaurus (nur die Tabellen- und Spaltennamen sind verschieden), jedoch ist es möglich, dass Anfragen bei langen rekursiven Abfragefolgen, welche bei Hypernymen und Hyponymen auftreten können, bis zu 10 Sekunden dauern. Eine genauere Analyse der Anfragedauer folgt in Abschitt 4.2.2. Diese lange Laufzeit der Anfragen liegt in der Anzahl der Datensätze und der teilweise stark verzweigenden Relationen begründet.

Dies ist nicht akzeptabel, weshalb alternative Ideen der Umsetzung benötigt werden:

1. Zusammenfügen der Tabellen zu einer Tabelle durch Übersetzen der id's in Strings, um einen Join zu vermeiden.
2. Hypernym- bzw. Hyponym Relationen rekursiv auflösen, das heißt „Entschachteln" der Relationen.
3. Laden der Daten in den Hauptspeicher unter Nutzung einer Hashtabelle.

4.2.1 Alternative Umsetzungen

Der folgende Abschnitt bespricht alternative Ideen der Umsetzung für den UMLS-Metathesaurus. Alle Ideen sind auch für andere Datenquelle anwendbar, wenn auch nicht jede Idee zu besserer Performance führt. Trotzdem werden alle getesteten Alternativen vorgestellt, um das Spektrum der Ideen zu präsentieren.

Id-Auflösung

Dies bedeutet, dass das Datenbankschema denormalisiert wird und infolgedessen redundante Daten in Form von mehrfach gespeicherten Strings entstehen. Für dieses Vorhaben ist es nötig, die Konzept-Ids der Tabelle *mrrel* mit Hilfe der Tabelle *mrconso* in Strings aufzulösen und in die Tabelle *triple* zu übernehmen. Anschließend müssen die Synonyme aus der Tabelle *mrconso* über die Konzept-Ids ermittelt werden und in der Tabelle *triple* hinzugefügt werden.

Bei dieser Variante müssen durch die Denormalisierung Stringvergleiche anstelle von Id-Vergleichen stattfinden, welche ebenfalls Zeit in Anspruch nehmen. Außerdem benötigt diese Methode mehr Speicher, da die Strings nicht mehr über eine id referenziert werden, sondern direkt in der Tabelle abgespeichert sind. Falls die Zeitersparnis durch den nicht mehr benötigten Join größer ist als der Zeitverbrauch, der für die Stringvergleiche benötigt wird, so wäre diese Methode gegenüber der ersten zu bevorzugen.

Entschachtelung der Relationen

Bei dieser Idee werden alle Hypernym- bzw. Hyponym-Beziehungen rekursiv aufgelöst und die entschachtelten Relationen abgespeichert. Dies erfordert mehr Speicherplatz, ursprünglich tiefgeschachelte Hypernym- bzw. Hyponym-Relationen können nun aber mit nur einer Anfrage an die Datenbank gefunden werden.

Hashmap

Eine Hashmap oder auch Hashtabelle besitzt für jedes Element einen Schlüssel/Key und einen zugehörigen Wert/Value. Über den Key können sehr schnell Datenelemente ausfindig gemacht werden. Wird eine Hashmap benutzt, werden die kompletten Informationen in den Hauptspeicher geladen. Eine Anfrage sollte dann sehr schnell Ergebnisse erzielen.

4.2.2 Anfragen

Um die Performance der jeweiligen Implementierungen überprüfen zu können, wird ein realistisches und zugleich breit gestreutes Feld an Beziehungstypen zwischen den Wortpaarungen benötigt. Hierbei sollen sowohl schnelle als auch potentiell langsame Anfragen auftreten, die bei einer ausreichend großen Menge einen gesunden Mittelwert für die Ausführungszeit für eine „Durchschnittsanfrage“ ergeben. Als Testmenge für die Anfragen wurden insgesamt 1000 Wortpaare aus dem UMLS-Metathesaurus gewählt. Dabei wurde versucht, ein realistisches Szenario durch Einbringen der verschiedenen Relationen nachzuahmen. Insgesamt besteht die Testmenge aus ca. 30% Assoziationen, ca. 30% Hyper- und Hyponyme, 10% Synonymen und ca. 30% zufälligen Wortpaarungen.

