UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Bachelorarbeit

NUTZUNG VON CONCEPT MAPS ZUR ONTOLOGIE-EVOLUTION

Sören Dohmen

Abteilung Informatik und angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

29. Mai 2012

Betreuer:
Prof. Dr. H. U. Hoppe
Dipl.-Inform. Sabrina Ziebarth
Dipl.-Inform. Stefan Weinbrenner

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Eidesstattliche Versicherung

Ich versichere hiermit ehrenwörtlich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe. Wörtlich übernommene Sätze und Satzteile sind als Zitate belegt, andere Anlehnungen hinsichtlich Aussage und Umfang unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist auch noch nicht veröffentlicht.

_____

Ort, Datum, Unterschrift

1. Einleitung

Gute und das bedeutet insbesondere auch immer wieder aktualisierte und somit gepflegte Ontologien sind nützlich, um Daten unter heterogenen Systemen austauschen und wiedernutzen zu können. Deswegen ist es sinnvoll, den Entwicklungs- und Pflegeprozess von Ontologien durch Automatisierung zu verbessern. Eine Ontologie, deren Wissen veraltet oder unvollständig ist, führt zu Anwendungen, die veraltetes, unvollständiges Wissen verwenden und daher ungewünschte Ergebnisse liefern. Pflege und Wartung von Ontologien ist also notwendig, aber kostenintensiver, wenn sie nur von Hand vorgenommen wird. In dieser Arbeit wird deshalb ein Verfahren zur semiautomatischen Erweiterung von Ontologien durch Concept Maps entwickelt.

1.1 Motivation

Concept Maps sind ein weitverbreitetes Mittel zur strukturierten Modellierung von Wissen. Sie werden beispielsweise in schulischen Lehr-Lern-Situationen zur Wissensevaluation eingesetzt. [1] Die Verbreitung der Concept Mapping Technik als Lern-, Strukturierungs-, und Diagnoseinstrument in den verschiedensten Disziplinen sorgt dafür, dass diese Technik vielen Menschen vertraut ist. Sie sind bereits in Schule oder Studium mit ihr in Kontakt gekommen.

Lernende können selbstständig in Einzel-, Partner-, oder Gruppenarbeit Concept Maps erstellen und damit ihrem Verständnis der Zusammenhänge eines bestimmten Themengebiets (einer Domäne) Ausdruck verleihen. Mittels Concept Maps können graphische Repräsentationen von strukturiertem Wissen erstellt werden. Sie bestehen aus Begriffen, sogenannten Konzepten, welche durch beschriftete Kanten, die Relationen zwischen den Konzepten darstellen, miteinander verbunden sind.

Concept Maps gehen also über eine rein textuelle Repräsentation des Wissens ihres Themengebietes, beispielsweise in Webseiten oder Büchern, weit hinaus, da sie dem Wissen eine Struktur geben. Dadurch ist nicht mehr nur der Mensch als direkter Konsument des Wissens denkbar, sondern durchaus auch Computer(programme), die ihre Struktur entsprechend interpretieren können.

Ontologien stellen ähnlich wie Concept Maps eine strukturierte Formalisierung von Wissen dar. Bei Ontologien kann dieses Wissen, wie bei Concept Maps, mit Fokus auf eine bestimmte Domäne repräsentiert sein. Entwickler von Ontologien begrenzen tendenziell einen Zusammenhang und treffen eine Entscheidung, welches Wissen zu einer Domäne gehört und welches nicht zu einer Domäne gehört. Entscheidend ist darüber hinaus, dass Ontologien aus einer maschineninterpretierbaren Formalisierung, von allgemein akzeptiertem und daher von einer Gruppe von Menschen getragenem Wissen über bestimmte Begriffe, bestehen. Die Begriffe können zueinander in Beziehung gesetzt sein. Des Weiteren können auch Einschränkungen auf die Begriffe definiert sein.

Wegen ihrer den Ontologien ähnlichen Struktur stellen Concept Maps eine vielversprechende Basis zur Extraktion von Wissen zur individuellen Unterstützung von sogenannten „Knowledge Engineers“ (KE) dar. Ein KE ist ein Experte bezüglich Ontologieentwurf und Ontologiewartung.

Ein wesentliches Problem bei der Entwicklung und Wartung von Ontologien ist, dass jemand, der sich zwar mit den Methoden zum Erstellen einer Ontologie auskennt, oft nicht im selben Maße mit der zu repräsentierenden Domäne (z. B. Chemie) vertraut ist. Domänenexperten (z. B. Chemiker, s. o.) wiederum kennen zwar ihre jeweilige Domäne sehr gut, sind aber mit der Vorgehensweise des Ontology Engineering nicht vertraut. Ziel der in dieser Bachelorarbeit entwickelten Methode ist es, ebenfalls einem Domain Expert, der sich mit der Wartung und Erstellung von Ontologien, insbesondere in technischer Hinsicht, nicht auskennen muss, sich aber sehr gut mit dem die Domäne betreffenden Wissen auskennt, zu befähigen, die Ontologie zu erweitern. Das wird dadurch ermöglicht, dass dem Experten am Ende des Wissensextraktionsprozesses nur noch die Entscheidung über Sinnhaftigkeit oder Sinnlosigkeit der ihm zur Überführung in die Ontologie gemachten Vorschläge obliegt. Diese Entscheidung benötigt keine technischen Kenntnisse, sondern das Sachverständnis eines Fachmanns.

Das Entwickeln von Ontologien von Hand aus einer großen Datenmenge, sowie die Pflege und Erweiterung einer Ontologie ist sehr arbeitszeitintensiv [2] und in seiner Komplexität mit der Softwareentwicklung vergleichbar, wie die Gegenüberstellung der Lifecycle-modelle in Abbildung 4 zeigt. Daher ist es sinnvoll, wenn der Informationsgewinnungsprozess zum Teil oder als Ganzes automatisiert wird. Das gewünschte Ergebnis ist die Verfeinerung des in der Ontologie gespeicherten Wissens. Concept Maps können Fehler, sowohl hinsichtlich der sprachlichen Ausformulierung der Konzepte, als auch hinsichtlich der in ihnen gewählten Struktur enthalten. Weil das Ziel einer Verfeinerung eine hohe Precision ist, eignen Concept Maps sich nicht zur vollautomatischen Wissensextraktion, weswegen die, durch das in Ihnen repräsentierte Wissen gewonnenen, Vorschläge von einem Experten verifiziert werden müssen. Durch jeden Grad an Automatisierung kann Zeit und Geld gespart werden, welches in die weitere Verbesserung der Ontologie investiert werden kann.

1.2 Aufgabenstellung

Die Aufgabenstellung umfasst die Überführung einer Menge von Concept Maps in ein gemeinsames statistisches Modell. Dazu werden, durch entsprechend angewandte Retrievalmechanismen, ähnliche Konzepte innerhalb einer Menge von Concept Maps bestimmt. Um die Ähnlichkeit zu ermitteln, werden Synonymlexika abgefragt und Stammformreduktion, wie in Abschnitt 2.8 beschrieben, durchgeführt, sowie die lexikalische Ähnlichkeit anhand der Edit- Distance bzw. der Levenshteindistanz (siehe Abschnitt 2.6) ermittelt. Des Weiteren werden gewichtete Kanten zwischen Konzepten entsprechend ihrer Auftretensfrequenz bestimmt.

Das so erstellte Modell wird mit einer gegebenen Ontologie derselben Domäne verglichen und aus dem Vergleich werden Ergänzungsvorschläge für die Ontologie abgeleitet. So werden neue Konzepte, Synonyme für vorhandene Konzepte, sowie neue Relationen zum Vorschlagen

ermittelt. Diese Vorschläge werden dem Knowledge Engineer (KE) bzw. einem Domänenexperten (DE) in einem interaktiven Benutzer-Interface zur Abarbeitung vorgestellt werden. Im Rahmen der von ihm angenommenen Vorschläge wird die Ontologie entsprechend um die auf diese Weise bestätigten Vorschläge erweitert.

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in fünf Kapitel. Der Einleitung folgt das Kapitel Grundlagen, in dem Hintergrundinformationen zu den zentralen Bausteinen der Arbeit, Ontologien und Concept Maps dargelegt werden. Des Weiteren wird die verwendete Middleware – die TupleSpaces – eingeführt. Auch wird ein kurzer Einstieg in n-Gramme, Stoppwörter, Lemmatisierung und Stemming gegeben, also die Techniken des Information Retrieval, die zur Lösung der Aufgaben dieser Bachelorarbeit beitragen. Das Konzept beschreibt dann die Grundproblematik und die in der späteren Implementationsphase umgesetzten Lösungsansätze.
Am Ende der Arbeit werden die wesentlichen Ergebnisse zusammengefasst sowie ein Ausblick auf Weiterentwicklungsmöglichkeiten des Themas gegeben.

2. Grundlagen

In diesem Kapitel werden die zum Verständnis der in dieser Bachelorarbeit behandelten Problemstellung benötigten Grundlagen eingeführt. RDF(S) und OWL werden kurz eingeführt, weil es sich um die in der für diese Bachelorarbeit gegebenen Ontologie verwendete Syntax handelt.

2.1 RDF(S)

Das Resource Description Framework (RDF) ist eine W3C-Empfehlung. [3] Es wurde entworfen, um Informationen auf einem wenig einschränkenden, flexiblen Weg auszudrücken. Das dem RDF zugrunde liegende Datenmodell ist eine Sammlung von Tripeln. Diese bestehen aus den Elementen Subjekt, Prädikat und Objekt und machen, wenn sie miteinander in Verbindung gebracht werden, eine Aussage über eine Ressource, die folgenden Zusammenhang beschreibt:

Subjekt hat die Eigenschaft *Prädikat* mit dem Wert *Objekt [4]*, oder, etwas trivialer ausgedrückt, *Objekt* ist das *Prädikat* von *Subjekt*.

Gleichzeitig beschreiben diese Tripel einen Graphen, da Beziehungen zwischen Subjekt und Objekt ausgedrückt werden. Subjekt und Objekt sind gleichzeitig die Knoten des Graphen. Die Kante wird durch das Prädikat definiert. Jeder Teil dieses Tripels - oder auch Satzes - wird durch einen Uniform Ressource Identifier (URI) bezeichnet, um Verwechslungen zu verhindern, die daraus resultieren könnten, dass zwei Ressourcen denselben Bezeichner besitzen.

Die Syntax lautet: URI = Schema://Anbieter/Pfad?Anfrage#Fragment

<#740088> <#name> "Sören"
<#bachelorarbeit> <#name> "Bachelorarbeit" <#740088> <#schreibt> <#bachelorarbeit>

Tabelle 1: RDF Tripel in der von TIM BERNERS-LEE vorgeschlagenen Notation 3

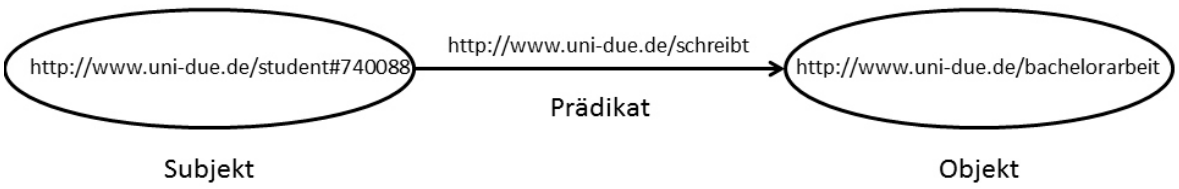

Abbildung 1: RDF Tripel als gerichteter Graph

Der einzige Teil eines solchen Tripels, der nicht durch eine URI bezeichnet sein muss, ist das Objekt. Dieses darf eine Zeichenkette oder eine Ganzzahl sein, ein sogenanntes Literal. Damit ist ein Literal ein Datenwert. Einem solchen Datenwert kann auch ein Datentyp zugewiesen werden. Hierzu empfiehlt das W3C für RDF die Verwendung der Datentypen von XML Schema, da es sich

bei den dort definierten Datentypen um wohlbekannte und in vielen Programmiersprachen implementierte Datentypen, wie *string* oder *integer* handelt.

Bei *740088* und dem klein geschriebenen *„bachelorarbeit“* handelt es sich lediglich um einen Bezeichner. Um dem Bezeichner einen Sinn zu geben, muss das Prädikat *„Name“* definiert werden. So hat nach obigem Beispiel also der Bezeichner *„740088“* den Namen *„Sören“*.

Um den RDF-Graphen auszudrücken, existieren verschiedene Sprachen. Weit verbreitet ist die Repräsentation in XML [5] und die von TIM BERNERS-LEE vorgeschlagene Notation 3 (N3) [6].

Mit RDF sind Aussagen über Individuen, das sind einzelne Resources und deren Beziehungen, möglich. Wenn darüber hinausgehende Aussagen über allgemeine Mengen von Individuen, also Klassen, gemacht werden sollen, so kann dies mit Hilfe von RDF Schema (RDFS) geschehen.

Mit RDFS können außerdem Aussagen über die semantischen Beziehungen der Ressourcen getroffen werden. [7] Hierzu existiert in RDFS beispielsweise die Möglichkeit zum Typisieren von Klassenbezeichnern als rdfs:Class, um eindeutig klarstellen zu können, dass es sich bei diesem Klassenbezeichner um eine Klasse und nicht um ein Individuum handelt. Auch existieren Möglichkeiten zur Unterklassenbeziehungsangabe (rdfs:subClassOf) und zur Untereigenschaftenangabe (rdfs:subPropertyOf) sowie der Einschränkungen des Definitionsbereichs (rdfs:domain) und des Wertebereichs (rdfs:range). So ist es also bereits mit Hilfe von RDFS möglich, einfache sogenannte „lightweight“ Ontologien zu formalisieren, die die Ableitung von implizitem Wissen gestatten. [8]

2.2 OWL

OWL ist ein auch im Hinblick auf die sprichwörtlich weise Eule gewähltes leicht modifiziertes Akronym, das für Web Ontology Language steht. [9] Sie ist die aus DAML+OIL hervorgegangene W3C Empfehlung zum Beschreiben einer Ontologie. Seit Ende 2009 liegt OWL in der Version 2 als W3C Empfehlung vor. [10] Diese bietet neue Features und die Nutzung von anderen Syntaxen an. Da die gegebene Ontologie diese nicht nutzt und OWL 2 volle Abwärtskompatibilität garantiert [10], wird an dieser Stelle OWL 1 beschrieben.

Es gibt drei verschieden ausdrucksstarke Dialekte von OWL. So kann der Anwender sich je nach Praxisfall für die in ihrer Ausdrucksstärke passendste Teilsprache entscheiden. Ein architektonischer Vorteil ist, dass jede der ausdrucksschwächeren Teilsprachen eine echte Teilsprache der jeweils ausdrucksstärkeren Sprache ist. So gibt es OWL Lite, OWL DL (DL bedeutet Description Logic) und OWL Full. Jedes die RDF-Syntax nutzende OWL-Dokument ist ein RDF-Dokument, aber nicht jedes RDF-Dokument ist OWL. [9] Pro OWL-Teilsprache ist die Benutzung entsprechender RDFS Features vorgesehen. OWL Lite lässt zusätzlich Restriktionen und Charakteristikadefinitionen auf die Relationen (rdf:Property) zu. Ebenso können Schnittmengen von Klassen gebildet werden und es gibt Versionsinformationen. OWL Lite ist die am wenigsten ausdrucksfähige Teilsprache, dafür aber entscheidbar und in einer geringeren Komplexitätsklasse als OWL DL, das jedoch ebenfalls noch entscheidbar ist. OWL Full, das die beiden anderen

Teilsprachen enthält, ist unentscheidbar. Es wird zwar in voller Komplexität auf Editorebene unterstützt, lässt aber wegen der Unentscheidbarkeit kein volles Reasoning zu. In OWL Full und OWL DL werden zusätzlich Klassenaxiome, echte Kardinalitäten, boolsche Kombinationen von Klassenausdrücken sowie „hasValue“ definiert. Mit Hilfe von „hasValue“ kann ein fester Wert für eine Property festgelegt werden. Dies bedeutet, dass alle Instanzen, die diesen Wert für die Property besitzen, auch automatisch Mitglieder der Klasse sind, in der „hasValue“ definiert wurde. Letztlich geht es in OWL darum, zusätzlich zum definierten Vokabular, Möglichkeiten zum logischen Schließen und damit der Inferenz zu eröffnen.

Alle OWL-Dokumente bestehen aus dem Kopf, in dem allgemeine Angaben wie XML-Namespaces definiert sind und Kommentare sowie Versionierungsinformationen angegeben werden können und einem Restdokument, in dem die eigentliche Ontologie beschrieben wird. [9] Die XML-Namespaces sind dabei das Mittel zum eindeutigen Definieren des verwendeten Vokabulars des betreffenden Dokuments. [11]

2.3 Ontologien

Eine in der Informatik häufig auftretende Aufgabe ist es, Wissen darzustellen, Wissen zu kommunizieren und Wissen zu teilen. Menschen haben eine Allgemeinbildung, Grundwissen, Hintergrundwissen und Kontextwissen. Menschen können Bücher lesen, Begriffe in Schlagwortsammlungen oder Lexika nachsehen oder andere Menschen fragen, um ihr Wissen zu erweitern und Zusammenhänge zwischen einzelnen Gegenstandsbereichen auf unterschiedlichen Ebenen und aus unterschiedlicher Perspektive herstellen. So ist beispielsweise der Begriff „Buch“ vom Begriff „Film“ klar unterschieden, beide hängen aber auch über den Begriff „Medien“ zusammen.

Computer können nicht ohne Weiteres einen solchen Zusammenhang wie im geschilderten Beispiel herstellen. Computer können auch nicht die Bedeutung hinter einem solchen Begriff erfassen. Sie benötigen Informationen darüber, welche Begriffe zusammenhängen und wie diese Begriffe zusammenhängen. Die Klärung dieser Fragen kann neben vielen weiteren semantischen Modellen zur Wissensrepräsentation, wie Katalogen, Thesauri und Glossaren, auch formale Ontologien (siehe Abschnitt 2.3.2) entstehen lassen. Die Ontologie ist dabei sowohl das Vokabular als auch die Definition der Bedeutung des Vokabulars. Eine Ontologie führt eine standardisierte Terminologie sowie Beziehungen und Ableitungsregeln ein.

Bereits Gottfried Wilhelm Leibnitz hat vor langer Zeit erkannt, dass „d(D)ie Kunst, die Dinge in Geschlechter und Arten zu ordnen, (…) von nicht geringer Bedeutung (ist) und (…) sowohl dem Urteil als dem Gedächtnis erheblich (dient). (…) Ein guter Teil der Ordnung hängt davon ab (…) die nicht nur dazu dient, die Dinge zu behalten, sondern sie sogar zu finden.“ [12]

Auch in der theoretischen Philosophie wird sich darum bemüht mit der Ontologie, der Lehre von allem Seienden (gr. όν, seiend und λόγος, Lehre, Wort), alles Seiende (insbesondere Gott) mit den Mitteln der Sprache zu erfassen. [13] Klarheit über die Strukturen des Seienden zu gewinnen ist in

der Informatik insofern von Bedeutung, als es der Ontologie in der Informatik um die formalsprachlich ausgedrückte und formal geordnete Darstellung einer Sammlung von Begrifflichkeiten und der zwischen den Elementen dieser Sammlung bestehenden Beziehungen, geht. [14]

THOMAS GRUBER definiert den Begriff Ontologie als „formale, explizite Spezifikation einer gemeinsamen Konzeptualisierung". [14] Konzeptualisierung bedeutet in diesem Zusammenhang „eine abstrakte, vereinfachte Sicht der Welt" [14], die repräsentiert werden soll. Sie bedingt, dass die Konzepte, die Ergebnis der Konzeptualisierung sind, unabhängig von den verwendeten Begriffen sein sollen. „Globale Erwärmung" und „global warming" sind dieselbe Konzeptionalisierung, jedoch in zwei verschiedenen Sprachen. Gemeinsam bedeutet hier, dass eine Gruppe von Menschen (z. B. Wissenschaftler) sich auf ein gemeinsames Verständnis eines Gegenstandsbereichs einigt. „People can't share knowledge if they don't speak a common language."[15] Die explizite formale Spezifikation wird dadurch erreicht, dass die Typen der Konzepte sowie Constraints bzw. Relationen definiert werden, also präzise und maschineninterpretierbar beschrieben werden. Dies ermöglicht, dass die Ontologie in der Folge auch vom Computer interpretiert und verarbeitet werden kann.

Eine Ontologie ist also eine formale und präzise Beschreibung einer abstrakten, vereinfachten Sicht auf einen Ausschnitt der Welt. Ontologien in der Informatik beschäftigen sich folglich immer nur mit einem Ausschnitt der Welt und nie mit der ganzen Welt. Die ganze Welt und deren Sachverhalte ontologisch zu erfassen wäre eine zu komplexe Aufgabe. Es wäre auch nicht möglich, da es immer verschiedene Sichten gibt und eine Einigung auf eine gemeinsame Konzeptualisierung der gesamten Welt schlicht unrealistisch ist. Dies ist für Anwendungsfälle auch nicht nötig, da es ausreicht, die für die zu erledigende Aufgabe nötigen Ontologie(n) zu erstellen oder zu befragen und gemeinschaftlich problemorientiert zu nutzen.

2.3.1 Repräsentative Primitive

So besteht eine Ontologie im Wesentlichen aus „eine(r) Menge von repräsentativen Primitiven mit welchen man eine Wissensdomäne modelliert." [16] Die repräsentativen Primitiven sind die Konzepte, Instanzen, Attribute und Relationen. Ihre Definition beinhaltet Informationen über ihre Bedeutung und Constraints bezüglich ihrer logisch konsistenten Anwendung. Konzepte werden auch Klassen oder Begriffe genannt, so wie Instanzen auch als Individuen oder Objekte bezeichnet werden. Für Relationen ist ebenfalls der Begriff Rollen gebräuchlich. Attribute stellen eine spezielle Form der Relation dar. Des Weiteren existieren noch Axiome, die Regeln innerhalb der Domäne festlegen.

Konzepte sind eine abstrakte Beschreibung einer Menge von Individuen mit gleichen Eigenschaften. Die Attribute des Konzeptes Person könnten beispielsweise Geschlecht, Fingerabdruck, Beruf, Geburtsdatum, Geburtsort, Augenfarbe, Haarfarbe und Gewicht sein.

Diese Bachelorarbeit könnte beispielsweise eine Instanz des Konzeptes Bachelorarbeit sein. Das Vorgenannte ist gleichzeitig eine Relation in der Ontologie, nämlich diejenige, die eine Instanz einem Konzept zuordnet. Instanzen können auch mehreren Konzepten zugeordnet sein. Des Weiteren können Konzepte auch Unterkonzepte von anderen Konzepten sein. So ist es möglich, das Konzept „Bachelorarbeit“ als Unterkonzept von „wissenschaftlicher Text“ zu betrachten. „Wissenschaftlicher Text“ kann dann noch viele weitere Unterkonzepte besitzen. Konzepte und Instanzen können Eigenschaften in Form von Attributen haben. Ein Beispiel dafür ist die Seitenzahl des Konzeptes Bachelorarbeit: „zwischen 40 und 50“ und die Seitenzahl dieser speziellen Instanz: „45“. So lassen sich Regeln und auch Constraints definieren, die für bestimmte Konzepte, deren Instanzen und den Relationen zwischen ihnen gelten müssen.

Eine Ontologie bildet ein semantisches Netz, welches sich grafisch wie in Abbildung 2 visualisieren lässt.

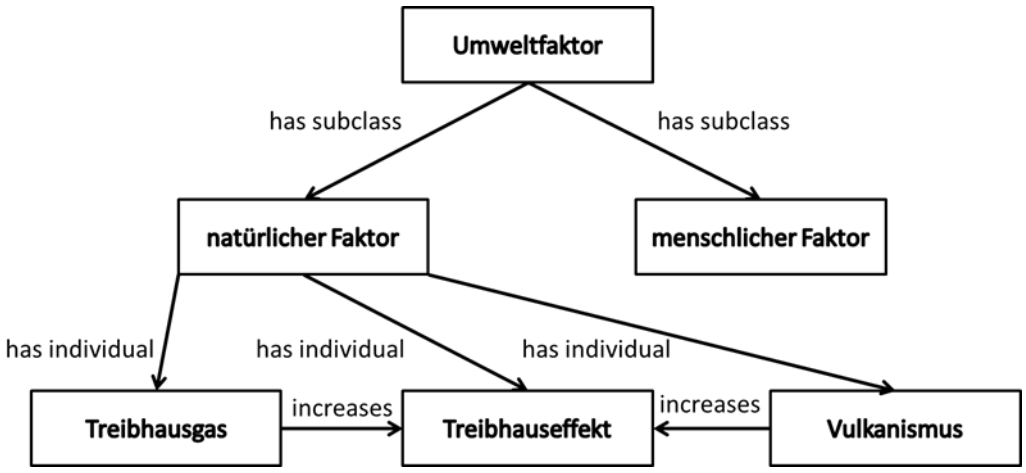

Abbildung 2: vereinfachter Teilausschnitt einer Ontologie

2.3.2 Arten von Ontologien

Ontologien lassen sich entweder aufgrund der Reichhaltigkeit ihrer internen Struktur typisieren (siehe Abbildung 3: Abgrenzung von Ontologien untereinander nach McGuinness [17]), oder basierend auf dem Gegenstand ihrer Konzeptualisierung. [18] Basierend auf dem Gegenstand ihrer Konzeptualisierung lassen sich laut Guarino vier Haupttypen herausstellen. [19] **Top Level Ontologien** beschäftigen sich mit größeren, weltumfassenden, mindestens bereichsübergreifenden Begriffen. Guarino nennt hierfür die Beispiele Raum, Zeit, Materie, Objekt, Ereignis sowie Aktion, solange sie unabhängig von einem speziellen Problem oder einer speziellen Domäne betrachtet werden. Guarino führt weiter aus, dass es dann sinnvoll sein kann, für solche Konzepte vereinheitlichte Top Level Ontologien zu definieren, die dann einem großen Benutzerkreis zugänglich sind. [19] **Domänenontologien** und **aufgabenbezogene Ontologien** stellen das Vokabular für eine bestimmte Domäne oder Aufgabe zur Verfügung. **Anwendungsbezogene Ontologien** stellen Spezialisierungen und Vereinigungen von Domänen-

und aufgabenbezogenen Ontologien dar. Diese Letzteren machen häufig Aussagen über die Rollen der Dinge möglich, die durch die Ontologie repräsentiert werden. [19]

Basierend auf der Komplexität ihrer inneren Struktur werden zwei große Gruppen von Ontologien unterschieden. „Lightweight"-Ontologien wie Glossare und Thesauri definieren ein Vokabular und sogar Abhängigkeiten der Konzepte untereinander. „Heavyweight" - Ontologien, die auch formale Ontologien genannt werden, ermöglichen zusätzlich zu Struktur und formalem Vokabular beispielsweise auch das Einführen von Einschränkungen und das logische Schließen. [18] Die konkrete Ontologie, mit der sich diese Arbeit beschäftigt, fällt in die Kategorie der formalen Ontologien. Das bedeutet selbstverständlich auch, dass, wie von GRUBER gefordert, eine formale Spezifikation der durch die Ontologie repräsentierten Dinge möglich ist. Es bedeutet durchaus nicht, dass die in der Abbildung 2 ausgedrückten Glossare oder Thesauri nicht auch Teil einer formalen Ontologie sein können. Diese Techniken sind nur selbst nicht mächtig genug, um eine Ontologie im formalen Sinne auszudrücken. Alle in der Abbildung nicht grau hinterlegten Ontologien sind sogenannte „Lightweight"-Ontologien und die grau hinterlegten sind „Heavyweight"-Ontologien.

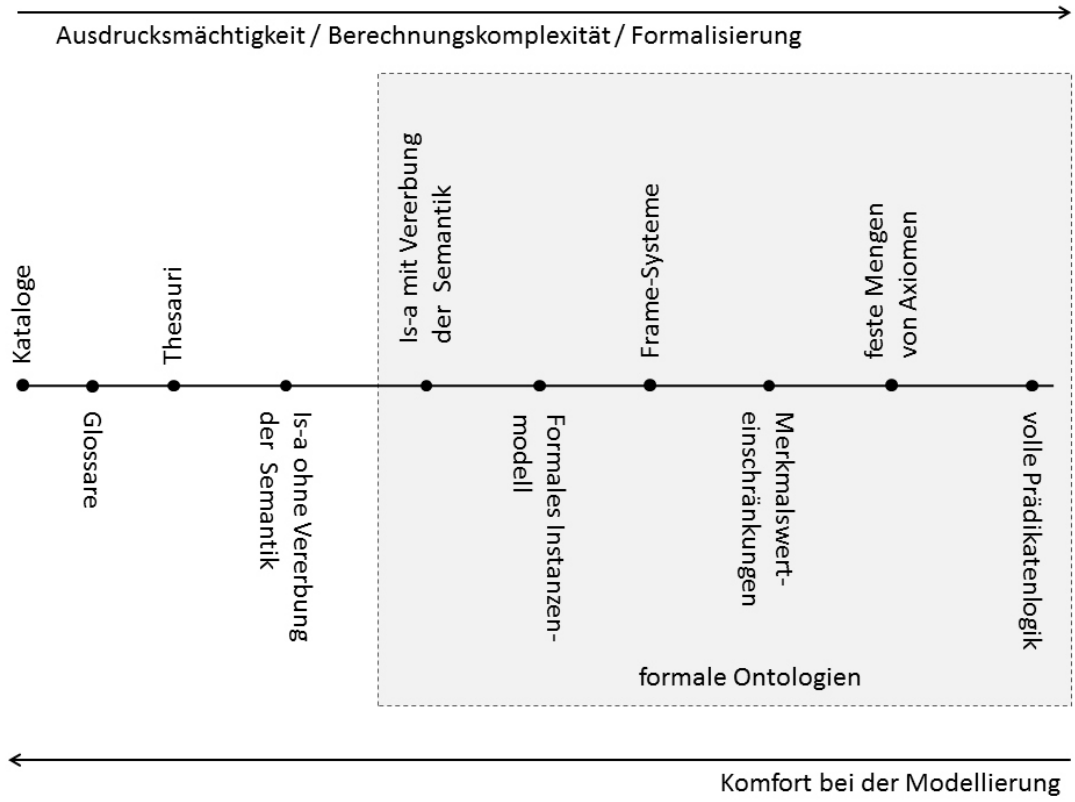

Abbildung 3: Abgrenzung von Ontologien untereinander nach MCGUINNESS [17]

Es gibt drei große Anwendungsfelder für Ontologien. [20] Erstens unterstützen Ontologien die **Kommunikation** zwischen Personen, Organisationen, Prozessen, Systemen und Agenten. Sie tun

dies durch einheitliche Konzeptionalisierung und eine einheitliche Sprache. Auch können sie eine Art Mediatorfunktion bei heterogenen Systemen übernehmen und dadurch die Interoperabilität und den Wissensaustausch bzw. die Wiederverwendung von Wissen fördern. Ontologien ermöglichen **automatisches Schließen** und die **Repräsentation und Wiederverwendung von Wissen**. Überall, wo sich Informatik mit Wissen im weitesten Sinne beschäftigt, lässt sich auch eine Bedeutung für Ontologien ableiten. Mit der Hilfe von Ontologien können Daten nämlich „exportiert, übersetzt, abgefragt, und verbunden werden, zwischen unabhängig voneinander entwickelten Systemen und Diensten“. [16] So werden Ontologien in Gebieten wie der künstlichen Intelligenz, Datenbanken und Informationssystemen, im Bereich des Wissensmanagement, in Multiagentensystemen, in Internetanwendungen, in digitalen Bibliotheken, beim Information Retrieval und der Verarbeitung natürlicher gesprochener Sprache eingesetzt. [21] Es muss geklärt sein, wie vorgegangen wird, welche Designkriterien, Methoden und Werkzeuge genutzt werden, um sie zu erstellen.

2.3.3 Ontologie-Lebenszyklus

Der Entwicklungsprozess einer Ontologie ist mit dem Entwicklungsprozess von Software vergleichbar, wie die Gegenüberstellung zweier einfacher Wasserfallmodelle in Abbildung 4 verdeutlicht. Die schwarzen Doppelpfeile markieren eine Ähnlichkeit der Phasen.

Der Entwicklungsprozess einer Ontologie beginnt mit einer Spezifikationsphase. Während dieser initialen Phase wird versucht zu klären, warum und für welchen Zweck die Ontologie entwickelt wird, sowie von wem sie letztendlich benutzt werden wird. Dazu werden die Anforderungen an die Ontologie analysiert und in einem entsprechenden Anforderungsspezifikationsdokument auf der Basis einer natürlichen Sprache festgehalten. [22]

An die Spezifikationsphase schließt sich die Designphase bzw. Konzeptionalisierung an. Dies ist der eigentliche Modellierungsprozess. Es wird ein konzeptionelles Modell beschrieben, dass die Ontologie in der Art ausdrückt, dass sie den in der vorangegangenen Spezifikationsphase ermittelten Spezifikationen entspricht. Dies kann auf verschiedene Weise geschehen. Die Methoden reichen von sehr informellen Methoden, wie z. B. Mind Maps, bis hin zu formalen Methoden wie Taxonomien. [22]

Hierdurch entsteht ein konzeptionelles Modell der Ontologie, welches die Modellierungsprimitive umsetzt. Es besteht bereits aus den Konzepten der Domäne und den Beziehungen zwischen diesen Konzepten. Unter Umständen wird bei diesem Schritt festgestellt, dass bestimmte Gruppen von Konzepten in einer großen Anzahl von Relationen vorkommen. In einem solchen Fall muss erwogen werden, die Ontologie in eine oder mehrere Unterontologien aufzuspalten, um die Übersichtlichkeit zu wahren. [22]

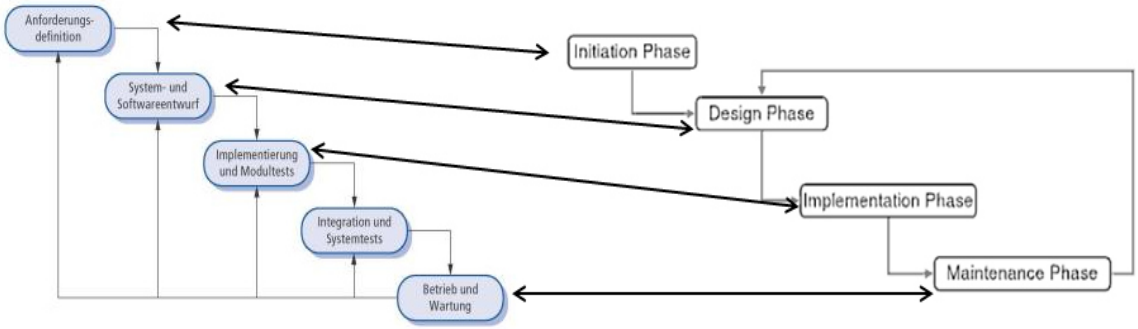

Abbildung 4: Gegenüberstellung der Wasserfallmodelle der Softwareentwicklung (links) [23] und der Ontologieentwicklung (rechts) [24]

Die Formalisierung einer Ontologie beinhaltet die Transformation des konzeptuellen Modells in ein formales Modell. Hierbei handelt es sich jedoch, je nach Fall des Entwicklungsprozesses, noch nicht um die endgültige Fassung der Ontologie. [24, 25] So unterscheidet PINTO [25] einen Formalisierungs-, sowie einen Implementationsschritt. Wohingegen FERNANDEZ [26] zwischen einem Formalisierungs-, einem Integrations- und einem Implementationsschritt unterscheidet. Letztlich sind diese Unterscheidungen eine Aufgliederung des Formalisierungsprozesses, dessen Ziel es ist, eine maschinenlesbare Version der Konzeptionalisierung zu erhalten. Hierzu werden die Wertebereiche der Konzepte festgelegt. Des Weiteren werden die Konzepte hierarchisch strukturiert. Dies geschieht mittels einer strukturierenden Relation wie „is-a“ oder „is-subclass“ oder „part-of“.

In der Implementationsphase wird die so formalisierte Ontologie mittels einer Ontologie-Spezifikations-Sprache [27] implementiert. Bei einer solchen Ontologiesprache handelt es sich um ein formales System zur Beschreibung von Ontologien. [28] Eine solche wurde mit OWL bereits vorgestellt.

Nach der Implementationsphase folgt die Evaluierungs- und Wartungsphase, während derer geklärt wird, wie gut die Ontologie den Anforderungen entspricht. Danach kann die Ontologie genutzt werden. Sie muss aber kontinuierlich mit neuem Wissen erweitert werden und eventuellen neuen Tatsachen angepasst werden. [26] Diese Wartungsarbeiten können je nach Aufwand bedeuten, dass nur ein Teil oder die gesamte Prozesskette neu durchlaufen werden muss.

Um ebendiesen Prozess der Ontologiewartung und die Verbesserung der Ontologie durch Ontology Learning geht es in dieser Arbeit. Daher werden die Begriffe im Kapitel Konzept (siehe Abschnitt 3.1) näher beschrieben und die konzeptuellen Bestandteile dieser Arbeit kontextualisiert.

2.4 Concept Maps

Concept Maps wurden 1972 von JOSEPH D. NOVAK entwickelt. Sie sind ein Werkzeug zum graphischen Darstellen und Organisieren von Wissen. [29] Sie bestehen aus Konzepten und Beziehungen zwischen Konzepten. Ein Konzept wird von NOVAK definiert als „a perceived regularity in events or objects, or records of events or objects".

Die genaue Übertragung des Begriffs „Concept Map" ins Deutsche wäre „Begriffslandkarte", ein Begriff, der metaphorisch gut beschreibt, worum es bei Concept Maps geht: Landkarten modellieren Wissen über Gebiete. Concept Maps modellieren Wissen über Begriffe.

In der deutschsprachigen Literatur zum Thema hat sich allerdings der Begriff „Begriffsnetz" als Übersetzung von „Concept Maps" durchgesetzt [1], der teilweise allerdings auch in einer etwas anderen, erweiternden Bedeutung gebraucht wird und damit eine andere Form von Concept Maps beschreiben kann, als die ursprünglich von NOVAK definierte, da klassische Begriffsnetze in aller Regel nicht wie Concept Maps hierarchisch strukturiert sind. [1] In dieser Arbeit werden die Begriffe *Concept Map* und *Begriffsnetz* synonym verwendet, weil die zugrunde liegenden Concept Maps auch nicht hierarchisch strukturiert sind.

Konzepte werden durch einen Kreis oder ein Rechteck repräsentiert. Ein solches grafisch dargestelltes Konzept kann ein Label besitzen, das aus einem oder mehreren Wörtern oder einem Symbol bestehen darf. Eine Beziehung zwischen zwei Konzepten wird Relation genannt und wird durch eine verbindende Linie zwischen zwei Konzepten dargestellt. Zwei oder mehr Konzepte, die mit solchen Linien verbunden sind, können eine Proposition darstellen, wenn sie eine Aussage machen. Eine Proposition, die aus zwei Konzepten und deren Relation besteht, wie „Bachelorarbeit" „ist ein" „wissenschaftlicher Text" ist gleichzeitig die kleinste Form der Concept Map.

Concept Maps sind nach der Definition von NOVAK eigentlich hierarchisch aufgebaut. [29] Die Hierarchie wird dadurch erreicht, dass die Konzepte, die am allgemeinsten sind, in der Concept Map oben stehen. Konzepte, die spezieller sind, werden dann weiter unten in der Concept Map notiert. Die hierarchische Anordnung der Konzepte kann sich mit dem Kontext ändern. Wenn eine Concept Map sich beispielsweise mit dem Thema KFZ beschäftigt, wird PKW sicherlich als zentrales Konzept in einer höheren Ebene stehen, als in einer Concept Map zum Thema Fortbewegung.

Die Concept Map in Abbildung 5 beschreibt was Concept Maps sind, woraus sie bestehen, wovon sie abhängen und stellt auch dar, dass es mit ihnen möglich ist, Zusammenhänge zu erkennen.

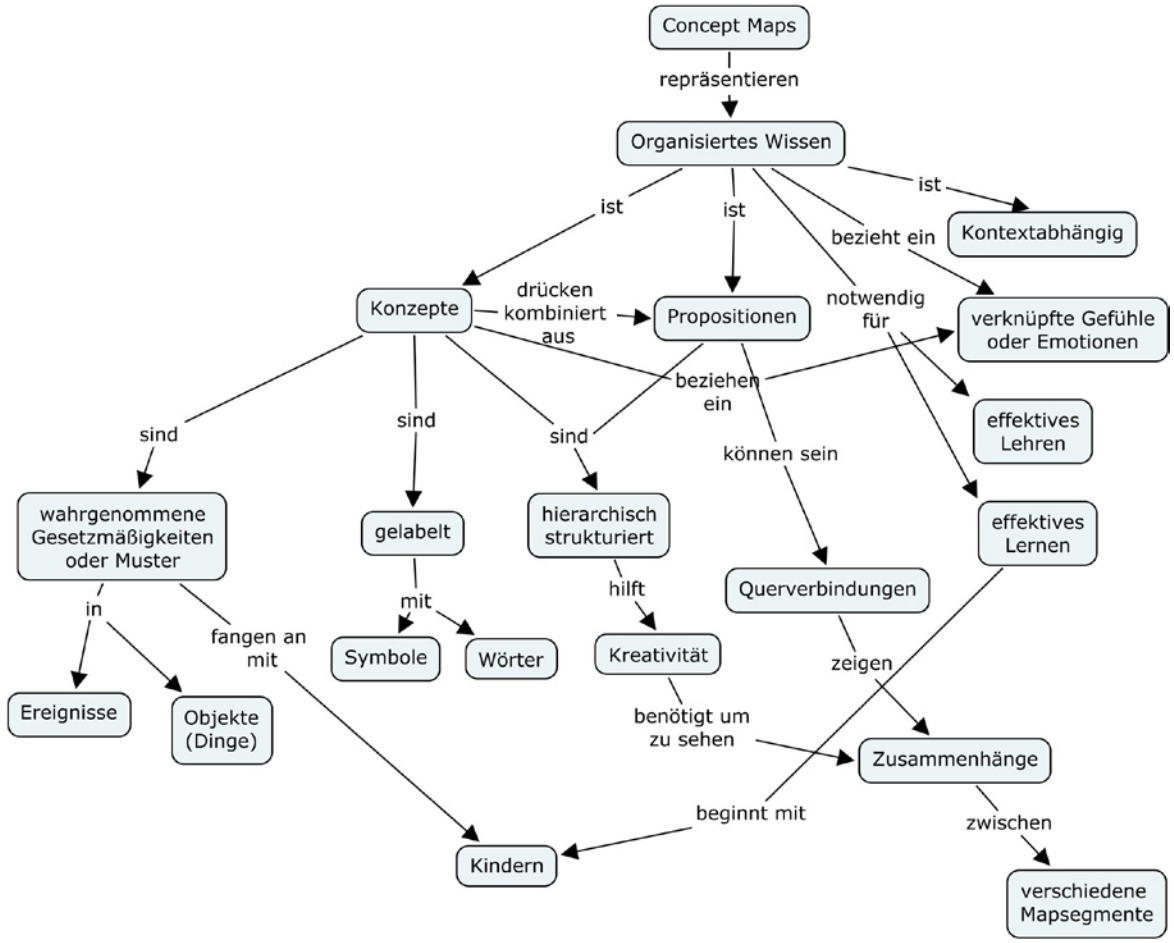

Abbildung 5: Concept Map über Concept Maps nach NOVAK [29]

2.5 TupleSpaces

Die in TupleSpaces umgesetzte Black-Board-Architektur eignet sich als Grundlage für ein Multiagentensystem, bei dem einzelne, in sich abgeschlossene Softwareeinheiten, die daher auch in unterschiedlichen Sprachen verfasst sein können, unabhängig voneinander, also autonom, (Teil-)Probleme lösen. [30] Dabei sind TupleSpaces sowohl Datenspeicher als auch Medium für die Interagentenkommunikation. Jeder Agent darf zu jeder Zeit Daten in Form von Tupeln in die TupleSpaces schreiben oder aus den TupleSpaces lesen. Basierend auf der durch TupleSpaces bereitgestellten Infrastruktur können die Agenten dann TupleSpaces dazu nutzen, Aktionen auszulösen und auf Änderungen der Umgebung (der TupleSpaces) zu reagieren. Sobald von einem Agenten Daten in TupleSpaces geschrieben wurden, haben die anderen Agenten Zugriff darauf. So können Probleme kooperativ, ohne zwingende Kenntnis über die Existenz und die Stati der anderen Agenten, gelöst werden. Es fällt somit kein zusätzlicher Aufwand zur Verwaltung der Interagentenkommunikation an. Ebenso muss keine zusätzliche Kommunikationsschnittstelle definiert werden, da die Agenten, falls sie kommunizieren müssen, für die Kommunikation TupleSpaces nutzen können.