Die benötigte Zeit für die Ausführung der beschriebenen Testmenge betrug 87 Sekunden für die Variante, wie sie auch bei der OpenThesaurus-Strategie zum Einsatz kommt. Eine Anfrage dauert demzufolge im Mittel knapp 90 ms, entschieden zu lange für ein großes Testszenario. Das Einführen von Indizes hat leider ebenfalls nicht den erwünschten Performanzschub erzielt.

Bei einer Entschachtelung der Relationen werden sogar 362 Sekunden benötigt. Dies liegt an der Vergrößerung der Anzahl der Datensätze und der damit verbundenen aufwendigeren Suche, so dass eine Anfrage im Schnitt länger braucht als ohne Entschachtelung. Außerdem benötigt diese Variante mehr Speicherplatz, weshalb sie keine Alternative zur bisherigen Idee darstellt.

Die Idee der HashMap liefert sehr schnell Resultate mit dem gleichen Ergebnis. Das einmalige Preprocessing zur Erstellung der Objekt-Datei mit der Endung .ser für Serializeable ist nach 3,5 Sekunden abgeschlossen. Diese Datei enthält alle benötigten Informationen in einem binären Format. Das Einlesen dieser Datei, welches zu Beginn notwendig ist, dauert zwei Sekunden. Dieser einmalige Overhead zahlt sich vor allem bei vielen zu vergleichenden Wortpaarungen aus, da die Anfragen sehr schnell ausgeführt werden können. Für die Anfragen des Testszenario werden 270 Millisekunden benötigt. Eine Anfrage dauert im Schnitt also 0,27 Millisekunden bzw. 270 Mikrosekunden. Diese Umsetzung schlägt alle anderen Ideen um Längen, weshalb diese in das Mapping-Enrichtment-Modul eingebettet wird.

4.2.3 Überblick über den Quellcode

Auch bei dieser Implementierung dient ein UML-Diagramm, wie in Abbildung 6 gezeigt, der Veranschaulichung über die Klassen und Methoden der UMLS-Metathesaurus Hashmap-Umsetzung.

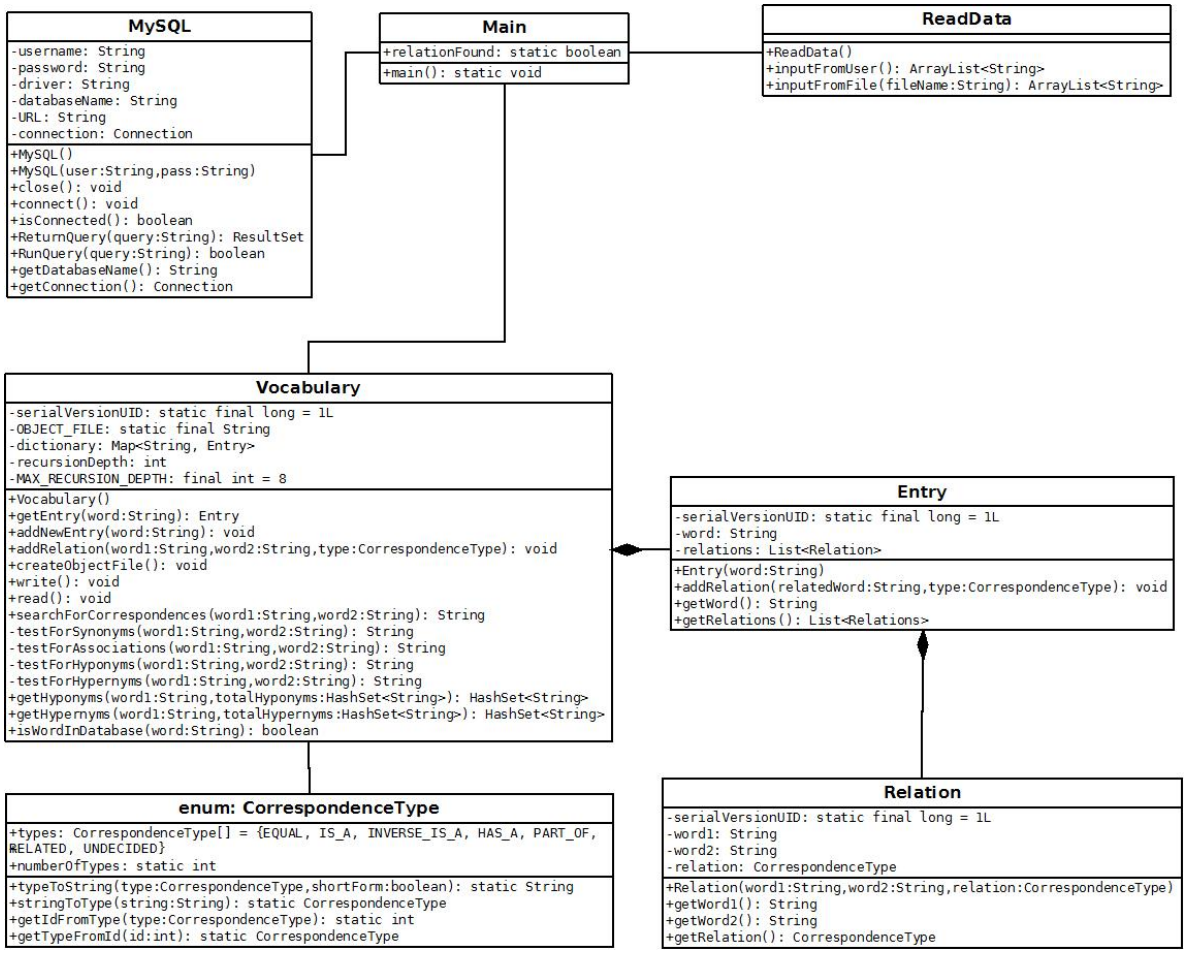

Abbildung 6: UML-Diagramm: Implementierte Klassen für den UMLS-Metathesaurus

Die Klasse *Main* ruft getreu ihrem Namen die *main*-Methode auf, welche diese Strategie vollständig umsetzt. Dabei ist die Idee, Tripel der Form (Quelle, Relation, Ziel) in die Hashmap zu laden. Hierzu ist es sinnvoll, die mit der alternativen Idee der Id-Auflösung entstandene Tabelle *triple* zu nutzen. Die Relationen wurden jedoch zur besseren Lesbarkeit in „equal“, „related“ und „is-a“ überführt. Das Hauptvokabular soll eine Hashmap bilden, mit dem jeweiligen Wort als Key und einem Objekt Entry/Eintrag als Wert. Dieser Eintrag besteht aus einer Liste von Relationen, welche wiederum Tripel der Form (Wort1, Wort2, Relation) enthält. Wort1 ist nur aus Gründen der Vollständigkeit und Wiederverwendbarkeit vorhanden, kann aber bereits über den Key der Hashmap ermittelt werden. Alle Objekte zusammen repräsentieren den Wortschatz, der als Grundlage für die Anfragen dient.

Zum Erzeugen der in 4.2.1 beschriebenen Hashmap wird die Methode *createObjectFile* aufgerufen. Diese benötigt die Strings der Tabelle *mrconso* und alle Spalten der Tabelle *triple* aus der vorhandenen MySQL-Datenbank. Alle Beziehungstypen werden in beide Richtungen gespeichert. Dies hat den Vorteil, dass nur eine Suchanfrage nach dem jeweiligen String erforderlich ist, wodurch der Prozess der Anfrage erleichtert wird.

Damit nicht bei jedem Programmneustart auf die Datenbank zugegriffen werden muss, werden die erstellten Objekte als .ser abgespeichert. Dadurch müssen die Objekte nicht erst zur Laufzeit erstellt werden, sondern werden nur noch eingelesen. Demzufolge ist das Erstellen der serialisierten Objektdatei ein einmaliger Overhead. Wird nun nach Relationen von Wortpaarungen nach Einlesen mit Hilfe einer Instanz der Klasse *ReadData* gesucht, wird für Wort1 die Liste von Relationen geholt, anschließend in diesen nach Wort2 und dem jeweiligen Beziehungstyp gesucht und dieser im Falle einer Übereinstimmung zurückgeliefert. Für Hypernyme und Hyponyme findet dieser Prozess wieder rekursiv bis zu einer maximalen Rekursionstiefe statt. Dabei enthält die Aufzählung/Enumeration *CorrespondenceType* alle Korrespondenztypen.