Jeder Client, der mit dem zentralen TupleSpaces-Server verbunden ist, kann Daten in Form von Tupeln schreiben, lesen und nehmen, wie unten in Abbildung 6 illustriert. Nehmen ist eine kombinierte Lesen-löschen-Operation. Eine direkte Kommunikation der Clients untereinander ist nicht möglich. Die TupleSpaces basieren auf der Koordinationssprache Linda, einer Idee von DAVID GELERNTER. [31] GELERNTER beschreibt TupleSpaces, die als Basis für die Kommunikation nebenläufiger Prozesse dienen. Ein Tupel ist hierbei analog zur Mathematik als eine endliche Liste von Elementen zu betrachten. Im Gegensatz zu relationalen Datenbanken, wo die Datensätze, welche die Zeilen einer Relation darstellen, auch Tupel genannt werden, muss und kann für Tupel in TupleSpaces kein Datenbankschema definiert sein. Beim Schreiben muss zwar der jeweilige Datentyp der Attribute des Tupels angeben werden, ansonsten existiert keinerlei Einschränkung und daher müssen auch nicht, wie in einer Datenbank, schon vorhandene Spalten ungünstigerweise mit „NULL“ aufgefüllt werden. Es können für jeden Zweck neue Tupel definiert werden.

Die Tupel werden nicht flüchtig in TupleSpaces abgelegt, was für die einzelnen Agenten bedeutet, dass sie zum Zeitpunkt des Schreibens eines für sie relevanten Tupels nicht aktiv sein müssen, sondern jederzeit Zugriff auf Tupel bekommen, die für sie relevant sind. TupleSpaces als Middleware stellt sicher, dass der Zugriff auf die gespeicherten Tupel konfliktfrei möglich ist.

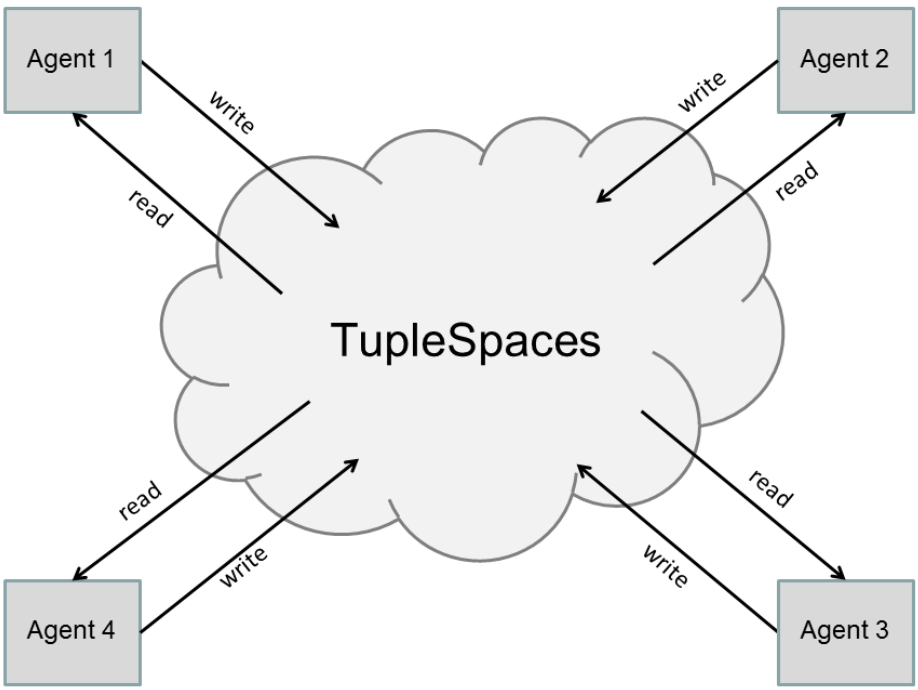

Abbildung 6: TupleSpaces als Multiagentensystem

Die in dieser Arbeit genutzte TupleSpaces-Implementation namens SQLSpaces bietet Clients für verschiedene Sprachen an. Es existiert ein Java Client, ein Android Client und neben dem in dieser Arbeit genutzten PHP Client gibt es auch noch Clients für Prolog, Ruby und C#.[32]

2.6 Ähnlichkeits- und Distanzmaße von Zeichenketten

Zum Beurteilen der Ähnlichkeit zweier Zeichenketten existieren verschiedene Methoden. Die Distanzmaße messen die Unähnlichkeit zweier Zeichenketten, d. h. je größer die Distanz, desto verschiedener sind zwei Zeichenketten. Bei den Ähnlichkeitsmaßen gilt hingegen, dass je größer das ermittelte Maß ist, desto ähnlicher sind sich die beiden betrachteten Zeichenketten.

2.6.1 Die Levenshtein-Distanz (LD)

Bei der LD handelt es sich um die minimale Anzahl von Lösch-, Ersetzungs- und Einfügeoperationen um ein Wort x in ein Wort y zu transformieren. [33] Die LD von „*Ontologie*“ und „*Concept*“ ist 8.

Die möglichen Schritte, um von Ontologie nach Concept zu kommen:

1. Füge „C“ hinzu. Contologie
2. Lösche „t“ Conologie
3. Lösche „o“ Conlogie
4. Ersetze „l“ Concogie
5. Ersetze „o“ Concegie
6. Ersetze „g“ Concepie
7. Ersetze „i“ Concepte
8. Lösche „e“ Concept

Die LD ist also ein Maß, mit dem die Distanz von Wörtern bestimmen werden kann. Je ähnlicher die Wörter sind, desto geringer ist ihre LD. Bei gleichen Wörtern beträgt die LD 0. So kann die LD auch dafür genutzt werden, um aus einem gegebenen Wortschatz ähnliche Wörter vorzuschlagen, also Wörter, die eine geringe LD haben. Allerdings gibt es auch Wörter, die eine geringe LD haben und doch eine völlig unterschiedliche Bedeutung besitzen. Dies ist insbesondere bei „kurzen“ Wörtern der Fall. Wird z. B. „Mund“, „Hund“, „Rund“ und „Fund“ betrachtet, so ist festzustellen, dass trotz geringer LD von jeweils 1 die Wörter weder falsche Schreibweisen eines der anderen sind, noch die gleiche oder eine ähnliche Bedeutung haben.

2.6.2 Die Editierdistanz (ED)

Bei der LD haben alle Operationen – also löschen, einfügen und ersetzen – dieselben Kosten. [33] Die ED setzt voraus, dass eine Ersetzung die doppelten Kosten hat. Diese Überlegung resultiert aus der Tatsache, dass für eine Ersetzung eine Lösch- und eine Einfügeoperation ausgeführt werden müssen. Die ED ist also ein anders gewichtetes Maß für Zeichenkettendistanz als die LD.

2.6.3 n-Gramme

Bei einem n-Gramm handelt es sich um einen Teil einer Zeichenkette, wobei n für die Länge der Teilzeichenkette steht. [34] Häufig werden 2- oder 3-Gramme betrachtet. Es gilt, dass für eine Zeichenkette der Länge l sowie dem an Anfang und Ende der Zeichenkette l entsprechende n-Gramme existieren. Die maximale Anzahl der möglichen Terme beträgt bei einem Alphabet von 26 Buchstaben und vier Sonderzeichen sowie dem Leerzeichen dann auch 31^n Terme. Dies bedeutet für n=3 eine Termobergrenze von 29.791 Termen. Da jedoch nicht alle möglichen Gramme auch tatsächlich vorkommen [35], kann man diese Tatsache zum Erkennen und unter Umständen auch zum Beheben von Eingabefehlern nutzen [36].

Die Menge der aus der Zeichenkette „Ontologie“ resultierenden 3-Gramme ist dann beispielsweise:

X ={ _On, Ont, nto, tol, olo, log, ogi, gie, ie_}

Die Menge der aus der Zeichenkette „Ontology“ resultierenden 3-Gramme ist:

Y ={ _On, Ont, nto, tol, olo, log, ogy, gy_ }

Je mehr dieser n-Gramme übereinstimmen, desto ähnlicher sind sich zwei Zeichenketten. Dazu lässt sich der Simple-Matching-Koeffizient bilden, der als die Größe der Schnittmenge ($X \cap Y$) definiert ist. Dieser liefert für das obige Beispiel einen Wert von 6.

Um den Wert des Simple-Matching-Koeffizienten zu normalisieren, kann man den DICE-Koeffizienten nutzen, der für zwei Terme X und Y wie folgt definiert ist:

$$\mathrm{DICE}(X, Y) = \frac{2 \times |n\text{-Gramme}(X) \cap n\text{-Gramme}(Y)|}{|n\text{-Gramme}(X)| + |n\text{-Gramme}(Y)|}$$

Das Ergebnis des DICE-Koeffizienten ist immer ein normalisierter Wert zwischen null und eins. Je näher der Wert an eins ist, desto ähnlicher sind sich die beiden zugrunde gelegten Zeichenketten.

Für die oben genannten Beispielzeichenketten wird dann der folgende DICE-Koeffizient berechnet:

$$\mathrm{DICE}(X, Y) = \frac{2 \times 6}{17} \approx 70{,}58\%.$$

Diese Angabe ist ein Ähnlichkeitsmaß für die beiden Zeichenketten.

In einem anderen Kontext ist auch eine andere Sinnbelegung des Begriffs n-Gramm möglich, als die hier vorgestellte. Es ist auch möglich, beispielsweise Texte in Wörter zu zerlegen, und damit die Auftretenswahrscheinlichkeit von Wörtern vorherzusagen. Hierum geht es in dieser Arbeit nicht, sondern um die von SHANNON [37] und CAVNAR [34] vorgeschlagene Interpretation, welche Wörter in Buchstabentupel zerlegt.

2.7 Stoppwörter

Stoppwörter sind Wörter, die einen Retrievalprozess nicht beeinflussen. Sie haben eine hohe Wahrscheinlichkeit auch in für eine Anfrage irrelevanten Dokumenten vorzukommen. Stoppworte tragen wenig bis gar nicht zur Unterscheidung der Konzepte einer Concept Map oder Ontologie bei. W. JOHN WILBUR und KARL SIROTKIN gehen sogar so weit, Stoppwörter als inhaltsleere Worte zu bezeichnen. [38]

Solche Stoppwörter können in Listen zusammengefasst werden, die stets sprachspezifisch sind. Die Stoppwörter der Liste werden bei einem Recherchevorgang ignoriert, wenn sie nicht explizit Teil einer Suchanfrage sind. Hierzu werden drei Arten von Stoppwörtern unterschieden [39]:

i) **Allgemeine Stoppwörter** sind Pronomen, Artikel, Konjunktion, Präpositionen und Partikel, die als Funktionswörter als inhaltsarm bzw. inhaltsfrei betrachtet werden. [40]
ii) **Domänenspezifische Stoppwörter** sind solche, die man in einer bestimmten Wissensdomäne nicht betrachten möchte.
c) **Dokumentspezifische Stoppworte** eignen sich zwar zum Finden von Dokumenten, jedoch nicht, um die besten Unterthemen innerhalb eines solchen Dokumentes zu finden. [41]

2.8 Conflation

Wörter in Texten sind flektiert, d. h. sie kommen in verschiedenen Formen, den morphologischen Varianten, vor. Das macht den Vergleich und die Suche von Wörtern schwierig, weil alle zu einem Wort passenden Flexionsformen betrachtet werden sollen. Lemmatisierung auf der einen Seite und Stemming auf der anderen Seite sind die beiden Ansätze, mit deren Hilfe versucht wird, eine gemeinsame Form für alle morphologischen Varianten eines Worts zu erhalten. [39]

2.8.1 Lemmatisierung

Ein Lemma ist die normierte Grundform eines Wortes und damit die Form, in der es auch in einem Lexikon steht. Bei der Lemmatisierung geht es darum, die Grundform zu einem Wort zu erzeugen. Es geht darum, Wörter von ihren Flexionsformen zu befreien, und Derivate zurückzuführen. [39] Es sollen also für Substantive der Nominativ Singular, für Verben der Infinitiv Präsens sowie für Adjektive und für Adverbien der Positiv ermittelt werden. Hierbei werden zwei verschiedene Verfahren unterschieden. Es gibt die **regelgeleiteten Verfahren**, die „auf maschinellem Wege ohne Verwendung eines Grundformenwörterbuchs lexikografische Grundformen“ eruieren. [42] Bei **wörterbuchbasierten Verfahren** wird neben einem Erkennungsalgorithmus ein Lexikon genutzt.

2.8.2 Stemming

Ziel des Stemming, oder auch Wortstammbildung, ist es, die Suffixe von Wörtern so zu entfernen, dass Wörter, die zusammengehören, denselben Stem erhalten, also Äquivalenzklassen von Wörtern zu identifizieren. Bei diesem Prozess muss kein Wort im linguistischen Sinne als Ergebnis

produziert werden. Stemming ist im Gegensatz zur Trunkierung [43] kein reines Stehenlassen einer fixen Anzahl an Zeichen bzw. Abschneiden ab einer bestimmten Stringlänge. Es wird mittels eines Algorithmus versucht, Derivationsaffixe sowie Flexionsaffixe zu entfernen. [44]

Präfixe abschneiden macht wenig Sinn, da viele Präfixe, zumindest in der deutschen Sprache, sinngebend sind. Es ist inhaltlich ein Unterschied, ob jemand ab- oder zulegt oder ob mir etwas ge- oder missfallen hat. Beim Stemming gibt es zwei grundsätzlich verschiedene Verfahren. Zum einen existiert das **Longest-Match-Stemming** und zum anderen das **iterative Stemming**. Der allererste von JULIE BETH LOVINS veröffentlichte Stemming Algorithmus [45] ist ein Longest-Match- Stemmer. Beim Longest-Match-Stemming wird die längste passende, d. h. in einer für den Algorithmus festgelegten Liste definierte, Endung entfernt. Falls ein Abschneiden stattgefunden hat, werden noch Transformationsregeln angewandt, um beispielsweise doppelte Buchstaben zu entfernen oder Aussprachevarianten zu behandeln.

Ein Beispiel für einen iterativen Stemmer ist das von MARTIN F. PORTER für die englische Sprache entwickelte Verfahren [46], das es mittlerweile auch für viele weitere Sprachen, u. a. auch für das Deutsche gibt. Hierbei werden mehrere Arbeitsschritte nacheinander abgearbeitet. Es wird also jeweils das Ergebnis des vorgelagerten Schrittes weiter bearbeitet. PORTER definiert hier zuerst, was ein Konsonant ist und dementsprechend, dass alles, was kein Konsonant ist, ein Vokal sein muss. Eine Aneinanderreihung von Konsonanten wird durch den Buchstaben C repräsentiert und eine Aneinanderreihung von Vokalen durch den Buchstaben V. Für ein Wort ergibt sich so eine alternierende Folge von Vs und Cs. Jedes Auftreten einer VC-Folge wird gezählt und ergibt das m-Maß des Wortes. Auf Basis dieses m-Maßes werden zwei Regionen definiert: R1, welche die Region nach dem ersten Konsonanten, der einem Vokal folgt, ist, oder falls ein solches Vorkommen nicht existiert, null ist. Für R2 gilt dasselbe innerhalb von R1. Das Wort Treibhauseffekt erhält so die R1: *hauseffekt* und die R2: *effekt*.

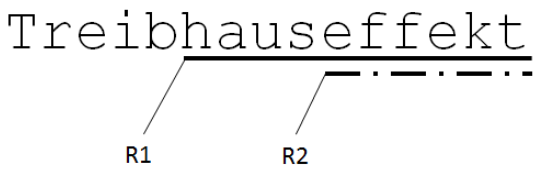

Abbildung 7: Illustration von R1 und R2 am Beispiel Treibhauseffekt

Es werden noch einige weitere Regeln definiert und danach fünf Iterationsrunden angewandt. An deren Ende steht der Stem der Wortform.

2.9 Wortschatz.Uni-Leipzig.de

Der Wortschatzservice der Universität Leipzig bietet verschiedene Web Services an, die einen direkten Zugriff auf die Daten des Projektes zulassen. Hierzu ist für die verschiedenen angebotenen Services jeweils eine SOAP-Schnittstelle definiert. Dies ermöglicht den weltweiten

webbasierten Zugriff mittels einer selbst entwickelten Software, die selbstverständlich SOAP unterstützen muss.

Dabei ist der aus digitalen Zeitungsartikeln und Webseiten zusammengetragene deutsche Korpus mit 500 Millionen Token in 36 Millionen Sätzen repräsentiert. Token sind in diesem Zusammenhang die einzelnen Wörter. Es sind bisher 9 Millionen Wortformen automatisch analysiert worden, die sich auf ihre Grundform abbilden lassen.

Momentan sind 19 von 21 angebotenen Web Services frei nutzbar. Die umgesetzte Implementierung beschränkt sich auf die Services *Baseform*, der die lemmatisierte Form, also die lexikalische Grundform des Eingabewortes, zurückgibt sowie *Synonyms* und *Thesaurus*, die beide Synonyme des Eingabewortes zurückliefern. Zusätzlich werden die Wortarten der Wörter mitgeliefert, wobei gilt, dass „N“ Nomen, „A“ Adjektive und Adverbien, „V“ Verben, „S“ Unbekannt, „VN“ Vorname, „NN“ Nachname (auch bei Wörtern, die nur selten „NN“ und ansonsten „N“ sind), „PN“ Personenname und „-“ nichts von alledem repräsentieren. [47]

Es existiert eine Anfragelimitierung, die allerdings nicht genauer spezifiziert wird.

3. Konzept

Um die Aufgabenstellung anzugehen, gilt es verschiedene Aufgaben aus diversen Gebieten der Informatik zu lösen. So liegen beispielsweise im Bereich der Computerlinguistik und des Information Retrieval ein großer Anteil der zu lösenden Teilprobleme. Hierzu wird zunächst die Datenbasis, d. h. die gegebene Menge von Concept Maps, sowie die gegebene Ontologie hinsichtlich ihrer Vergleichbarkeit untersucht. Es werden entsprechende Verfahren, wie das Filtern von Stoppwörtern oder das Nachschlagen von Synonymen in (Online)Lexika durch eigenständig arbeitende Softwareagenten angewandt, um die Qualität des Retrievals zu verbessern. Ein statistisches Modell zur Repräsentation der Concept Maps wird auf Basis der Ähnlichkeiten von Konzepten und Auftretenshäufigkeiten von Relationen entwickelt. Anhand eines Vergleichs des erstellten „statistischen" Modells und der gegebenen Ontologie werden Erweiterungsvorschläge für die Ontologie abgeleitet und dem KE zur Übernahme in die Ontologie vorgeschlagen. Um die Aufnahme von falschem Wissen zu verhindern, entscheidet der KE welche Vorschläge in die Ontologie aufgenommen werden.

Die zur Lösung der Problemstellung entwickelten Softwareagenten arbeiten in einem Multiagentensystem mit TupleSpaces als Middleware. Sie kommunizieren über TupleSpaces und speichern dort ihre Ergebnisse.

3.1 Ontology Learning

Die durch dieses Konzept vollziehbaren Verbesserungen der Ontologie sind in den Kontext des Ontology Learning einzuordnen. „Ontology Learning aims at developing methods and tools that reduce the manual effort for engineering and managing ontologies" [48] Durch das Nutzen von Concept Maps zur Evolution der Ontologie wird die Arbeit des KE erleichtert. Er muss dazu nicht selbst zeitaufwendig recherchieren, welche Konzepte sich zum Einpflegen in die Ontologie eignen. Stattdessen werden ihm Vorschläge gemacht, welche er auf der Basis seines Wissens prüfen kann.

Die in der Vorstellung des Konzepts eingeführten Schritte lassen sich in die von ALEXANDER MAEDCHE vorgeschlagene Architektur für Ontology Learning [48], die in Abbildung 8: Ontology Learning Architektur [48] zu sehen ist, eingliedern. „Import semistructured data" findet beim Einlesen der Concept Maps statt. Der nächste Schritt ist das „Resource Processing" in Form von Text Processing, das auch mittels angebundener Lexika möglich ist. Die so erhaltenen Daten werden dann mittels der „Algorithm Library", in die der Vergleich einzuordnen ist, aufbereitet und dem KE in einem GUI zur Übernahme in die Ontologie präsentiert. Hierzu stellt MAEDCHE fest, dass die GUI den KE darin unterstützt, die relevanten Daten auszuwählen. [48].