5 Evaluation

Im nachfolgenden Abschnitt wird eine Evaluation der Implementierungen der ausgewählten Datenquellen durchgeführt. Hierbei kommen geeignete Benchmarks für die Datenquellen zum Einsatz, die die jeweilige Sprache und Domäne berücksichtigen. Diese manuell erstellten Benchmarks repräsentieren einen Goldstandard und die Ergebnisse der Implementierungen werden gegen diesen getestet. Es resultieren richtig und falsch erkannte Korrespondenzen, mit deren Hilfe Precision, Recall und F-Measure als zuverlässige Gütemaße zur Bewertung von Systemen berechnet werden. Um einen Vergleich ziehen zu können, wie gut die erhaltenen Ergebnisse sind, werden zusätzlich die Ergebnisse aller Maße ohne die jeweilige Background Knowledge Strategie dargestellt.

Dabei ist zu beachten, dass die Evaluation in diesem Bereich ein schwieriges Unterfangen darstellt, da Beziehungen zwischen Wörtern auch für den Menschen nicht immer eindeutig sind. Hierzu zählt unter anderem die Erwartungshaltung eines Menschen zu den Wörtern. Das klassische Beispiel stellt die Erdbeere dar. Im biologischen Sinne wird sie den Nüssen zugeordnet, ein Verbraucher wird jedoch bei Beeren oder allgemeiner in der Obstabteilung danach suchen. Dies verdeutlicht die Problematik der Unterscheidung nach Gefühl und Definition, wobei letztere mit viel Recherche verbunden ist und wenig mit einer „intuitiven“ Zuordnung gemein hat.

Für die folgende Evaluation werden die Synonyme von allen anderen Korrespondenztypen getrennt und letztere als „nicht-triviale“ Korrespondenztypen bezeichnet. Besonderer Augenmerk liegt hier auf den nicht-trivialen Korrespondenztypen, da diese die größten Probleme bereiten und daher bislang einen recht geringen Recall aufweisen. Dieser soll durch die implementierten Datenquellen angehoben werden, ohne jedoch die Precision zu verschlechtern.

5.1 OpenThesaurus

Zur Evaluation des OpenThesaurus kommen zwei verschiedene Benchmarks zum Einsatz, die verschiedene Web Directories repräsentieren. Da der OpenThesaurus nur die Wörter und Phrasen der deutschen Sprache speichert, ist ein Benchmark nur in diesem Rahmen geeignet. Eine Verbesserung des Benchmarks in einer anderen Sprache ist mit dem OpenThesaurus nicht zu erwarten.

Benchmark 1 setzt sich aus insgesamt 340 Korrespondenzen zusammen, von denen im Goldstandard 277 als Synoynme festgelegt wurden.

Benchmark 2 beinhaltet Elemente der beiden Hierarchien Web und Yahoo und besteht insgesamt aus 234 Korrespondenzen. 155 der Korrespondenzen wurden im Benchmark als Synonyme klassifiziert.

Es ist zu erkennen, dass den Synonymen in den Benchmarks große Bedeutung zugeschrieben wird, da diese den wichtigsten Beziehungstyp beim Schema Matching darstellen. Die Gesamtergebnisse der beiden Benchmarks sind in Tabelle 7 dargestellt.

	Precision		Recall		F-Measure	
OpenThesaurus	ohne	mit	ohne	mit	ohne	mit
Benchmark 1	86,4%	87,0%	89,9%	90,7%	88,1%	88,8%
Benchmark 2	73,5%	75,6%	78,1%	79,7%	75,8%	77,6%

Tabelle 7: Precision, Recall und F-Measure bei den getesteten Benchmarks

Die Ergebnisse in ihrer Gesamtheit sind zufriedenstellend, jedoch liegt dies an der Tatsache, dass Synonyme bereits recht zuverlässig durch andere Strategien erkannt werden und in den Benchmarks einen Recht hohen Anteil besitzen. Dennoch unterscheiden sich die Benchmark-Ergebnisse nicht unerheblich. Dies lässt den Schluss zu, dass ein Benchmark letztendlich auch nur einen Richtwert darstellt und die zu überprüfenden Korrespondenzen der Schemata die Ergebnisse bedingen. Durch den Einsatz des Hintergrundwissens des OpenThesaurus sollten jedoch bei jedem angemessen großen Benchmark Precision, Recall und F-Measure besser werden. In welcher Größe die Verbesserungen liegen, ist von dem jeweils ausgewählten Benchmark abhängig. Der Recall kann im Idealfall um mehrere Prozentpunkte steigen und auch die Precision kann durch Erkennen falscher Korrespondenzen einen höheren Wert erreichen. Es ist jedoch zu erwarten, dass die Verbesserung beim Recall stärker ausfällt als bei der Precision, da die Background Knowledge Strategie genau das Problem des Recall angeht, speziell für nicht-triviale Korrespondenztypen.

Die nach Korrespondenztypen aufgeschlüsselten Ergebnisse der Benchmarks sind in Tabelle 8 und 9 dargestellt.

	Precision		Recall		F-Measure	
OpenThesaurus / Korrespondenz	ohne	mit	ohne	mit	ohne	mit
trivial	88,8%	89,7%	95,3%	95,3%	92,0%	92,5%
nicht-trivial	69,7%	69,5%	47,6%	50,7%	58,6%	60,1%

Tabelle 8: Precision, Recall und F-Measure bei Benchmark 1, unterschieden nach Korrespondenztypen

OpenThesaurus / Korrespondenz	Precision		Recall		F-Measure	
	ohne	mit	ohne	mit	ohne	mit
trivial	74,7%	76,1%	91,6%	92,9%	83,1%	84,5%
nicht-trivial	68,1%	73,3%	38,4%	41,7%	53,2%	57,5%

Tabelle 9: Precision, Recall und F-Measure bei Benchmark 2, unterschieden nach Korrespondenztypen

Aus den beiden Tabellen wird ersichtlich, dass die Ergebnisse, besonders beim Erkennen nicht-trivialer Korrespondenzen, besser werden. Dies folgt vor allem aus der Tatsache, dass das Erkennen nicht-trivialer Beziehungen durch andere Strategien begrenzt ist und eine Sprache im Allgemeinen viele unregelmäßige Wörter besitzt. Die OpenThesaurus-Strategie erkennt beispielsweise die „equal"-Beziehung zwischen den Begriffen „alternative Medizin" und „Alternativmedizin", welche mit keiner anderen Strategie erkannt wird. Des Weiteren sind die Ergebnisse für Synonyme im Vergleich zu anderen Beziehungstypen bereits sehr gut, weshalb eine Verbesserung in diesem Bereich nur noch geringfügig möglich ist.
Insgesamt liegen die Verbesserungen der Ergebnisse der einzelnen Maße im Bereich von 1 - 5%, welche für die Einbindung einer einzelnen Datenquelle positiv zu betrachten ist.

5.2 UMLS-Metathesaurus

Da der UMLS-Metathesaurus vollständig in Englisch gespeichert wird, bringt nur ein Benchmark mit englischen Korrespondenzen die gewünschte Veränderung in den Ergebnissen. Benchmarks in anderen Sprachen oder aus anderen Domänen würden keine oder nur minimale Verbesserungen erzielen. Daher wird ein Benchmark verwendet, der im EnrichmentCenter den Namen „Health" besitzt. Dieser enthält insgesamt 395 Korrespondenzen, von denen der Großteil, genauer 354, den trivialen Korrespondenztyp im Goldstandard aufweist. Der Benchmark ist ein Auszug aus den Krankheitskatalogen von Yahoo und dem Open Directory Project (ODP).

In Tabelle 10 sind die Gesamtergebnisse des Benchmarks für den UMLS-Metathesaurus dargestellt. Auch hier sind die Ergebnisse des Benchmarks mit ca. 95% zufriedenstellend.