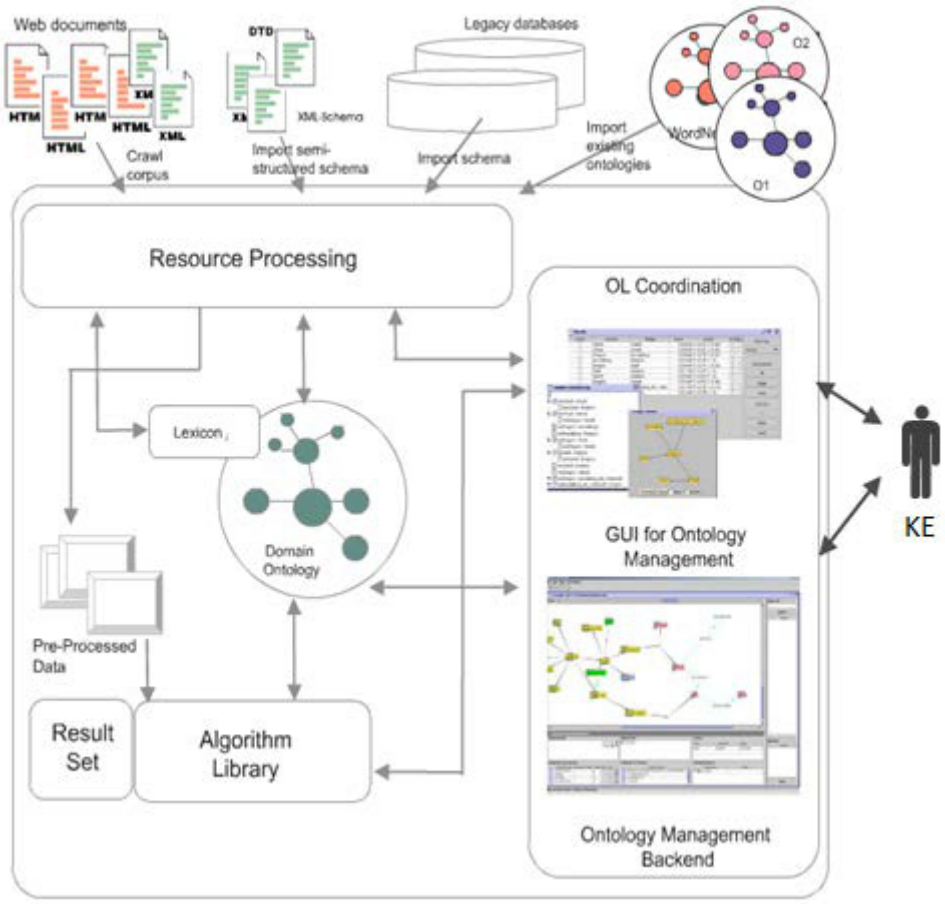

Abbildung 8: Ontology Learning Architektur [48]

3.2 Datenbasis

Die Datenbasis besteht aus einer Menge von 37 Concept Maps und einer Ontologie. Sowohl Ontologie als auch Concept Maps beschäftigen sich mit dem Themengebiet *„Globale Erwärmung“*. [49]

3.2.1 Die Concept Maps

Die 37 Concept Maps sind im Schulunterricht entstanden. Sie enthalten im Schnitt etwa 15 Konzepte je Concept Map. Insgesamt existieren 543 von den Schülern per Freitextfeld eingegebene Konzepte. Diese 543 Konzepte sind mit insgesamt 563 Relationen verbunden. Die Relationenbezeichner waren vordefiniert und konnten per Dropdown-Menü gewählt werden. Es war somit keine Freitexteingabe möglich. Deshalb mussten die Relationenbezeichner hinsichtlich ihrer linguistischen Gesichtspunkte nicht weiter untersucht werden. Sieben Relationentypen existieren: *„beruht auf“, „bezieht ein“, „erhöht“, „hat Einfluss auf“, „hat Verbindung zu“, „ist ein“, „verringert“*. In den 543 Konzepten lassen sich, auf Basis einfacher Gleichheit zwischen den Labeln der Konzepte, 269 unterschiedliche Konzepte identifizieren. Da die Konzepte im Gegensatz zu den Relationen frei eingegeben werden konnten, reicht dieser Vergleich jedoch nicht aus, um alle

gleichen Konzepte zu erkennen. Durch die Freitexteingabe werden direkt einige sprachliche Probleme impliziert, die zur Erkennung der Gleichheit von Konzepten aufgelöst werden. Sie sind deshalb schlecht vergleichbar, weil entweder Abkürzungen wie *Durchschnittstemp*. genutzt wurden, oder Schreibfehler wie *Abohlzung* vorliegen. Andere Konzepte unterscheiden sich durch ihre Flexionsform wie *Auswirkung auf die Umwelt* gegenüber *Auswirkungen auf die Umwelt*. Eine weitere Gruppe von Konzepten, die das Gleiche bedeuten, lässt sich dann erst identifizieren, wenn die Konzeptlabel bereits ihrer Flexionsaffixe und zusätzlich der in ihnen existierenden Stoppwörter bereinigt sind. Konzepte, die mehrere unterschiedliche Konzepte in ihrem Label vereinen, treten in vielfacher Schreibweise auf. Es gibt zum einen Label in Konzepten mit Ergänzungsstrich, wie „Land- und Viehwirtschaft“. Zum anderen existieren Label in Konzepten, durch welche zwei Begriffe ausgedrückt werden, die mit einem „und“ getrennt sind, wie „Methan und Lachgas“. Schließlich gibt es noch eine Gruppe, in der das eine Konzept ein Synonym des anderen Konzeptes ist. Der Auszug aller Konzepte der Concept Maps befindet sich in Tabelle 3: 269 "unterschiedliche" Konzepte im Anhang.

3.2.2 Die Ontologie

Die Domäne der Ontologie ist *„globale Erwärmung“*. Es sind drei Oberklassen definiert. Die Klasse Konzept, die Klasse Stichwort und die Klasse Stichwortliste. In der Klasse Konzept sind neun Konzeptunterklassen definiert. Diese enthalten in bis zu zwei Unterebenen weitere Unterklassen, also Unterkonzepte, sowie, auf der untersten Ebene, Instanzen der jeweiligen Unterklassen von Konzept. In der Klasse Stichwort sind 100 Stichworte, von *„Abfälle“* über *„Kraftstoffe“* bis zu *„Zertifikate“* definiert, die über die Relation „belongsToList“ einer von 8 Stichwortlisten zugeordnet sind. Alle Klassen und Instanzen in der Ontologie besitzen bisher genau ein Label in der deutschen Sprache und ein Label in der englischen Sprache. Abbildung 9 illustriert einen Ausschnitt der zugrunde liegenden Ontologie. „*Außentemperatur*“, „*Niederschlag*“, „*Sonneneinstrahlung*“ und „*Wind*“ sind dort eine Instanz der Klasse „*Klima*“. Dies ist eine Unterklasse von „*geographischer Parameter*“. So fortgesetzt lassen sich weitere Abhängigkeiten ableiten, bis über die Wurzelklasse „Konzept“ die oberste Ebene mit „Thing“ erreicht wird, das als Sammelklasse in der OWL-Spezifikation definiert ist und alle „Individuen“ in sich vereint, indem es die Superklasse aller Klassen ist. [9]

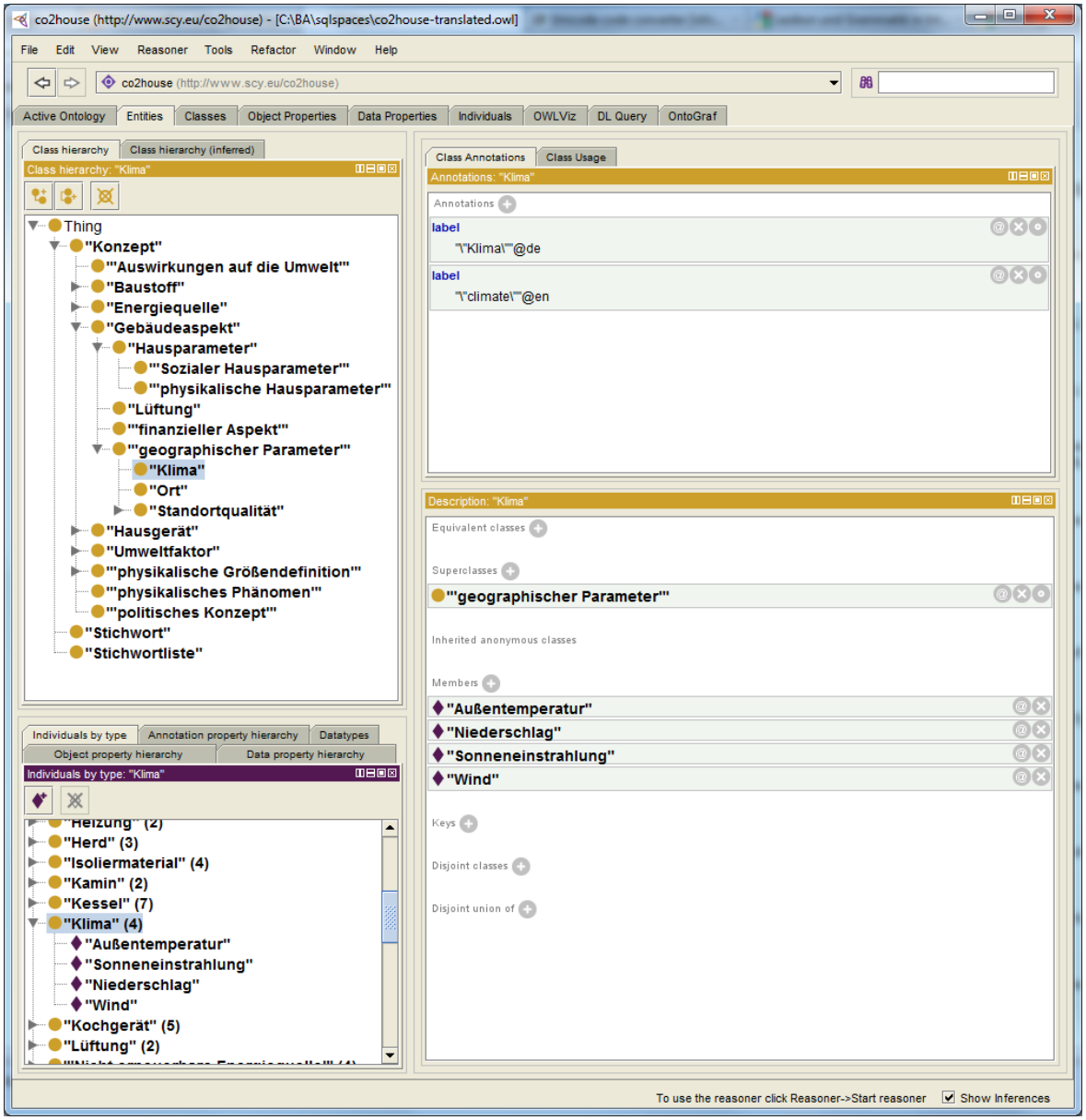

Abbildung 9: Ausschnitt aus der zugrunde liegenden Ontologie

Zusätzlich zu „belongsToList“ gibt es 12 Relationen (*„hasRelationTo“, „coincidesWidth“, „dependsOn“, „hasInfluenceOn“, „decreases“, „increases“, „includes“, „isInfluencedBy“, „isDecreasedBy“, „isIncreasedBy“, „isPrerequesiteOf“, „isMadeOf“*), die Objektabhängigkeiten beschreiben, die – wie in folgender Abbildung zu sehen – hierarchisch strukturiert sind.

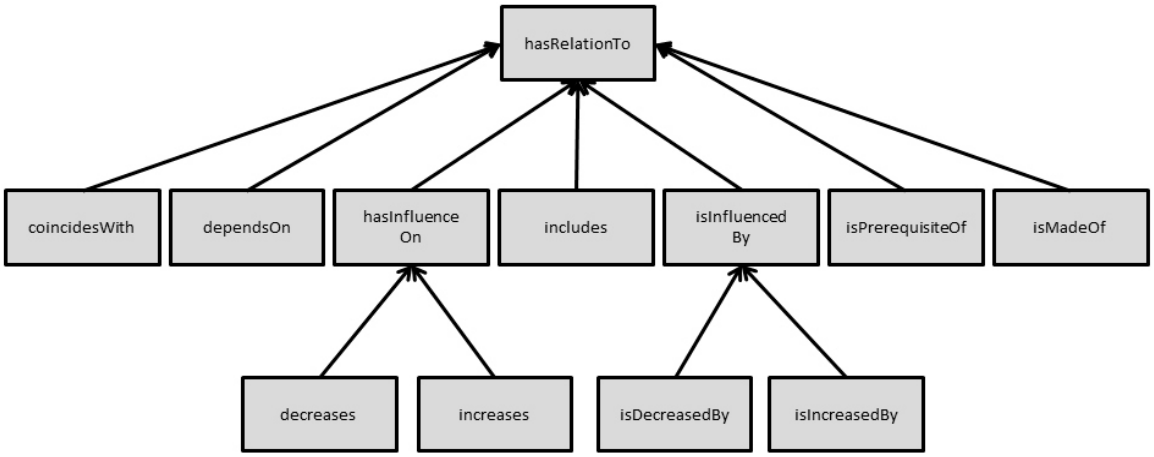

Abbildung 10: Zwölf Relationen zum Ausdrücken von Objektabhängigkeiten (der Pfeil hat die Bedeutung: subPropertyOf)

Aus dieser hierarchischen Sortierung der Relationen und der Tatsache, dass die Namen für die Beziehungen in den Concept Maps die deutschen Bezeichner der in Abbildung 10 dargestellten Relationentypen sind, lässt sich das Mapping der Relationen, wie in Tabelle 2 dargestellt, ersehen. Ebenso können auch Verallgemeinerungen gemacht werden. Es lassen sich alle Relationen durch *hasRelationTo* ausdrücken. *increases* und *decreases* können durch *hasInfluenceOn* verallgemeinert werden.

Concept Map	Entsprechung in der Ontologie	Gegenrichtung in der Ontologie
beruht auf	dependsOn	
bezieht ein	includes	
erhöht	increases	isIncreasedBy
hat Einfluss auf	hasInfluenceOn	isInfluencedBy
hat Verbindung zu	hasRelationTo	
ist ein	subClassOf	
verringert	decreases	isDecreasedBy

Tabelle 2: Mapping der Concept Map Relationen auf die Relationen der Ontologie

3.3 Auflösung linguistischer Probleme

Wie bereits in Abschnitt 3.2.1 beschrieben, müssen verschiedene linguistische Probleme aufgelöst werden, um zusammengehörige Konzepte erfolgreich vergleichen und vereinen zu können. Hierzu gehören neben Rechtschreibkorrekturen die Auflösungen von Abkürzungen, die innerhalb der Labels der Concept Map Konzepte auftreten. Synonyme müssen identifiziert werden und Wörter in Äquivalenzklassen zusammengefasst werden, um den Recall einer Suche zu vergrößern.

3.3.1 Rechtschreibkorrektur

Als Verfahren zur Rechtschreibkorrektur wird eine Kombination mehrerer Verfahren angewandt. Zum einen werden mittels einer Kombination der Ähnlichkeitsmaße, die auf die in Abschnitt 2.6.3

beschriebenen n-Gramme angewandt werden und der in Abschnitt 2.6.2 beschriebenen Distanzmaße Rechtschreibfehler identifiziert und behoben. Diese Rechtschreibfehler sind die Manifestation von Tippfehlern. Zum anderen wird eine Liste geführt, die typische Fehler und Korrekturen enthält. Rechtschreibkorrekturen sollten auf die Originalkollektion angewandt werden, in diesem Fall die Konzepte der Concept Maps, weil sie nicht Sinn verändernde Ersetzungen sind. Hierzu werden die Label der Konzepte tokenisiert. Das bedeutet, dass die Label in ihre einzelnen Wörter zerlegt werden. [50] Die im Folgenden beschriebenen Verfahren arbeiten, mit Ausnahme des listenbasierten Verfahrens, mit den so entstandenen Token.

3.3.1.1 Listenbasiertes Verfahren

In einem, dem eigentlichen Verfahren vorgelagerten, Bearbeitungsschritt wird eine Liste mit Schreibfehlern festgelegt. Dies kann auch zur Verbesserung der Ergebnisse in einem tiefer in der Prozesskette angelagerten Schritt stattfinden, wenn danach der gesamte Wissensextraktionsprozess erneut angestoßen wird. Diese Liste kann dann auf die Concept Maps angewandt werden und es können mit ihrer Hilfe drei Arten von Korrekturen durchgeführt werden. **Zeichencodierungsfehler**, die in der Kollektion auftreten und in fehlerhafter Darstellung von Sonderzeichen resultieren, können so behoben werden. Konkret traten in den betrachteten Concept Maps Konzepte auf, die Zeichen enthielten, welche durch „&#“ escaped wurden. **Abkürzungen** können in einem solchen Verfahren auf ihre ausgeschriebenen Repräsentationen abgebildet werden. Korrekturen für **weitere Schreibfehler** können in diese Liste eingepflegt werden, um die Ergebnisse der Korrektur zu verbessern.

3.3.1.2 Verfahren basierend auf Distanz- und Ähnlichkeitsmaßen

Die in Abschnitt 2.6.3 beschriebenen n-Gramme werden bei diesem Verfahren dazu genutzt, Schreibfehler anhand der vermutlich richtig geschriebenen Wörter zu korrigieren. Dazu wird die Annahme getroffen, dass ein richtig geschriebenes Wort in der Kollektion häufiger vorkommt als ein falsch geschriebenes. Die Häufigkeit des Auftretens soll dabei größer als zwei sein, um zufällig häufiger falsch geschriebene Wörter auszusortieren. In einer größeren Kollektion muss wahrscheinlich eine andere Grenze gesetzt werden, weshalb der Nutzer des Systems diesen Wert verändern kann. Für die vorliegende Ausprägung hat sich der Wert durch Betrachten der Ergebnisse als offenbar gut erwiesen. Des Weiteren bleibt zu beachten, dass kurze Worte wie *„Mund“* und *„Rund“* zwar eine kleine LD (siehe Abschnitt 2.6.1) besitzen, jedoch auch ihr DICE- Koeffizient (siehe 2.6.3) klein ist. Wird ein Schwellwert für den DICE-Koeffizienten festgelegt, der groß genug ist, so wird ebenfalls die auf der scheinbaren Ähnlichkeit durch eine kleine LD oder ED basierende, möglicherweise fehlerhafte, weil sinnentstellende Korrektur, wie im vorgenannten Beispiel, von kurzen Wörtern nicht stattfinden. Daher muss nicht extra eine Mindestlänge für die zu korrigierenden Wörter festgelegt werden. Dies kann jedoch aus Gründen der Effizienz ratsam sein, da für diese kurzen Wörter dann keine n-Gramme bestimmt werden müssen und keine LD berechnet werden muss. Ein als richtig geschrieben betrachtetes Wort tritt häufiger als s-Mal auf. Der Schwellwert für s kann vom Nutzer festgelegt werden. Wenn ein solches als richtig geschrieben festgelegtes Wort in einem Konzept auftritt, dann werden alle anderen Wörter,

deren LD einen Wert von 2 nicht überschreitet und mindestens einen DICE-Koeffizienten von 0,8 zu diesem besitzen, durch das als richtig geschrieben definierte Wort ersetzt. Hiermit können drei Arten von Schreibfehlern zuverlässig behoben werden, sofern das entsprechende Wort häufig genug in der Kollektion auftritt. **Buchstabendreher**, wie in *„Abohlzung“* werden erkannt und korrigiert. **Auslassungen** von Buchstaben, wie in *„Gletscherscmelze“* werden ebenso wie **Tippfehler**, wie in *„Nahrungakette“* entsprechend korrigiert.

3.3.2 Synonymdienste

Zum Nachschlagen von Synonymen und zur Lemmatisierung werden verschiedene Online- und Offlinedienste genutzt. Die Wahl fiel dabei auf wortschatz.uni-leipzig.de und openthesaurus.org, da beide online verfügbar sind und jeweils eine frei verfügbare Schnittstelle zum Abrufen anbieten. Ein ebenfalls interessanter Dienst ist *GermaNet – The German Wordnet*, denn er bietet als lexikalisch-semantisches Netz eine ähnliche Funktionalität, wie die vorgenannten. Bei *GermaNet* ist jedoch eine Registrierung erforderlich, was das Weiternutzen einer auf seiner Basis entwickelten Software als problematisch gestaltet. Jeder neue Nutzer des in dieser Arbeit entwickelten Systems müsste sich dann bei GermaNet registrieren und die Nutzungsbedingungen akzeptieren.

Die genutzten Dienste eignen sich des Weiteren dazu festzustellen, ob ein Wort ein Kompositum ist und ob ein Ergänzungsstrich wie in Abschnitt 3.2.1 beschrieben vorliegt.

3.3.2.1 Wortschatz.uni-leipzig.de

Es ist mit automatischen Methoden schwierig, Homonyme und Polyseme zu disambiguieren. Der genutzte Wortschatzservice der Universität Leipzig sieht leider keine Angabe einer Domäne zwecks Disambiguierung und dadurch Vermeidung von falschen bzw. schlechten Synonymvorschlägen vor. So liefert die Synonymsuche nach *„Niederschlag“* neben durchaus sinnvollen Vorschlägen wie *„Dauerregen“, „Gewitter“, „Gewitterregen“, „Gewitterschauer“, „Hagel“, „Hagelschauer“, „Regen“* usw. leider auch das Polysem *„Knock-out“* sowie weitere ungünstige Vorschläge wie *„Ablagerungen“, „Absatz“, „Bodensatz“* und *„Sediment“*.

In dem Wortschatz scheinen auch ungeeignete Synonyme gespeichert zu sein. Einer der Synonymvorschläge für „Ozean“ ist „Pazifik“. Dies jedoch ist ein falscher Zusammenhang, „Pazifik“ wäre eine Instanz des Begriffs „Ozean“ und mitnichten ein Synonym für das Konzept „Ozean“. Gute Synonyme wären in diesem Fall „Meer“, „See“ und „Weltmeer“, wobei leider „See“ selbst wiederum ein Homonym ist. Dies zeigt, dass ein vollautomatisches Verfahren zur Ontologieevolution hier nicht angeraten ist, und ein KE oder DE zur Disambiguierung der Begriffe nötig ist.

3.3.2.2 Openthesaurus.org

Es kann auch vorkommen, dass der Wortschatzservice nicht in der Lage ist, ein Synonym zu einem gegebenen Wort zu liefern. In diesem Fall wird ein weiterer Service zum Liefern von Synonymen genutzt. Hierbei handelt es sich um openthesaurus.org. Bei diesem Service existiert eine

Anfragelimitierung, wie bei Wortschatz.uni-leipzig.de auch, sodass mehr als 60 Anfragen pro Minute nur möglich sind, wenn dort um eine Erhöhung des Limits gebeten wird. Es besteht jedoch die Möglichkeit, die Datenbank herunterzuladen und beispielsweise in einem Datenbanksystem wie MySQL zu nutzen. Da es jedoch im Hinblick auf die Skalierbarkeit und Nutzung von anderen als der in dieser Arbeit betrachteten, eher an die Aktualität gebundenen, Domänen sinnvoll ist, möglichst aktuelle Daten zu erhalten und weiterzuverarbeiten, wird openthesaurus.org für den oben beschriebenen Fall genutzt. Hiermit wird die Anfragelimitierung nicht erreicht. Der Korpus von openthesaurus umfasst etwa 69.000 Wörter. [51]

3.3.3 Deutscher Stemming Algorithmus

Zum Stemmen der deutschen Wörter, die als Label in den Concept Maps auftreten, eignet sich der im Grundlagenkapitel beschriebene Stemmingalgorithmus, wenn er für die deutsche Sprache angepasst ist. Auf Basis des von MARTIN PORTER für die englische Sprache entwickelten Algorithmus wurde ein solches Vorgehen für die deutsche Sprache vorgeschlagen [52] (siehe Anhang).

Das Ergebnis ist ein gestemmtes Wort. Der Vorteil dieses Verfahrens ist, dass mit Hilfe eines überschaubaren Regelwerks Stems erstellt werden, die dann den Recall, also die Trefferquote bei einem Vergleich von zwei Konzepten, verbessert. [53] Der Nachteil ist, dass sogenanntes Overstemming und Understemming auftreten können. Dies verschlechtert die Precision, also die Genauigkeit und somit die Wahrscheinlichkeit, mit der zwei Konzepte als gleich betrachtet werden können. Overstemming bezeichnet das Erzeugen von künstlichen Ambiguitäten, d. h., zwei in unterschiedliche konzeptuelle Gruppen gehörende Wörter werden auf denselben Stem zurückgeführt. [53] Dies stellt in der späteren Implementierung insofern ein Problem dar, da es den Recall erhöht. Dies bedeutet, dass mehr Ergebnisse vom KE oder DE bearbeitet werden müssen. Ein weiteres Problem bereitet das Understemming. Es bedeutet, dass zwei Wörter in der gleichen konzeptuellen Gruppe sind, aber ein unterschiedlicher Stem für sie erzeugt wird. Ein Beispiel aus der gegebenen Kollektion verdeutlicht dies. Es ist zu erwarten, dass *„Versauerung“* und *„versäuerte“* auf denselben Stem reduziert werden. Der Algorithmus stemt jedoch *„Versauerung“* zu *„versaur“* und *„versäuert“* zu *„versaurt“*. Beiden Problemen kann mit Hilfe von Listen mit irregulären Wörtern begegnet werden. Um für das umzusetzende Projekt nicht noch eine weitere Liste pflegen zu müssen, wird jedoch hiervon abgesehen.

3.4 Vorgegebene Tupelformate

Im Rahmen der Bachelorarbeit „Entwicklung eines Concept Mapping Tools mit Ontologie-Anbindung“ von BOB BAUMANN wurde bereits je ein Tupelformat für Knoten und Kanten der Concept Maps entwickelt. Das dort vorgeschlagene Format ist für Knoten: String(UID), String(``Node´´), String(content), String(color) und für Kanten: String(UID), String(``Edge´´), String(content), String(``sym´´/``asym´´), String(fromUID), String(toUID).

In dem vorgeschlagenen Tupelformat liegt die Schwäche, dass die Quelle nicht mehr zugeordnet werden kann. Somit ist weder bekannt, ob beispielsweise FreeStyler [54] oder ein XML – Import als Ursprung der Daten betrachtet wird. FreeStyler ist ein an der Universität Duisburg-Essen

entwickeltes Modellierungswerkzeug mit einer Plug-in basierten Architektur. Die Information über die ursprüngliche Concept Map geht bei dieser Repräsentation der Knoten und Kanten ebenfalls verloren, sodass sich nur mittels rechenzeitintensiver Suchverfahren wie Breiten- oder Tiefensuche ermitteln lässt, welche Knoten und Kanten zu einer Concept Map gehört haben. Das Format wird daher um zwei weitere Felder am Anfang erweitert, welche die ursprüngliche Concept Map sowie die Quelle des Imports angeben. Für Knoten ist es dann: String(MapID), String(Source), String(UID), String(``Node´´), String(content), String(color) und für Kanten entsprechend: String(MapID), String(Source), String(UID), String(``Edge´´), String(content), String(``sym´´/``asym´´), String(fromUID), String(toUID). Dies ermöglicht alle Knoten und Kanten einer Concept Map ohne weitere Suche direkt zu referenzieren und aus TupleSpaces auszulesen.

3.5 Kandidaten für Vorschläge

Aus der Betrachtung der Concept Maps und der Ontologie folgt, dass es Konzepte in den Concept Maps gibt, die nicht in der Ontologie vertreten sind. Solche Konzepte geben dann, wenn sie in vielen Concept Maps auftreten, einen Kandidaten für einen Vorschlag an den Nutzer des Systems ab, wie in Abbildung 11 durch rote Kreise kenntlich gemacht wurde. In der Ontologie, die durch den unteren rechten Graphen teilrepräsentiert ist, kommen die Konzepte „*Menschliches Enwirken*“ und „*verbrennen fossiler Brennstoffe*“ nicht vor, während sie in der exemplarisch ausgewählten Concept Map eine Beziehung zu „*Treibhauseffekt*“ haben.

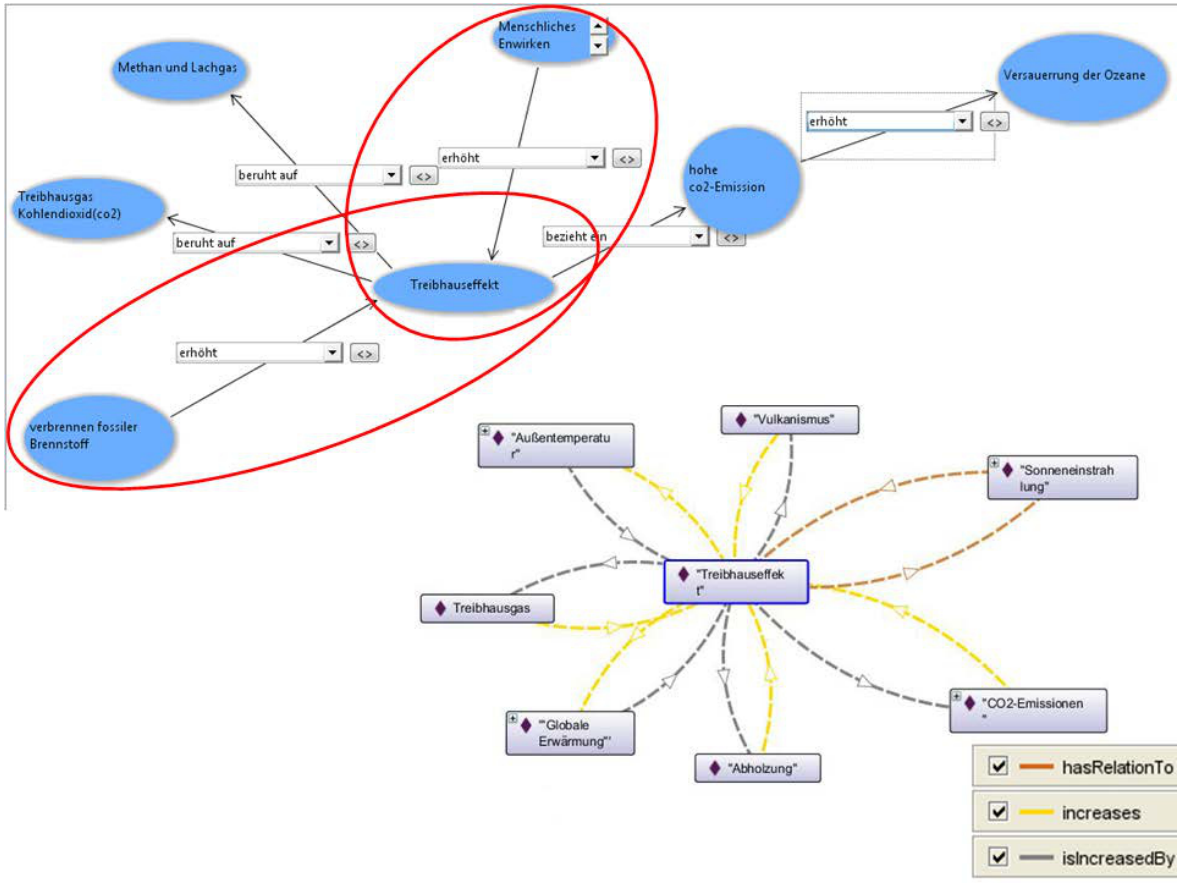

Abbildung 11: zwei Kandidaten für einen Vorschlag an den Nutzer (rot markiert)

3.6 „Statistisches“ Modell der Concept Maps

Es wird ein Vergleich der Konzepte auf Basis der ermittelten Synonyme, Stems, Lemmas sowie des ED und der LD durchgeführt. Mit Hilfe von jeweils festgelegten Schwellwerten, ab denen zwei solche Konzepte als ähnlich betrachtet werden, entsteht ein „statistisches“ Modell der Menge der Concept Maps. Die ähnlichen Konzepte werden jeweils zu einem „Überkonzept“ zusammengefasst, das wiederum dem Konzeptbegriff in der Ontologie, mit seiner Möglichkeit viele verschiedene Label für ein Konzept zu benennen, entspricht. Um Verwechslungen vorzubeugen, wird im Folgenden der Begriff Konzeptcontainer für eine solche Zusammenfassung verwandt. In einem solchen Konzeptcontainer werden zusätzlich zu den Labeln der ähnlichen Konzepte Verweise auf deren Synonyme abgelegt. Die Dimension eines Konzeptcontainers ist definiert als die Anzahl der in ihm vereinten Konzepte. Ein Konzeptcontainer hat dann Verbindungen zu anderen Konzeptcontainern. Diese Relationen fließen ebenfalls in das „statistische“ Modell ein. Wo zuvor in einer einzelnen Concept Map nur eine Relation zwischen zwei Konzepten aufgetreten ist, tritt zwischen zwei Konzeptcontainern diese Relation möglicherweise mehrfach auf. Das führt zu gewichteten Kanten zwischen den Konzeptcontainern. Für die Gewichtung werden alle sieben Relationentypen der Concept Maps einzeln aufgrund ihrer Auftretenshäufigkeit zwischen zwei Konzeptcontainern abgezählt. Der „hat Verbindung zu“ –

Relation wird eine Sonderrolle zuteil, denn sie vereint alle anderen Relationen in sich. Dies soll berücksichtigt werden und dazu führen, dass die Vorschläge aufgrund eines häufigen Auftretens überhaupt einer Relation zwischen zwei Konzeptcontainern gemacht werden.

Werte, die ebenfalls mit in die Statistik einfließen, sind die Anzahl der Maps und die Dimension der Konzeptcontainer. Anhand dieser beiden Werte lässt sich die Auftretenshäufigkeit eines Konzeptes festlegen. Die relative Konzepthäufigkeit sei dann definiert als

$$h_{\text{[illegible]}}(\text{[illegible]}) = \frac{\dim(\text{[illegible]})}{\text{[illegible]}} \cdot |\text{[illegible]}|$$

So entsteht ein Modell, das eine „große“ Concept Map repräsentiert, die alle anderen Concept Maps in sich vereint und durch diese Vereinigung sowohl gewichtete Kanten, als auch gewichtete Konzepte enthält.

3.7 Benutzerinterface

Mit Hilfe des Modells der Concept Maps können Vorschläge für die Erweiterung der Ontologie abgeleitet werden. Das Modell der Concept Maps wird hierzu mit der Ontologie verglichen. Kommt ein Konzept in der Ontologie nicht vor, ist jedoch als Konzeptcontainer im Modell der Concept Maps höher als ein vom System voreingestellter und vom KE einstellbarer Schwellwert gewichtet, wird daraus ein Vorschlag generiert. Dazu wird untersucht, ob der Konzeptcontainer eine Relation zu einem Konzept(container) enthält, die eine Verbindung mit einem in der Ontologie auftretenden Konzept zulässt. Falls das nicht so ist, muss dem KE später beim Darstellen der Vorschläge die Möglichkeit gegeben werden, eine solche Verbindung festzulegen. Gibt es die Verbindung, so kann diese Verbindung auch derart vorgeschlagen werden, dass die am häufigsten auftretende Relation diese Verbindung repräsentiert. Zwei Beispiele: Es wird ein neues Konzept gefunden, das in der Ontologie nicht vorkommt. Der Konzeptcontainer trägt das Hauptlabel „menschliches Einwirken“. „Menschliches Einwirken“ ist häufig über die Kante „erhöht“ mit „Treibhauseffekt“ verbunden. „Treibhauseffekt“ ist ein Konzept aus der Ontologie. Demnach ist der mittels der Mappingtabelle (Tabelle 2: Mapping der Concept Map Relationen auf die Relationen der Ontologie) an den KE zu machende Vorschlag: „Menschliches Einwirken“ mit einer Auswahlliste der Synonyme, „increases“ mit einer Auswahlliste der anderen Relationentypen, „Treibhauseffekt“ mit einer Auswahlliste der Synonyme. Ist „Treibhauseffekt“ nicht in der Ontologie vertreten, generiert sich zunächst derselbe Vorschlag, der jedoch einen zusätzlichen mit ihm verbundenen Vorschlag hervorruft. Dieser zusätzliche Vorschlag enthält dann den Konzeptcontainer „Treibhauseffekt“ mit seiner Auswahlliste an Synonymen, eine Auswahlliste der möglichen Relationen und eine Auswahlliste der in der Ontologie auftretenden Konzepte.

Auf diese Weise können, falls vom KE gewünscht, aus einer Relation im Falle des Nicht-Auftretens beider Konzepte zwei neue Konzepte in die Ontologie übernommen werden. Dies reduziert die Anzahl der Arbeitsschritte des KE. Dem KE ist darüber hinaus eine Möglichkeit gegeben,

Synonyme für in der Ontologie vorhandene Konzepte sowohl selbst zu bestimmen als auch Synonyme anhand von durch die Synonymdienste ermittelten Synonymen zu übernehmen.

In einem letzten und wichtigen Schritt wird die Ontologie um die angenommenen Vorschläge erweitert. Hierzu werden die Aussagetripel in RDF-Tripel überführt und in den Space der Ontologie geschrieben. Dieser Anreicherungsvorgang kann pro angenommenem Vorschlag sofort stattfinden. Nach einem in die Ontologie übernommenen Vorschlag kann die Vorschlagsfindung neu angestoßen werden, wenn dies vom KE gewünscht ist.

4. Implementierung

Es handelt sich bei TupleSpaces um eine Middleware, die sich dazu eignet, als Blackboard genutzt zu werden und damit eine Basis für Multiagentensysteme darzustellen. Die im Konzept beschriebenen Teilaufgaben lassen sich entsprechend in einzelne Softwareagenten aufgliedern, die mehr oder weniger unabhängig voneinander Teilprobleme lösen, um das Gesamtziel zu erreichen. Dafür werden die in Softwareagenten heruntergebrochenen Lösungen bestimmter Teilbereiche in Phasen gegliedert. Die Agenten brauchen bei der konkreten Problemstellung die Daten aus der vorhergehenden Phase, wie sich bei der Beschreibung des Systems zeigen wird.

4.1 Architektur

Wie in Abbildung 12 zu sehen, wurden für das implementierte System auf Basis von TupleSpaces Unterspaces definiert, die die verschiedenen, zu einem Kontext gehörenden Tupel in sich aufnehmen. So ist es möglich, die Daten durch die einzelnen Spaces vorzustrukturieren. Es gibt den Space, der die Concept Maps beinhaltet. Ein weiterer Space, der „Words“-Space, nimmt zuerst die Wörter aus den Concept Maps auf und später auch Synonyme und Grundformen sowie Stems. Im „Modell“-Space wird das „statistische“ Modell der Concept Maps repräsentiert und diese Daten sind die Basis für den Vergleich, der in Vorschlägen resultiert, die in den „Proposals“- Space geschrieben werden. Der Vergleich findet mit der im „ontology“-Space abgebildeten Ontologie statt.

Es gibt noch einen weiteren Space, der für die Interagentenkommunikation genutzt wird. Dies ist der Commandspace, in den Agenten schreiben können, wenn sie ihre Arbeit erledigt haben. Diese Tupel werden in weiteren Verlauf „Fertigstellungstupel“ genannt.

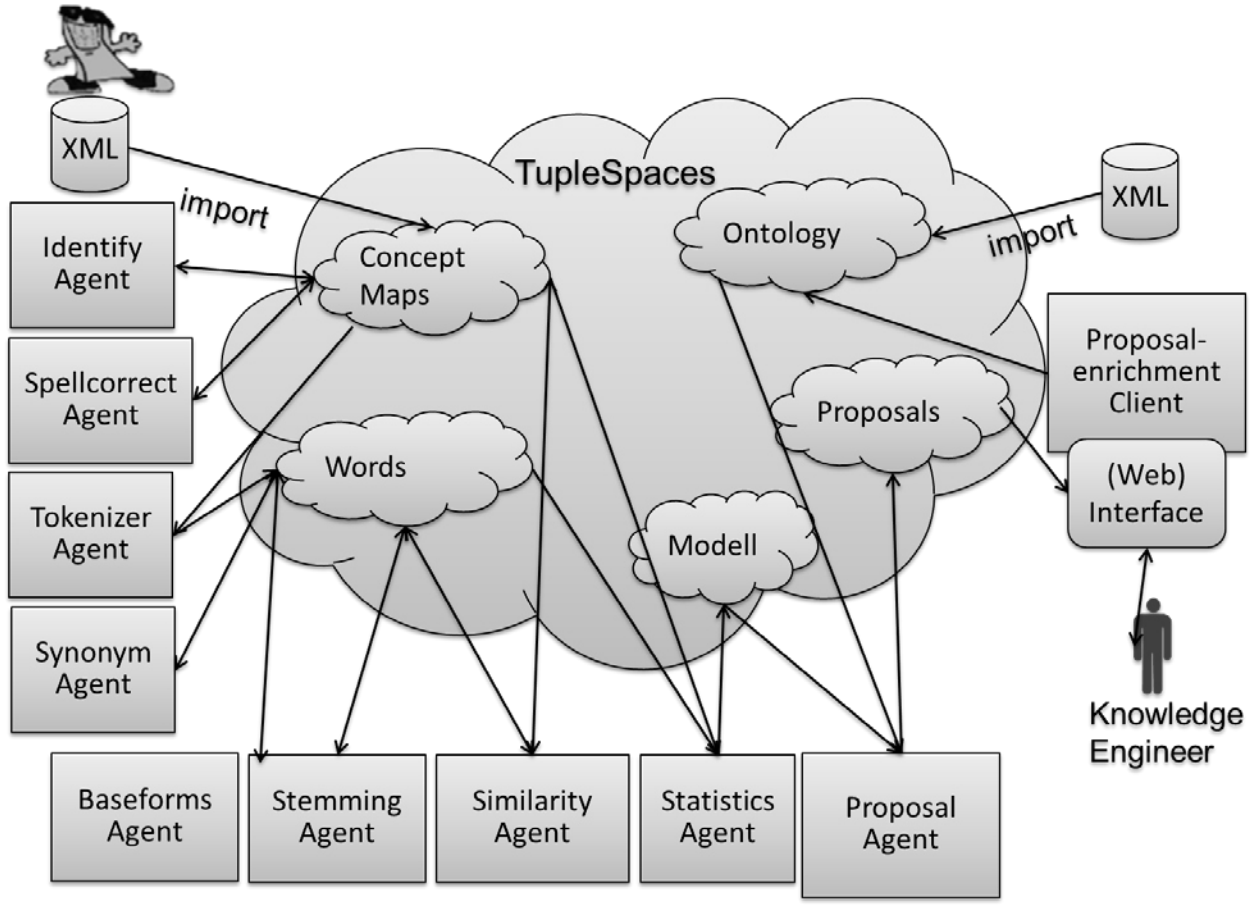

Abbildung 12: Architektur des Systems

4.2 Phasierung

Die Arbeit des Systems lässt sich in vier verschieden Phasen untergliedern, in denen unterschiedliche Softwareagenten zum Gesamtergebnis beitragen. Die erste Phase ist eine zweischrittige **Identifizierungs- und Korrekturphase**, in der die Zugehörigkeit der vorhandenen Knoten und Kanten zu Concept Maps ermittelt wird und die Wörter der Label der Konzepte korrigiert werden. Die zweite Phase ist eine **Wortprozessierungsphase**, in der die Wörter analysiert und aufbereitet werden. Hierzu werden sie tokenisiert. Für die Token werden dann Synonyme gesucht. Die Token werden auch gestemmt und es wird versucht Lemmata für die Token zu ermitteln. Die dritte Phase besteht aus zwei Stufen zum **Bestimmen des „statistischen" Modells**, in der zunächst Ähnlichkeiten zwischen allen Konzepten bestimmt werden und dann auf Basis der ermittelten Ähnlichkeiten Konzepte zu Konzeptcontainern zusammengefasst werden. Die vierte und letzte Phase ist die **Vorschlagsphase**, in der Vorschläge aus einem Vergleich des statistischen Modells mit der Ontologie generiert werden und dann vom Nutzer angenommen werden können.