UMLS	Precision		Recall		F-Measure	
	ohne	mit	ohne	mit	ohne	mit
Health-Benchmark	94,6%	94,6%	95,4%	95,9%	95,0%	95,2%

Tabelle 10: Precision, Recall und F-Measure bei dem Health-Benchmark

Dennoch folgt eine genauere Betrachtung der in trivial und nicht-trivial unterteilten Korrespondenzen in Tabelle 11, um die Ergebnisse genauer analysieren und die Problemkorrespondenzen aufdecken zu können.

UMLS / Korrespondenz	Precision		Recall		F-Measure	
	ohne	mit	ohne	mit	ohne	mit
trivial	95,6%	96,1%	98,8%	98,3%	96,8%	97,2%
nicht-trivial	92,5%	87,0%	60,9%	65,8%	76,7%	76,4%

Tabelle 11: Precision, Recall und F-Measure, unterschieden nach Korrespondenztypen

Durch das Clustern der Korrespondenztypen wird, wie bei dem OpenThesaurus, der Recall bei den nicht-trivialen Beziehungen als Hauptproblem identifiziert. Dies wirkt sich auch auf den Wert des F-Measures aus. Das Einbringen von Hintergrundwissen für die medizinische Domäne durch den UMLS-Metathesaurus hebt den Recall geringfügig an, da spezielle Korrespondenzen nicht durch andere Strategien erkannt werden können. Beispielsweise ist es nur mit dem UMLS-Methathesaurus möglich eine „equal“-Beziehung zwischen „Anxiety Disorder“ und „Anxiety“ zu identifizieren. Diese Begriffe beschreiben allgemein eine Angststörung. Dies bestätigt den Einsatz von Hintergrundwissen als Erfolg versprechende Strategie zur Verbesserung eines gegebenen Schema Mappings. Der Rückgang der Precision bei den nicht-trivialen Korrespondenzen ist ein kleiner Wermutstropfen. Da jedoch der Wert der Precision mit 87,0% immer noch sehr hoch ist und die Steigerung des wichtigeren Maßes, dem Recall, nicht unerheblich ist, kann des UMLS-Metathesaurus als zuverlässiger Wissenspeicher genutzt werden.

6 Zusammenfassung

Durch die zunehmende Entwicklung von Systemen jeder Art und deren Automatisierung oder Teilautomatisierung steigt die Heterogenität in Bezug auf deren Schemata. In Folge dessen werden Matcher benötigt, die diese Schemata zuverlässig gegeneinander abgleichen können. Da jedoch einfache String-Matcher, Algorithmen oder Strategien, die die Struktur der Schemata nutzen, beschränkt sind, kommen zusätzliche Wissenspeicher zum Beispiel in Form von Thesauri zum Einsatz. Diese erweitern die Möglichkeiten des Schema Matchers oder können nachträglich auf ein gegebenes Schema Mapping angewandt werden und verbessern so die Qualität.

Dafür wurden im Rahmen der Bachelorarbeit nach der Einführung von grundlegenden Begriffen eine Auswahl an Datenquellen betrachtet und zwei Erfolg versprechenende Datenquellen selektiert und näher untersucht. Die zwei Datenquellen OpenThesaurus und der UMLS-Metathesaurus wurden bei der Implementierung in ihrer Struktur geändert, ohne jedoch den Wissensgehalt zu beeinflussen. Außerdem wurden Anfragen für die vorhandenen Korrespondenztypen implementiert. Es wurden verschiedene Ansätze für jede Datenquelle auf ihre Effizienz geprüft und diejenige Herangehensweise im Mapping-Enrichment-Module integriert, welche in kürzester Zeit die Ergebnisse liefert. Beim OpenThesaurus kam dabei Datenbank zum Einsatz, mit deren Hilfe die Anfragen realisiert werden konnten. Im Falle des UMLS-Metathesaurus wurden die benötigten Daten aus der vorhandenen Datenbank in eine Hashmap überführt und diese als Objektdatei gespeichert. Anfragen sind dann nach Einlesen dieser Datei in kürzester Zeit möglich, da die Daten im Hauptspeicher gehalten werden. Abschließend wurden die erzielten Ergebnisse auf ihre Verbesserung in Bezug auf das Schema Mapping überprüft.