4.2.1 Identifizierungs- und Korrekturphase

Damit die Daten ihrer Quelle zugeordnet werden können und ein leichtes Rückschließen auf die Zugehörigkeit von Kanten und Knoten zu Concept Maps möglich ist, wird ein Verfahren zur Identifizierung der Concept Maps angewandt.

Die Relationen der Concept Maps liegen nun als sogenannte Edges (Kanten) vor. Die Konzepte liegen als sogenannte Nodes (Knoten) vor. Diese werden nun zunächst einer Rechtschreibkorrektur unterzogen. Das geschieht, damit die Konzepte nicht jedes Mal, wenn sie benötigt werden, erneut korrigiert werden müssen.

4.2.1.1 Identifizierung

Die Daten liegen, wie in Abschnitt 3.4 bereits beschrieben, in einer Form vor, die weder einen Rückschluss auf die Herkunft zulassen, noch eine einfache Zuordnung der Relationen sowie der Konzepte zu den ursprünglichen Maps ermöglicht. Deshalb wird in einem ersten Schritt, nachdem die Maps durch einen Importer in den „conceptmaps“-Space geschrieben wurden, durch den Identify Agent die Herkunft der Knoten und Kanten bestimmt und eine Zuordnung zu den Herkunfts-Concept-Maps vorgenommen. Dazu werden, wenn sie noch vorhanden sind, die ursprünglichen XML-Dateien, in denen die Concept Maps einzeln gespeichert sind, eingelesen. Gleichzeitig werden die „conceptmaps“-Tupel aus dem Space genommen. Dann werden sowohl Kantentupel als auch Knotentupel mit den Dateien abgeglichen.

Für den Fall, dass keine Quelldateien vorliegen, werden für alle zusammenhängenden Gebilde aus Knoten und Kanten gemeinsame „Map“-IDs per MD5-Hash berechnet. Dieses Verfahren ermöglicht zumindest noch auf einfache Weise Rückschlüsse über zusammenhängende „Gebilde“. Dies müssen keine Concept Maps sein, da Concept Maps auch in fragmentierter Form vorliegen können, d. h., sie enthalten Konzepte oder Relationen, die keine Verbindung zu dem Rest der Concept Map haben. Da dies für die Vorschläge jedoch nur insofern einen Unterschied macht, als sich die Anzahl der Maps um die Fragmente erhöht, werden in der weiteren Betrachtung solche „Gebilde“ auch Concept Maps genant.

Als Ergebnis werden Tupel, wie in Abschnitt 3.4 vorgeschlagen, in den Space zurückgeschrieben, die als ersten Wert die ID der Quell-Concept-Map enthalten und als zweiten Wert die Information, dass es sich um einen XML-Import handelte. Dieses Format wurde so gewählt, damit es sich auch in anderen Umgebungen, wie z. B. dem FreeStyler, nutzen lässt und auch dort eine Aussage über die Herkunft zulässt. Ist der Identify Agent fertig, so schreibt er ein Tupel mit der Signatur string(„MapIdentification“), string(„done“) in den Commandspace.

4.2.1.2 Korrektur

Die Konzepte weisen, wie in Abschnitt 3.2.1 beschrieben, Schreib- und Tippfehler sowie Abkürzungen auf. Diese werden vom Spellcorrect Agent aufgelöst. Dazu werden in diesem, wenn er durch das Fertigstellungstupel des Identify Agenten geweckt wurde, verschiedene Techniken angewendet. Die Tupel der Knoten werden hierzu ausgelesen. In einem ersten Schritt werden

mittels Stringvergleichen und einfachen *Regulären Ausdrücken* Codierungsfehler behoben und unerwünschte Zeichen wie doppelte Leerzeichen und Zeilenumbrüche entfernt. Die Wörter werden normalisiert, das soll hier bedeuten, dass unerwünschte Großschreibung im Wort eliminiert wird. Dies betrifft zum Einen die vollständige Großschreibung eines Wortes und zum Anderen die Großschreibung der ersten beiden Buchstaben. Ersteres wird durch die fälschlicherweise gedrückte Feststelltaste hervorgerufen. Die Großschreibung der ersten beiden Buchstaben resultiert aus dem zu lange gedrückt Halten der Umschalttaste. Textverarbeitungsprogramme wie Microsoft Word oder Open Office korrigieren diesen letzten Fehler automatisch, ohne den Nutzer zu fragen. Zusätzlich wird angenommen, dass Abkürzungen nicht länger als fünf Buchstaben sind. Dann wird eine Korrekturliste, die Korrekturen und Auflösungen von Abkürzungen enthält, auf die einzelnen Wörter der Knotentupel angewandt. Wenn ein Tupel korrigiert wurde, dann wird es im Space mittels „update“ aktualisiert.

Ist dieser Vorgang für alle Knoten beendet, werden die Knotentupel erneut ausgelesen, damit im weiteren Verlauf mit den aktualisierten Werten gearbeitet werden kann. Die Knotentupel werden nun auf Tippfehler untersucht. Hierzu wird keine Liste benutzt. Die mit dem Verfahren erzielten Ergebnisse haben nach Beobachtung dazu geführt, dass die Liste für den ersten Schritt verkleinert werden konnte. Die im Folgenden beschriebenen Operationen werden immer auf die einzelnen Wörter der Knoten angewendet. Die Wörter werden in diesem auf n-Grammen basierenden Verfahren (Abschnitt 3.3.1.2) miteinander verglichen.

Zuerst wird die Auftretenshäufigkeit bestimmt. Wenn ein Wort in der Kollektion der Knoten öfter vorkommt, als ein vorher festgelegter Schwellwert (hier: 2), wird das Wort als richtig geschrieben interpretiert und dient als Basis für eine Korrektur der „ähnlich“ geschriebenen Wörter aus der Kollektion. Der Schwellwert, ab dem ein Wort als richtig geschrieben betrachtet wird, kann zur Optimierung der Ergebnisse vom Benutzer des Systems eingestellt werden (siehe Abbildung 16), denn es ist anzunehmen, dass mit Größe der Kollektion auch die Häufigkeit von auf gleiche Art falsch geschriebenen Wörtern steigt. Um die Anzahl der Vergleiche zu reduzieren, wurde angenommen, dass am Anfang eines Wortes nicht falsch geschrieben wird. Dadurch kann man den Vergleich auf Wörter mit demselben Anfangsbuchstaben reduzieren.

Eine häufige Falschschreibung am Anfang eines Wortes bezieht sich auf das versehentliche Großschreiben des zweiten Buchstaben, dass durch die Normalisierung im vorherigen Schritt bereits korrigiert wurde. Ist nun in der Kollektion ein Wort ermittelt, das Basis für eine Korrektur ist, werden für dieses Wort die n-Gramme gebildet. Das n wird in den Einstellungen festgelegt und ist auf 2 voreingestellt. Dann werden, in einer Schleife, für jedes Wort aus der Kollektion, dass denselben Anfangsbuchstaben hat, ebenfalls die n-Gramme ermittelt und zu den für beide Wörter ermittelten n-Grammen der DICE-Koeffizient bestimmt.

Eine Korrektur findet dann unter der Bedingung statt, dass der DICE-Koeffizient den voreingestellten, und vom Benutzer einstellbaren, Wert von 0,8 überschreitet, also die Wörter ähnlich sind, aber gleichzeitig der Koeffizient nicht gleich 1 ist, was bedeuten würde, dass sie

gleich sind. Eine weitere Bedingung ist, dass die Wörter die gleiche Endung haben, denn sonst würden Pluralendungen und Flexionen auf den ersten gefundenen Wert „korrigiert“. Mit dieser Bedingung bleiben zunächst alle Wortformen erhalten. Auch diese Korrekturen werden dann auf die Tupel angewandt, indem das korrespondierende Tupel im Space aktualisiert wird. Auch dieser Agent teilt seine Fertigstellung im Commandspace mit. Er nutzt dazu folgende Tupelsignatur: string(„SpellCorrect“), string(„done“).

Aus dem Label „Gletscherscmelze“ wird in diesem Schritt „Gletscherschmelze“ und aus „Versauerung der Oceane“ wird „Versauerung der Ozeane“. In Abbildung 13 wird dies im Zusammenhang mit den anderen Phasen, bis zum Similarity Agenten, nochmals illustriert.

4.2.2 Wortprozessierungsphase

Die Wortprozessierungsphase teilt sich in zwei Stufen. In einer ersten Stufe werden die Labels der Konzepte in Einzeltoken zerlegt und in den „Word“-Space geschrieben. In der zweiten Stufe werden für diese Token Synonyme, Stems und Baseforms bestimmt.

Abbildung 13 illustriert die Ergebnisse der einzelnen Schritte der Wortprozessierungsphase. Zum besseren Verständnis wird das Korrekturergebnis des Spellcorrect Agent mit aufgeführt. Spellcorrect Agent und Tokenizer Agent laufen exklusiv ab. Das bedeutet hier, dass die anderen Agenten noch nicht arbeiten, solange jeder dieser beiden seine Arbeit nicht vollendet hat.

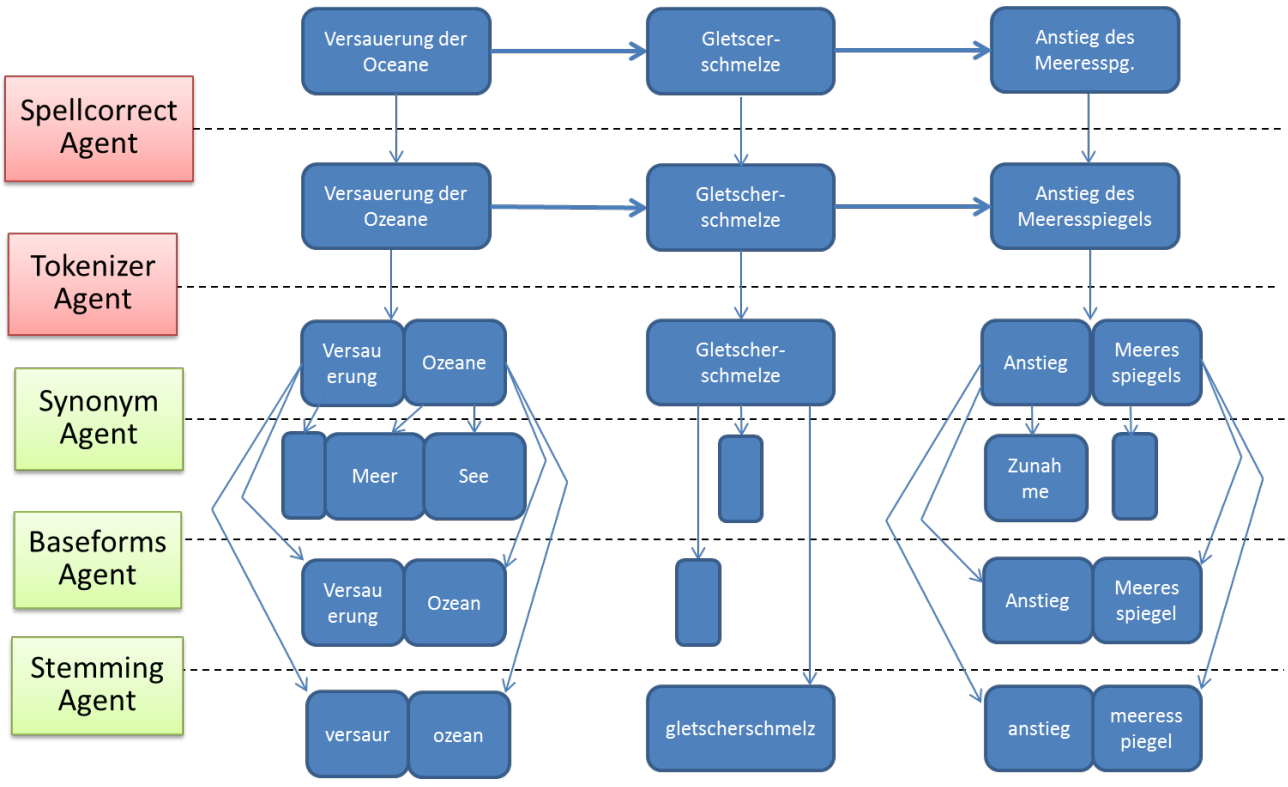

Abbildung 13: Ergebnisse der verschiedenen Stufen der Korrektur durch den Spellcorrect Agent und der Wortprozessierungsphase

Deshalb sind sie rot eingefärbt. Die grün eingefärbten drei Agenten arbeiten gleichzeitig auf den Token, die der Tokenizer Agent erstellt hat.

4.2.2.1 Tokenisierung

Um weiter mit den Wörtern der Labels zu arbeiten ist es sinnvoll, diese als eigene Tupel vorliegen zu haben. Der Tokenizer Agent nimmt hierfür eine Tokenisierung der Label vor, sobald er durch das Commandtupel des Spellcorrect Agent aufgeweckt wurde. Dazu werden zunächst listenbasiert die Stoppwörter entfernt und dann Wortweise die Label als Token in den Space geschrieben. Das Tupelformat dafür ist: string(„token“), string(Token), boolean(„false“), boolean(„false“), boolean(„false“). Die drei boolschen Werte geben den später laufenden drei Agenten dieser Phase die Möglichkeit, den Status der Bearbeitung in das Tupel zu schreiben. Ein weiteres Tupel mit der Signatur string(„lettering“), string(Token), string(Node-UID), string (Map-UID) wird geschrieben, um das zu diesem Token gehörende Originallabel identifizieren zu können. Um beim Generieren der Vorschläge, im späteren Verlauf, eine vereinfachte Bearbeitung der stoppwortbereinigten Label zu ermöglichen, wird noch ein drittes Tupel, diesmal je Label, geschrieben. Dieses Tupel enthält das von Stoppwörten befreiten Originallabel des Konzeptes und einen Verweis auf dessen Node-UID. Sein Format ist string(„aggregatedTokens“), string(Node- UID), string(stoppwortbefreites Label).

Wie man in Abbildung 13 sehen kann, werden aus dem Label „Versauerung der Ozeane“ die Token „Versauerung“ und „Ozeane“ und der aggregatedToken „Versauerung Ozeane“.

Der Tokenizer Agent schreibt ein Fertigstellungstupel mit folgender Tupelsignatur: string(„tokenization“), string(„done“).

4.2.2.2 Synonyme, Stems und Baseforms

Die drei Agenten der zweiten Stufe dieser Phase, der Synonym Agent, der Baseforms Agent und der Stemming Agent arbeiten gleichzeitig. Sie benötigen zwar dieselbe Grundmenge an Daten, aber ihre Ergebnismenge ist disjunkt und sie bauen ebenfalls nicht aufeinander auf. Um trotzdem sicherzustellen, dass keine vermeidbaren Berechnungen angestellt werden, nehmen sie das Tokentupel, das sie bearbeiten, aus dem „Words“-Space und schreiben, nachdem die Prozesse für dieses Tupel abgelaufen sind, das Tokentupel mit aktualisiertem Statuswert zurück. Der Synonym Agent nutzt hierfür die hinterste Stelle des Tokentupel, der Baseforms Agent schreibt seinen Status in den mittleren der drei boolschen Werte des Tokentupel und der Stemming Agent nutzt den vorderen boolschen Wert.

Der Synonym Agent nimmt sich ein Tokentupel aus dem „Words“-Space, in dessen Signatur an hinterster Stelle ein „false“ steht. Dann fragt er mit dem Token den Wortschatzdienst (siehe Abschnitt 2.9) ab und schreibt für jedes Synonym ein Tupel mit der Signatur string(„Synonym“) string(Token), string(Synonym). Für die Verbindung zum Wortschatzdienst wurde ein Connector zu dessen SOAP-Schnittstelle implementiert. Wie im Konzept bereits diskutiert, wird nicht für jedes Token ein Synonym geliefert. Um in diesem Fall eventuell doch noch Synonyme für ein

Token zu erhalten, wird dann Openthesaurus abgefragt. Hierzu wurde ein XML-Connector implementiert. Ist der Durchlauf für das Token erledigt, so wird das Tokentupel mit neuem Status in den „Words“-Space geschrieben.

Der Baseforms Agent geht analog zum Synonym Agenten vor. Er nutzt ebenfalls den im Rahmen dieser Arbeit implementierten SOAP-Connector, um Daten vom Wortschatzservice zu erhalten. Wird eine Grundform ermittelt, so wird ein Tupel mit der Signatur string(„Baseform“) string(Token), string(Baseform) in den „Words“-Space geschrieben. Unabhängig davon, ob ein Baseformtupel geschrieben wurde, wird der Status des Tokentupel auf „bearbeitet vom Baseforms Agenten“ gesetzt und in den Space zurückgeschrieben.

Der Stemming Agent ermittelt zu jedem Token dessen Stem. Hierzu wurde der Algorithmus des Porterstemmers (siehe Abschnitt 2.8.2) für das Deutsche implementiert. Die gestemmten Token werden mit der Signatur string(„stem“) string(Token), string(Stem) in den „Words“-Space geschrieben. Danach wird Tokentupel mit aktualisiertem Status in den Space zurückgeschrieben.

4.2.3 Bestimmen des „statistischen“ Modells

Das Erstellen des „statistischen“ Modells geschieht in zwei Stufen. Zuerst werden Ähnlichkeiten zwischen den Konzepten bestimmt. Diese Aufgabe übernimmt der Similarity Agent. In der zweiten Stufe werden dann Konzepte in Konzeptcontainern zusammengefasst und Auftretenshäufigkeiten von Konzepten und Kanten bestimmt, um eine Gewichtung von Kanten und Konzeptcontainern vorzunehmen.

Der Similarity Agent fängt an die Ähnlichkeiten von Labeln zu bestimmen, sobald alle Tokentupel von den drei Wortprozessierungsagenten aus der vorherigen Phase bearbeitet wurde, also sobald die drei Wahrheitswerte für alle Tokentupel true sind. Ebenso muss der Tokenizer Agent ein Fertigmeldungstupel geschrieben haben, damit wirklich alle Token der Kollektion bearbeitet werden. Zur Ähnlichkeitsbestimmung werden die stoppwörterbefreiten Label untereinander verglichen. Zunächst werden für diesen Vergleich für jedes solche Label die Stems, Baseforms und Synonyme aus dem Space gelesen. Label, die aus derselben Concept Map stammen, werden nicht miteinander verglichen, um weniger Vergleiche berechnen zu müssen. Ohnehin ist eine Mehrfachverwendung eines Konzeptes in einer Concept Map nicht sinnvoll und sollte deswegen auch nicht zu einer stärkeren Gewichtung führen.

Die Stems werden verglichen und es wird ermittelt, ob diese für die verglichenen Label gleich sind. Die Anzahl der gleichen Synonyme je Labelpaar wird ermittelt und ebenso die Anzahl der gleichen Baseforms. Des Weiteren wird sowohl LD als auch ED berechnet. Wenn die Ähnlichkeit zwischen zwei Labels die voreingestellten, aber vom Benutzer änderbaren, Werte von LD-Übereinstimmung = 80% und ED-Übereinstimmung = 80% unterschreitet oder mindestens 10 gleiche Synonyme (auch dieser Wert kann vom Nutzer eingestellt werden) haben, dann wird ein „Similarity“-Tupel in den „Modell“-Space geschrieben. Sein Tupelformat lautet: string(„similarity“), string(Konzept1-UID), string(Konzept2-UID), integer(LD), integer(ED),

boolean(gleicherStem?), integer(Anzahl gleicher Synonyme), integer(Anzahl gleiche Baseforms). Sind alle Vergleiche abgeschlossen, schreibt der Similarity Agent ein Tupel in den Commandspace, string(„Similaritys“), string(„done“) und geht wieder in den Wartemodus.

Auf dieses Commandtupel wartet der Statistics Agent. Dieser liest zunächst alle Ähnlichkeitstupel ein und sucht auf Basis der zuvor bestimmten Ähnlichkeiten nun die zusammengehörigen Konzepte. Dazu müssen alle Konzepte zusammengefasst werden, die im ersten Similaritywert gleich sind. Diese bilden dann einen Konzeptcontainer. Dann werden je Konzeptcontainer die abgehenden Kanten gesucht und deren Node-UIDs gespeichert, um diese in Konzeptcontainer ersetzen zu können. Danach werden die Konzepte behandelt, die in keinem Konzeptcontainer enthalten sind. Diese seien definiert als Konzeptcontainer mit nur einem Wert.

Nachdem die Kanten zu diesen gesucht wurden, werden die Ziele durch ihre korrespondierenden Konzeptcontainer ersetzt, sodass im Ergebnis ein Tripel aus Konzeptcontainer – Kante – Konzeptcontainer entsteht. Die Konzeptcontainer haben eine Anzahl an Unterkonzepten, die ihre Gewichtung bestimmt. Diese wird mittels des Tupels string(„ConceptCount“), string(Container-UID), integer(Gewichtung) in den „Statistics“-Space geschrieben. Die Anzahl der Kanten wird in die Statistik aufgenommen. Das Tupel hierfür ist: string(„EdgeCount“), integer(Kantenzahl). Auch für die Anzahl der Maps wir ein Tupel geschrieben. string(„MapCount“), integer(Mapanzahl) ist dessen Signatur. Das Array mit den Konzeptcontainer – Verbindung – Konceptcontainer - Tripeln wird nun abgeflacht, sodass Statistiktupel geschrieben werden, die folgendes Format haben: string(„ConceptC2ConceptC“), string(Startkonzeptcontainer-UID), string(Zielkonzeptcontainer-UID), integer(Anzahl „beruht auf“ Relationen), integer(Anzahl „bezieht ein“ Relationen), integer(Anzahl „erhöht“ Relationen), integer(Anzahl „hat Einfluss auf“ Relationen), integer(Anzahl „hat Verbindung zu“ Relationen), integer(Anzahl „ist ein“ Relationen), integer(Anzahl „verringert“ Relationen). Der Statistics Agent meldet eine Fertigstellung in den Commandspace. Tupelformat: string(„Statistics“), string(„done“)

4.2.4 Vorschlagsphase

Die Vorschlagsphase beginnt der Proposal Agent, wenn er das Statistikfertigmeldungstupel sieht. Es wird gegen die Ontologie verglichen und anhand von in dieser auftretenden bzw. nicht auftretenden Konzepten aus den Statistiktupeln Vorschlagstupel mit dem Format string(„ConceptC2ConceptC“), string(Startkonzeptcontainer-UID), string(Zielkonzeptcontainer-UID), integer(Anzahl „beruht auf“ Relationen), integer(Anzahl „bezieht ein“ Relationen), integer(Anzahl „erhöht“ Relationen), integer(Anzahl „hat Einfluss auf“ Relationen), integer(Anzahl „hat Verbindung zu“ Relationen), integer(Anzahl „ist ein“ Relationen), integer(Anzahl „verringert“ Relationen), integer(Start kommt vor), integer(Ziel kommt vor) in den „Proposals“-Space geschrieben. Auch werden für die Konzepte in der Ontologie bei den schon bekannten Synonymdiensten Synonyme abgefragt und ebenfalls in den Space geschrieben. Das Format ist string(„ProposalSynonym“), string(Original), string(Synonym).

Abbildung 14: Vorschläge im Benutzerinterface (Ansicht aus Illustrationsgründen gekürzt)

Diese Vorschläge können dann vom Benutzerinterface für den Benutzer des Systems sichtbar gemacht werden, wie in Abbildung 14 zu sehen und vom Benutzer in die Ontologie übernommen werden. Zum einfacheren Erkennen, der vom System vorgenommenen Gewichtung, sind die Vorschläge farblich sortiert. Die in Abbildung 14 gezeigten Vorschläge haben noch keine Verbindung zur Ontologie. Damit keine Fragmente in die Ontologie geschrieben werden, muss der Nutzer die Verbindung zu einem Konzept der Ontologie mit angeben. Dazu stehen die verschiedenen Relationen und eine Liste der Konzepte der Ontologie zur Verfügung. Durch einen Klick auf „Eintragen“ wird die Relation zum Eintragen vorgemerkt. So können mehrere Vorschläge vorgemerkt werden. Beim Klick auf „Übernehmen“ werden dann die markierten Vorschläge in die Ontologie übernommen.

In der Entwicklungsphase sind Benutzerinterface und Eintragungsvorgang stark zusammengewachsen, sodass sie nun als eine Einheit betrachtet werden können. Diese ist in der Architektur (Abbildung 12) als ProposalEnrichment Client notiert. Die Vorschläge werden nach Gewicht der Konzepte sortiert und in drei verschiedene Kategorien sortiert. Neue Relationen sind Start – Kante – Ziel – Verbindungen, die noch gar nicht in der Ontologie vorkommen. Der Nutzer kann dann festlegen, mit welcher aus der Ontologie stammenden Relation dieser Vorschlag an welchen in der Ontologie vorhandenen Knoten gehängt wird. Dann gibt es neue Konzepte, das bedeutet, entweder Start- oder Zielkonzept sind bereits in der Ontologie vorhanden und das neue Konzept wird an den vorhandenen Knoten der Ontologie angehängt. Für diese beiden Kategorien kann jeweils der Bezeichner der Relation, die die Konzepte verbindet, aus einem Dropdown-Menü ausgewählt werden, in dem ebenfalls die Auftretenshäufigkeit der jeweiligen Relation angegeben ist.

Die dritte Kategorie behandelt Synonyme für vorhandene Konzepte. Diese werden auf einer eigenen Seite dargestellt, wie Abbildung 15 illustriert. Dem Nutzer wird die Möglichkeit gegeben,

je vorhandenem Konzept die zugehörigen Synonyme aus einer Liste auszuwählen und in die Ontologie zu übernehmen. Ebenfalls kann der Nutzer weitere Synonyme für das Label eingeben.

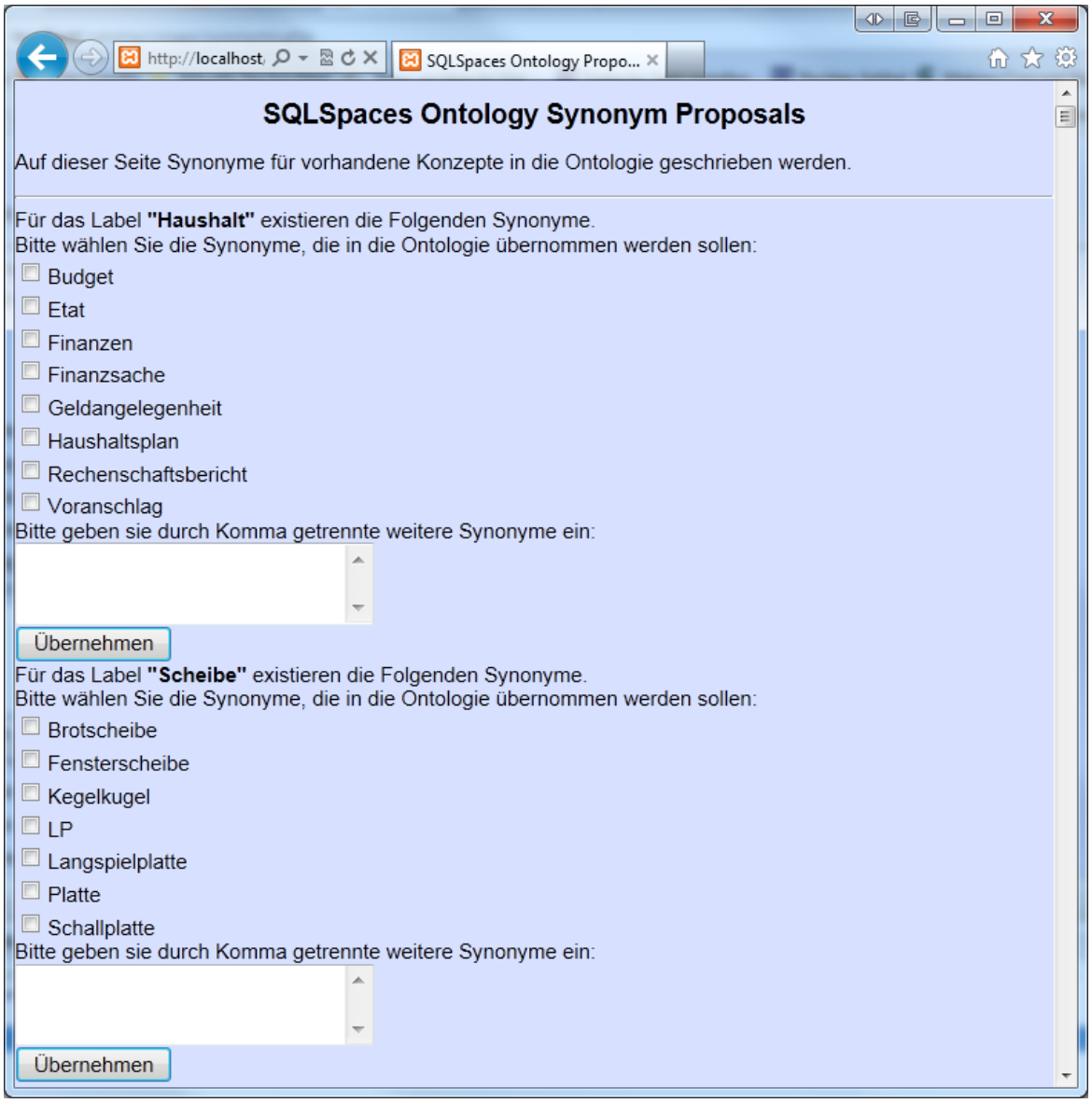

Abbildung 15: Synonymvorschläge für in der Ontologie vorhandene Konzepte

Für die Übernahme der Daten in die Ontologie muss berücksichtigt werden, dass zuerst immer die Bezeichner zu den Labels gesucht werden müssen, um ein Eintragen an der richtigen Stelle zu ermöglichen. Wird ein Vorschlag in die Ontologie übernommen, so wird er aus dem Vorschlagsspace gelöscht.

4.3 Einstellungen

Wie bereits erwähnt, gibt es die Möglichkeit verschiedene Einstellungen zu machen. Abbildung 16 zeigt, welche Einstellungen möglich sind. So kann man die Mächtigkeit der für die Korrekturen

verwendeten Zeichen-n-Gramme bestimmen, sowie den Auftretenshäufigkeitsschwellwert für den Fall, dass zwei Wörter als gleich angesehen werden, festlegen. Der DICE-Koeffizient ist der Korrektur Schwellwert, ab dem ein Wort zu seiner Korrekturentsprechung korrigiert wird.

Weitere Einstellungen können betreffend der Bildung des „statistischen" Modells gemacht werden. Die Abbildung 16 zeigt die voreingestellten Werte.

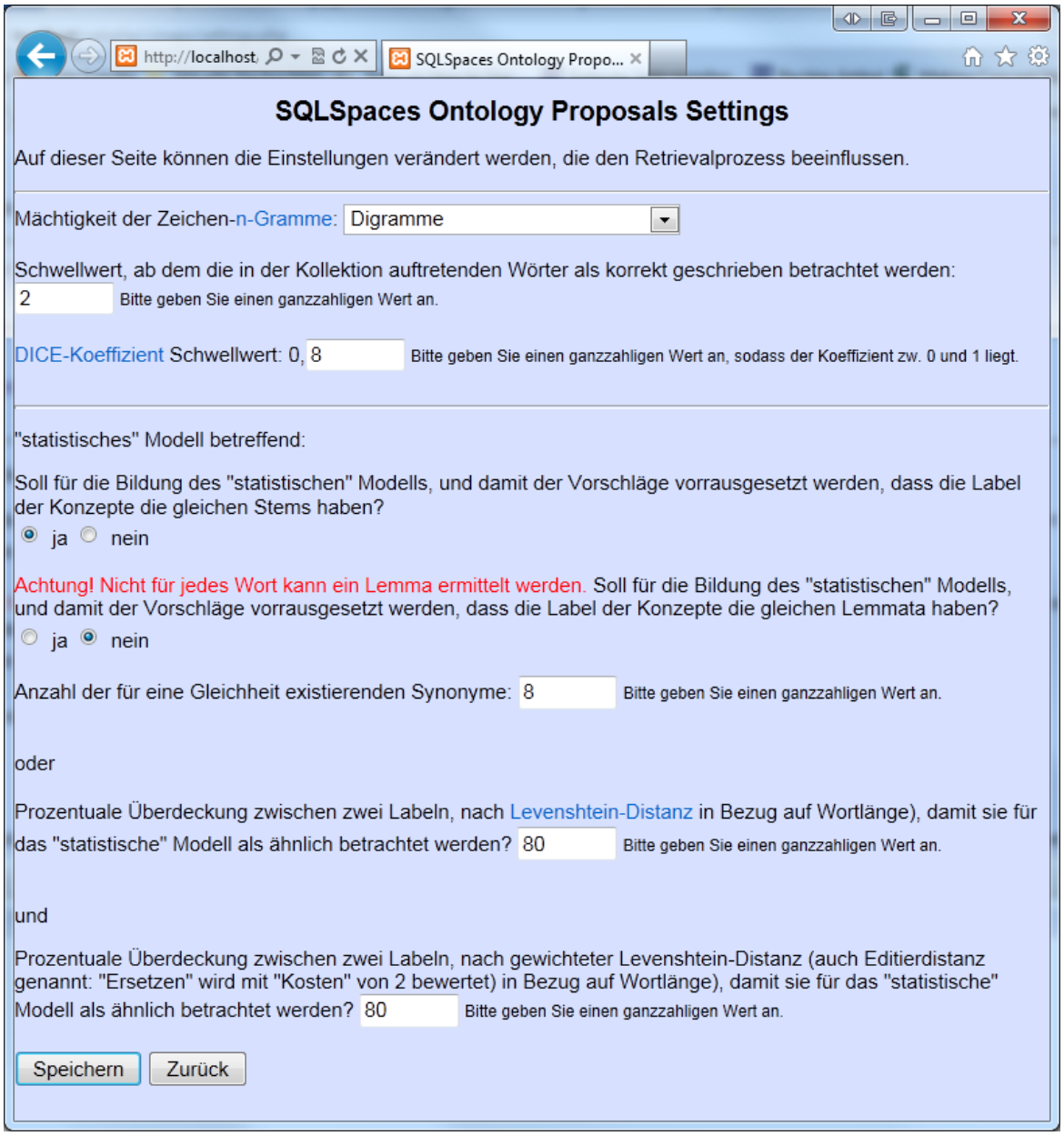

Abbildung 16: Einstellungsseite zur Beeinflussung des Retrievalprozesses

5. Zusammenfassung und Ausblick

5.1 Zusammenfassung

In dieser Arbeit ist ein System entwickelt worden, das einem KE oder DE die Möglichkeit gibt aus einer Menge von Concept Maps eine vorhandene Ontologie zu erweitern. Dieses Verfahren kann nicht automatisch stattfinden. Es muss teilautomatisch stattfinden, damit der KE die Anreicherung der Ontologie mit „Wissen“ verhindert, das nicht korrekt abgeleitet wird. Durch die Teilautomatisierung spart der Nutzer des Systems Zeit, die er in die weitere Verbesserung der Ontologie investieren kann.

Das System produziert Vorschläge, die es dem KE möglich machen, neue Konzepte und Relationen in die Ontologie zu übernehmen. In den generierten Vorschlägen lassen sich eine Reihe von sinnvollen Vorschlägen entdecken. Diese müssen nun nicht mehr vom KE zusammengetragen werden. Das interaktive Benutzerinterface stellt die Vorschläge in optisch ansprechender Weise und strukturiert dar. Das Benutzerinterface ermöglicht es dem KE, sinnvolle Vorschläge mit wenigen Klicks in die Ontologie aufzunehmen. Durch die optische Aufbereitung im Benutzerinterface und die Einfachheit des Benutzerinterface ist die Übernahme nicht nur einem KE vorbehalten, sondern auch durch einen DE möglich.

Es war auch sinnvoll, das System als Multiagentensystem mit TupleSpaces als Grundlage zu implementieren, da dadurch flexible Erweiterungsmöglichkeiten mittels weiterer Agenten unproblematisch gegeben sind.

5.2 Ausblick

Je höher der Grad der Teilautomatisierung der Ontologieevolution ist, desto weniger Zeit wird für die Evolution einer Ontologie benötigt. Das bedeutet, dass wenn man die Ergebnisse, die das Verfahren produziert, weiter verbessert, dann auch weniger Arbeitskraft zum Verbessern der Ontologie benötigt wird. Die Erweiterung könnte durch neue Agenten, die mehr Informationen bereitstellen können, umgesetzt werden. Diese Agenten können in unterschiedlichen Programmiersprachen verfasst sein.

Es ließe sich eine Verbesserung erreichen, wenn man bei dem im Abschnitt 3.3.1.2 beschriebenen n-Gramm-basierten Verfahren die Basis für die Vergleiche erhöhte. Es wäre möglich für die Beurteilung, ob ein Wort korrekt geschrieben wurde, eine größere Kollektion zugrunde zu legen. Diese Kollektion sollte domänenspezifisch sein. Beispielsweise könnten Zeitungs- oder Wikipediaartikel einer Domäne als Basis genutzt werden. Dann ließen sich drei Effekte erzeugen: Erstens sänke wahrscheinlich die Anzahl der falsch als korrekt erkannten Wörter und zweitens stiege die Anzahl der Wörter, die als Basis für eine Korrektur als korrekt geschrieben identifiziert würden. Drittens würden vermutlich mehr domänenspezifische Wörter erkannt und auch ggf.

korrigiert. So würden mehr automatische Korrekturen einer größeren Anzahl von Wörtern erreicht.

Wenn die Qualität der Synonyme gesteigert werden könnte, ließe sich hiermit der Arbeitsaufwand für den KE verringern. Dazu wären die Synonyme zu disambiguieren und einer Domäne zuzuordnen. Von MARCO GICCONE [55] ist in diesem Zusammenhang ein Verfahren entwickelt worden, das den Wortschatzdienst der Uni Leipzig (siehe Abschnitt 2.9) nutzt, um Wörter in ihrem Kontext zu disambiguieren. Diese Idee ließe sich in abgewandelter Form, denn bei gegebener Domäne ist der Kontext bekannt, nutzen, um vom Wortschatzservice nur als zur Domäne passend gekennzeichnete Wörter zu ermitteln. Es müsste ein Mapping der Domäne auf die über den Wortschatzservice zu beziehenden Themengebietsinformationen stattfinden.

Die Einsatzmöglichkeiten sind vielfältig, kann das Verfahren doch, wenn es mit anderen, domänenspezifischen Korrekturlisten angewandt wird, auch für andere Ontologien genutzt werden, die sich mit anderen Themengebieten befassen. So wäre es sinnvoll, das Verfahren zu testen und zu evaluieren. Dadurch könnte man ermitteln, wie sinnvoll die gemachten Vorschläge tatsächlich sind. Dies sollte, sowohl mit der genutzten Menge von Concept Maps, mit der Domäne „globale Erwärmung“, als auch mit anderen Concept Maps aus anderen Domänen geschehen.

Das System ließe sich durch eine Agent/Client-Kombination erweitern, welche die Entscheidungsfindung kleinschrittig erklären kann. Der Agent könnte für einen Vorschlag, anhand der vorhandenen Daten, ermitteln, welche Prozesse durchlaufen und welche Zwischenergebnisse produziert wurden. Der Client, der das User-Interface erweitert, kann diese Daten dann, auf Aufforderung durch den Benutzer, bereitstellen. Das würde die Akzeptanz vonseiten des Nutzers noch weiter steigern. [56]

Literaturverzeichnis

[1] I. Stracke, *Einsatz computerbasierter concept maps zur Wissensdiagnose in der Chemie: Empirische Untersuchungen am Beispiel des chemischen Gleichgewichts*. Münster [u.a.]: Waxmann, 2004.

[2] R. A. Gonzáles, N. Chen, and A. Dahanayake, *Personalized information retrieval and access: Concepts, methods and practices*. Hershey, Pa: IGI Global (701 E. Chocolate Avenue, Hershey, Pennsylvania, 17033, USA), 2008.

[3] G. Klyne and J. J. Carroll, *Resource Description Framework (RDF): Concepts and Abstract Syntax.* Available: http://www.w3.org/TR/rdf-concepts/ (2012, May. 09).

[4] M. Schädler, *Standardisierung von Web Services - Integration semantischer und sicherheitsbezogener Aspekte*. München: GRIN Verlag GmbH, 2007.

[5] D. Beckett, *RDF/XML Syntax Specification (Revised).* Available: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ (2012, May. 09).

[6] T. Berners-Lee, *An RDF language for the Semantic Web: Notation 3 Logic.* Available: http://www.w3.org/DesignIssues/Notation3.html (2012, May. 09).

[7] D. Brickley and R. Guha, *RDF Vocabulary Description Language 1.0: RDF Schema.* Available: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ (2012, May. 06).

[8] R. Volz, D. Oberle, and R. Studer, "Implementing views for light-weight Web ontologies," in *Database Engineering and Applications Symposium, 2003. Proceedings. Seventh International*, 2003, pp. 160–169.

[9] D. L. McGuinness and F. van Harmelen, *OWL Web Ontology Language Overview.* Available: http://www.w3.org/TR/2004/REC-owl-features-20040210/ (2012, May. 09).

[10] W3C OWL Working Group, *OWL 2 Web Ontology Language.* Available: http://www.w3.org/TR/owl2-overview/ (2012, May. 09).

[11] T. Bray and D. Hollander et al, *Namespaces in XML 1.0 (Third Edition).* Available: http://www.w3.org/TR/REC-xml-names/ (2012, May. 09).

[12] G. W. Leibniz, *Neue Abhandlungen über den menschlichen Verstand*, 1704.

[13] J. G. Walch, *Johann Georg Walchs philosophisches Lexicon*. Leipzig: Gleditsch, 1775.

[14] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220, 1993.

[15] T. H. Davenport and L. Prusak, *Working knowledge: How organizations manage what they know*. Boston, Mass: Harvard Business School Press, 2000.

[16] T. R. Gruber, "Ontology," in *Springer reference, Encyclopedia of database systems*, L. Liu and M. T. Özsu, Eds, New York, NY: Springer, 2009, pp. 1963–1965.

[17] D. L. McGuinness, "Ontologies Come of Age," in *Spinning the semantic Web: Bringing the World Wide Web to its full potential*, D. Fensel, Ed, Cambridge, Mass: MIT Press, 2003, pp. 171–196.

[18] A. Gómez-Pérez, M. Fernández-López, and O. Corcho, *Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web*: Springer, 2004.

[19] N. Guarino, *Formal ontology in information systems*: IOS Press, 1998.

[20] M. Gruninger and J. Lee, "Ontology applications and design: Introduction," *Communications of the ACM*, vol. 45, no. 2, 2002.

[21] A. Mädche, "Tutorial on Development and Application of Ontologies," in *Proceedings of ECML/PKDD*, 2001.