In zukünftiger Arbeit ist es möglich, weitere Datenquellen zu implementieren, die das Ergebnis des Schema Matchings weiter verbessern. Es ist jedoch nicht zu erwarten, dass alle vorhandenen Korrespondenzen zwischen Schemata gefunden werden, ohne dass falsch erkannte Korrespondenzen auftreten. Durch das Hervorbringen weiterer Strategien aus der Forschung oder das Einbinden von mehr Datenquellen mit Hintergrundwissen ist dennoch genug Potential für weitere Arbeiten vorhanden, um das Schema Mapping qualitativ hochwertiger zu machen.

Literatur

[1] University of Amsterdam. *EuroWordNet: Building a multilingual database with wordnets for several European languages.* 2013. URL: http://www.illc.uva.nl/EuroWordNet/.

[2] Peter Christen. *Data Matching.* 2012.

[3] University of Colorado. *Unified Verb Index.* 2013. URL: http://verbs.colorado.edu/verb-index/index.php.

[4] Patrick Arnold und Erhard Rahm. *Semantic Enrichment of Ontology Mappings: A Linguistic-based Approach.* Research Paper. Univerity of Leipzig, 2013.

[5] the Gene Ontology. *An Introduction to the Gene Ontology.* 2013. URL: http://www.geneontology.org/GO.doc.shtml.

[6] Dr. Michael Hartung. *Datenintegration Kapitel 6: Schemamanagement.* 2013. URL: http://dbs.uni-leipzig.de/file/DI-SS2013-Kap6.pdf.

[7] Andreas Ledl. *Thesaurusportal.* 2013. URL: http://thesaurusportal.blogspot.de/.

[8] Database Group Leipzig. *COMA 3.0.* 2013. URL: http://dbs.uni-leipzig.de/de/Research/coma.html.

[9] Web Data Integration Lab Universität Leipzig. *Schema- und Ontologiematching.* 2013. URL: http://wdilab.uni-leipzig.de/index.php?option=com_content&view=article&id=64&Itemid=67&lang=de.

[10] U.S. National Library of Medicine. *UMLS Quick Start Guide.* 2013. URL: http://www.nlm.nih.gov/research/umls/.

[11] Daniel Naber. *OpenThesaurus: ein offenes deutsches Wortnetz.* Paper. Open Source Projekt, 2005.

[12] Daniel Naber. *Statistiken - OpenThesaurus.* 2013. URL: http://www.openthesaurus.de.

[13] Prof. Dr. Felix Naumann. *Schema Mapping.* 2012. URL: http://www.tele-task.de/archive/series/overview/892/.

[14] Roberto Navigli. *BabelNet: a Wide-Coverage Multilingual Ontology.* 2013. URL: http://lcl.uniroma1.it/babelnet/.

[15] Adam Pease. *The Suggested Upper Merged Ontology (SUMO) - Ontology Portal.* 2013. URL: http://www.ontologyportal.org/.

[16] The Trustees of Princeton University. *WordNet a lexical database for English.* 2013. URL: http://wordnet.princeton.edu/.

[17] Uwe Quasthoff. *Information Retrieval Sommersemester 2013 Vorlesungen 3-4.* 2013. URL: http://asv.informatik.uni-leipzig.de/courses/128.

[18] Universität Tübingen. *GermaNet - An Introduction*. 2013. URL: http://www.sfs.uni-tuebingen.de/lsd/.

[19] University of Toronto. *Clio*. 2013. URL: http://queens.db.toronto.edu/project/clio/.

[20] UNESCO. *UNESCO THESAURUS*. 2013. URL: http://databases.unesco.org/thesaurus/.

[21] Felix Naumann Ulf Leser. *Informationsintegration*. 2007.

[22] Grady Ward. *Moby Thesaurus*. 2013. URL: http://thesaurus.babylon.com/moby.

[23] Erhard Rahm Zohra Bellahsene Angela Bonifati. *Schema Matching and Mapping*. 2011.

Abbildungsverzeichnis

Tabellenverzeichnis

Eigenständigkeitserklärung

„Ich versichere, dass ich die vorliegende Arbeit mit dem Thema „Untersuchen von Hintergrundwissen zur Verbesserung von semantischen Mappings“ selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann“.

Ort, Datum	Unterschrift