[22] H.-P. Schnurr, Y. Sure, R. Studer, and H. Akkermans, “On-To-Knowledge Methodology -- Baseline Version,” Institute AIFB, University of Karlsruhe 15, 2000.
[23] I. Sommerville, *Software-Engineering,* 8th ed. München ;, Boston [u.a.]: Pearson Studium, 2007.
[24] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López, “The NeOn Methodology for Ontology Engineering,” in *Ontology Engineering in a Networked World*: Springer-Verlag New York Inc, 2012, pp. 9–34.
[25] H. Pinto and J. Martins, “Ontologies: How can they be built?,” *Knowledge and Information Systems*, vol. 6, no. 4, pp. 441-464, 2004.
[26] M. Fernández, A. Gómez-Pérez, and N. Juristo, Eds, *METHONTOLOGY: From Ontological Art to Ontological Engineering*, 1997.
[27] A. Gomez-Perez and O. Corcho, “Ontology languages for the Semantic Web,” *Intelligent Systems,* vol. 17, no. 1, pp. 54–60, 2002.
[28] K. Umlauf, *Lexikon der Bibliotheks- und Informationswissenschaft*. Stuttgart: Hiersemann.
[29] J. D. Novak, *Learning, creating, and using knowledge: Concept maps as facilitative tools in schools and corporations,* 2nd ed. New York, NY: Routledge, 2010.
[30] S. Weinbrenner, A. Giemza, and H. Hoppe, “Engineering Heterogeneous Distributed Learning Environments Using Tuple Spaces as an Architectural Platform,” in *Advanced Learning Technologies*, 2007, pp. 434–436.
[31] D. Gelernter, *Generative communication in Linda*, 1983.
[32] *SQLSpaces download.* Available: http://sqlspaces.collide.info/download.html (2012, May. 24).
[33] V. Levenshtein, “Bounds for deletion/insertion correcting codes,” in *Information Theory, 2002. Proceedings. 2002 IEEE International Symposium on*, 2002, p. 370.
[34] William B. Cavnar and John M. Trenkle, “N-Gram-Based Text Categorization,” in *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 1994, pp. 161-175.
[35] A. M. Robertson and P. Willett, “Applications of n-grams in textual information systems,” *Journal of Documentation*, vol. 54, no. 1, pp. 48–67, 1998.
[36] J. J. Pollock and A. Zamora, “Automatic Spelling Correction in Scientific and Scholary Text,” *Commun. ACM*, vol. 27, no. 4, pp. 358–368, http://dblp.uni- trier.de/db/journals/cacm/cacm27.html#PollockZ84, 1984 (2012, May. 15).
[37] C. Shannon, *A mathematical theory of communication.* Available: http://www.bibsonomy.org/api/users/s-boutry/posts/72c23c0589536cc1a01c242f72f08c82 (2012, May. 09).
[38] W. J. Wilbur and K. Sirotkin, “The automatic identification of stop words,” *Journal of Information Science*, vol. 18, no. 1, pp. 45–55, 1992.
[39] R. Hausser, *Grundlagen der Computerlinguistik: Mensch-Maschine-Kommunikation in natürlicher Sprache*. Berlin: Springer, 2000.
[40] R. Hausser, “Drei prinzipielle Methoden der automatischen Wortformerkennung,” *Sprache und Datenverarbeitung*, no. 2, pp. 38–57, 1998.
[41] Xiang Ji and Hongyuan Zha, “Domain-independent Text Segmentation Using Anisotropic Diffusion and Dynamic Programming,” in *In Proceedings of SIGIR*: ACM Press, 2003, pp. 322-329.
[42] R. Kuhlen, *Experimentelle Morphologie in der Informationswissenschaft,* 1st ed. München: Verl. Dokumentation, 1977.

[43] H. H. Zimmermann, “Automatische Trunkierung beim Zugang zu textbezogenen Informationsbanken,” in *Schriften zur Informationswissenschaft*, vol. 2, *Wissensbasierte Informationssysteme und Informationsmanagement: Proceedings des 2. Internationalen Symposiums für Informationswissenschaft (ISI '91) zusammen mit dem 17. Internationalen Kolloquium für Information und Dokumentation*, H. Killenberg, R. Kuhlen, and H.-J. Manecke, Eds, Konstanz: Univ.-Verl. Konstanz, 1991, pp. 125–144.
[44] W. Stock, *Information Retrieval: Informationen suchen und finden*: Oldenbourg Verlag, 2006.
[45] J. B. Lovins, “Development of a Stemming Algorithm,” http://stinet.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0735504, 1968 (2012, May. 15).
[46] M. F. Porter, “An algorithm for suffix stripping,” *Program*, no. 14, pp. 130–137, 1980.
[47] V. Boehlke, "Hintergrundinformationen", E-Mail, (2012, May. 15).
[48] A. Mädche, *Ontology learning for the semantic Web,* 2nd ed. Boston [u.a.]: Kluwer, 2003.
[49] S. Weinbrenner, J. Engler, and H. Hoppe, “Ontology-Supported Scaffolding of Concept Maps,” in *Lecture Notes in Computer Science, Artificial Intelligence in Education*, G. Biswas, S. Bull, J. Kay, and A. Mitrovic, Eds.: Springer Berlin / Heidelberg, 2011, pp. 582–584.
[50] K.-U. Carstensen, *Computerlinguistik und Sprachtechnologie: Eine Einführung,* 3rd ed. Heidelberg: Spektrum Akad. Verl, 2010.
[51] openthesaurus.de, *Statistik.* Available: http://www.openthesaurus.de/statistics/index (2012, May. 09).
[52] M. F. Porter, *German stemming algorithm.* Available: http://snowball.tartarus.org/algorithms/german/stemmer.html (2012, May. 06).
[53] M. Consens and G. Navarro, *String Processing and Information Retrieval: 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, November 2-4, 2005: Proceedings*: Springer, 2005.
[54] K. S. Gaßner, *Diskussionen als Szenario zur Ko-Konstruktion von Wissen mit visuellen Sprachen*, 2003.
[55] M. Giccone, P. Gonsior, E. Sapmaz, and E. Widera, *Disambiguierung - Lösen von Mehrdeutigkeiten*. JW Goethe-University Frankfurt am Main, Dept. of Computer Science, 2006.
[56] O. Everling, Ed, *Certified rating analyst*. München ;, Wien: Oldenbourg, 2008.

Anhang

269 Konzepte

*->menschliche Ernährung, 02, 18 bis 59 Zentimeter, 18-59 cm Ende 2100, 2°-Ziel, ABbrechen großer Eismassen, Abholzung, Abholzung der Wälder, Abholzung und Landwirtschaft, Abohlzung, Abschmelzen der Eismassen, Abstrahlung von Wärmestrahlen, Agrar-/Viehwirtschaft, Analyse des
 Klima, andere Gase, Anstieg <2°C, Anstieg d. Durchschnittstemp., Anstieg d. Meeresspiegels, Anstieg der durchsch. Tem., Anstieg der Durchschnittstemp., Anstieg der Meeresspiegel, Anstieg der Temperatur, Anstieg der
 Temperatur, Anstieg des
Meeresspiegels, Anstieg des Meerespiegels, Anstieg des Meeresspiegels, Anstieg des Meeresspielgels, Anstieg des Wasserspiegels, Anstieg Durchschnittstemp., Anstieg Durchschnittstemperatu, Anstieg Meeresspiegel, Anstieg v. Wärmespiegel, Anstieg von CO2, Anstieg von Kohlendioxid, Anstieg von Treibhausgasen, anthropogene Erwärmung, anthropogener Treibhauseffekt, Arten in Existenz gefährdet, Artensterben, atmosphärischeCO2Konzentration, Aufnahme von C02, Aufnahme von CO2Emissionen-1/3, Aufnahmefähigkeit der Ozeane, Aufnahmefähigkeit der
 Ozeane, Aufnahmefähigkeit, Landoberflä, Auswirkung auf die Umwelt, Auswirkung auf Nahrung, Auswirkung auf Nahrungskette, Auswirkungen auf die Umwelt, behindert Kalkbildung, behindert Knochen- und, bis 2100 ca. 7m, bis Ende 21. Jhd: 18-59cm, Brennstoffe, CO2, CO2 Emissionen, CO2 zertifikate, CO2, Methan, Lachgas, CO2,Methan,Lachgas, Co2-Ausstoß, Co2-Budget, CO2-Emissionen, CO2-Zertifikate, CO_2 Zertifikate, CO_2 Zetifikate, CO², CO² Erhöhung, CO²-Budget, den Meeresspiegel, Die globale Erwärmung, die Kalkbildung, durch den Menschen, Durchschnittstemperatur, Durchschnittstemperaturanstieg, Einmischung aus der Politik, Erdatmosphäre, Erdatmosphäre reichert Gase an, erhöhter Meeresspiegel, erhöhter Niederschlag, Erhöhung, Erhöhung d. Durchschnittstemp., Erhöhung der Temperatur, erhöte CO2 Emissionen, Ernährung, Erwärmung, Erwärmung des Festlands, erwärmung von 1,1 - 6,4 °C, Erwärmung von Ozeanen+ Böden, erwartete Erwärmung 1,1-6,4°C, Europa, exestreme Wetterereignisse, Existenz der Meerestiere, extr. Wetterereignisse, Extreme (Wetter-)ereignisse, Extreme Wetterereignisse, extreme Wetterereignisse, extremere Wetterereignisse, extremere Wirbelstürme, Folgen, fossile Brennstoffe, fossile Brennstoffe verbrennen, Fossile brennstoffverbrennung, Freigabe von Kohlendioxid, Für die Umwelt, Gefahr für Leben im Meer, Gefahr für Meeresbewohner, Gefahr für Meerestiere, Gefahr für menschl. Ernährung, Gefährdung der Meeresbewohner, Gefährdung der Meerestiere, Gefährdung Meereslebewesen, Gefahren für das Leben im Meer, Gefärdung des Menschen, Gegenwirken der Menschen, geringer in südlichen Ozeanen, Geschwindigkeit, Gletscher, Gletscherschmelze, Gletscherschmelzen, Gletscherschmelzung, Gletscherschmz/Meereisrückgang, Gletscherscmelze, glob. durchsch.temp. 2°C, GLOBALE ERDERWÄRMUNG, Globale Erwärmung, globale Erwärmung (Folgen), globale Erwärmung verringern, Globale Erw\E4rmung, Globale Ewärmung, globale Klimaerwärmung, gravierende Folgen, Hitzewelle, Hitzewellen, Hitzewellen & Niederschläge, Hitzwellen, hohe co2-Emission, höhere Durschnittstemperatur, International, IPCC, IPCC/Weltklimarat, IPPC, Kalkbildung, kaum abschätzbare Folgen, Klimapaket, Klimapolitik, Kohlendioxid, Kohlenstoffdioxid, Korallenriffe sterben, Korallensterben, Lachgas, Land- u. Viehwirtschaft, Land- und Viehwirtschaft, Land- Viehwirtschaft, Land-und Viehwirtschaft, Landwirtschaft, Leben der Meerestiere, leben im Meer, Lösungsansätze, Lösungsversuche, Maßnahmen, Meere nehmen CO2 auf, Meereseis, Meeresspiegel, Meeresspiegelanstieg, mehr extreme We
ttererereignisse, mehr extreme Wettereignisse, mehr extreme Wetterereignisse, Mensch, Menschen und Tiere, menschengemachte Erwärmung, Menschliche Einwirkung, menschliche Ernährung, menschlicher Faktor, menschliches Einwirken, Menschliches Enwirken, Menschliches
 Einwirken, meschnl. Nahrungskette, Methan, Methan Erhöhung, Methan und Lachgas, nachhaltige Temperaturenkung, Nahrungakette, Nahrungskette, Nahrungskette im Meer gefährd., Naturereignisse, natürlicher Traubhauseffekt, natürlicher Treibhauseffekt, neg Auswirkung auf Nahrungsket, neg. für Nahrungskette, negativ für unsere Nahrung, negative Auswirkung, Nahrungsk, negative Auswirkungen, Niederschlag, Niederschläge, Niederschlagsmuster, polare Eisdynamik, politik, politische Maßnahmen, Rückgang des Meereises, Rückkopp
lung, Rückkopplungsmechanismus, saure Ozeane, Saurer Regen/Ozeane, Schalenaufbau der Meerestiere, schlimme ausw. auf Umwelt, Sekundäreffekte, Senkung des Treibhausgases, stark in nördlichen Breiten, starke Erderwärmung, starke Nie-
derschläge, starke Niederschläge, starke Niederschl\E4ge, strake Niederschläge, Temparatur anstieg, Treibhauseffekt, Treibhauseffekt durch CO2, Treibhauseffekt/-Gase, Treibhausgas, Treibhausgas Kohlendioxid(co2), Treibhausgasausstoß reduzieren, Treibhausgase, tropische Wirbelstürme, tropische Wirbelstürme extreme, unterbr. Nahrungskette, Ursache, Ursachen, verändert Niederschlagsmuster, veränderte Niederschlagsmuster, verändertes Niederschlagsmuste, Verbrennen foss. Brennstoffe, Verbrennen fossiler Brennstoff, Verbrennen fossilerBrennstoffe, Verbrennen von Brennstoffen, Verbrennung fos. Brennstoffe, Verbrennung foss. Brennstoffe, Verbrennung fossiler Sto
ffe, Verbrennung fossiler Stoffe, Versauerrung der Ozeane, Versauerte Ozeane, Versauerung der Ozeane, Versauerung des Meeresspiegels, Versauerung des Ozean, Versauerung des Ozeans, Verschwinden Meereises in Arkt, verstärkte Gletscher-
schmelze, Verstärkte Gletscherschmelze, verstärkter*

*Treibhauseffekt, Verstärkung nat. Treibhauseff., Verstärkung Treibhauseffekt, Verst\E4rkung des Treibhausefkt., viele Disskusionen, vom Menschen verschuldet, Wärmeabstrahlung, wärmen
Klima, warmes Klima, Wärmestrahlung, Wärmestrahlung ins All, Weltklimarat, Weltumfassende Abholzung, weltweit, Wetterextreme, Wirbelsturm, Wirbelstürme, Wirbelstürme/Hitzewellen, Zahl starker niederschläge, Ziel: max 2° Anstieg, Ziele, Zielsetzung, Zwei- Grad-Ziel*

Tabelle 3: 269 "unterschiedliche" Konzepte

Porter Stemmer für die deutsche Sprache

Betrachte die folgenden Buchstaben als Vokale: *a, e i, o, u, y, ä, ö, ü*
Ersetze *ß* durch *ss*, und wenn *u* oder *y* zwischen Vokalen auftreten, dann schreibe sie groß. Stelle R1 und R2 ein. Passe R1 so an, dass davor mindestens 3 Buchstaben stehen.
Eine gültige *s-Endung* ist: *s, d, f, g, h, k, l, m, n, r, t*
Eine gültige *st-Endung* ist: *s, d, f, g, h, k, l, m, n, t*
Dann befolge die Schritte 1-3:

Schritt 1:

Suche das längste der folgenden Suffixe:

(a) *em, ern, er* (b) *e, en, es*
(c) *s* (wenn zuvor eine gültige s-Endung steht)

und lösche es, wenn es in R1 liegt. Wenn Gruppe (b) matcht und zuvor ein *niss* steht, lösche das letzte *s*.

Schritt 2:

Suche das längste der folgenden Suffixe:

(a) *en, er, est*
(b) *st* (wenn zuvor eine gültige st-Endung steht, vor welcher mindestens 3 Buchstaben stehen.)

und lösche es, wenn es in R1 liegt.

Schritt 3:

Suche nach dem längsten der folgenden Suffixe und befolge die zugehörige Anweisung:

end, ung	löschen, wenn es in R2 liegt. Falls zuvor ein *ig* steht: lösche dieses *ig*, falls davor kein *e* steht.
ig, ik, isch	löschen, wenn es in R2 liegt und kein *e* davor steht.
lich, heit	löschen, wenn es in R2 liegt. Falls davor ein *er* oder *en* steht lösche dieses, es in R1 liegt.

keit	löschen, wenn es in R2 liegt. Falls davor ein *lich* oder *ig* steht, dann lösche dieses, wenn es in R2 liegt.

Zum Schluss werden U und Y wieder klein geschrieben und die Umlautungen entfernt.

Anleitung zum Installieren und Ausführen

1. Kopieren Sie den Ordner „Portable-VirtualBox“ lokal auf Ihren PC.
2. Führen Sie die darin enthaltene Datei „Portable-VirtualBox.exe“ aus.
3. Wählen Sie die VM „bachelorarbeit“ aus und klicken Sie auf starten.

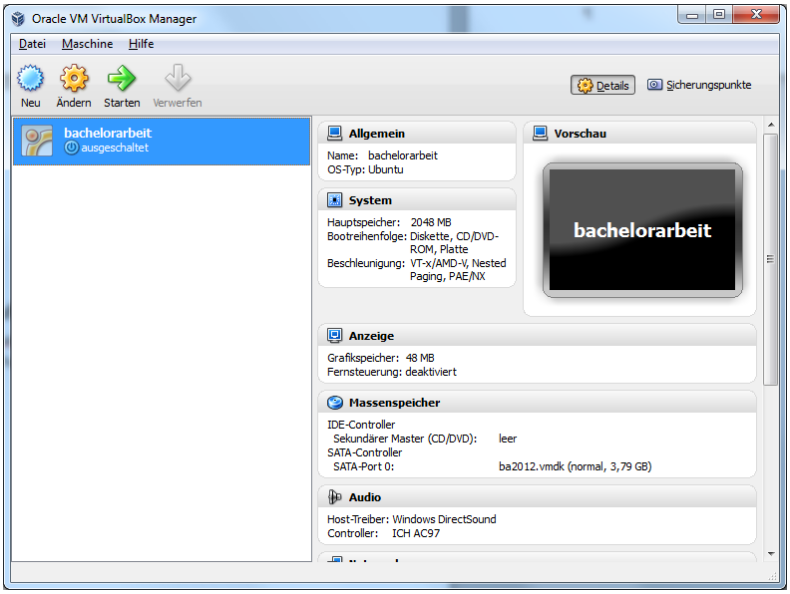

Abbildung A 1: Portable VirtualBox

4. Nachdem dem Start der VM öffnet sich ein Browserfenster und Sie werden um 15 Sekunden Geduld gebeten, damit alle restlichen, benötigten Komponenten starten können. Danach werden Sie zur Vorschlagsliste weitergeleitet. (Abbildung A 2)

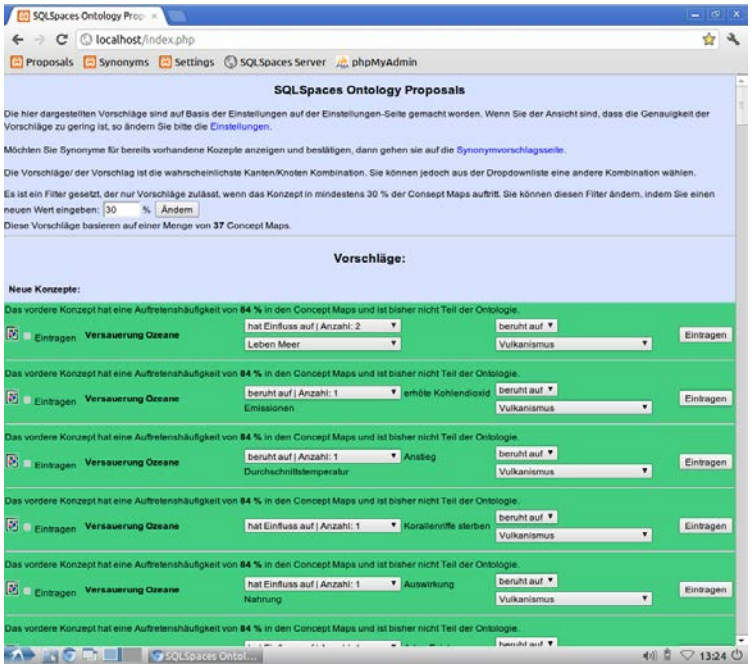

Abbildung A 2: Browser mit Lesezeichen für die benötigten Anwendungen

Falls diese noch nicht erscheint, so drücken Sie bitte den „neu Laden“-Button des Browsers.

5. Soll der Prozess der Vorschlagsfindung erneut stattfinden, so starten sie bitte „phpMyAdmin“ aus der Lesezeichenleiste des Browsers. Löschen Sie dann die Tabelle „sqlspaces“ und erstellen sie diese neu.
6. Danach starten Sie die VM neu.
7. Nach dem Neustart steht im Browserfenster folgende Anleitung zum weiteren Vorgehen:
    i. Starten Sie „LXTerminal“ vom Desktop
    ii. Geben Sie den Befehl „. loadSystem.sh“ ein
    c. Möchten Sie die Einstellungen verändern, so wählen Sie nun im Browser das „Settings“-Lesezeichen.
    d. Geben Sie den Befehl „. runSystem.sh“ ein

    Die beiden Befehle sind die einzigen in der Befehlshistorie, sodass sie bequem per „Pfeil nach oben“ ausgewählt werden können.
8. Ein kompletter Durchlauf dauert, je nach Geschwindigkeit des Systems, mit den gegebenen Concept Maps etwa 30 Minuten.
9. Danach können Sie, durch klicken auf das „Proposals“-Lesezeichen die Ansicht aufrufen, die in Abbildung A 2 zu sehen ist.
10. Möchten Sie Synonyme, zu in der Ontologie vorhandenen Labels, bestimmen, so klicken Sie auf das „Synonyms“ Lesezeichen.

Falls auf dem Linux-System Änderungen vorgenommen werden werden sollen, benötigen Sie das Passwort „ba2012“ für den Benutzer „bachelorarbeit“. Dieses ist als Textdatei auch auf dem Desktop abgelegt.

DOI: 10.17185/duepublico/71846
URN: urn:nbn:de:hbz:464-20200623-143135-1