Bachelorarbeit Lumin

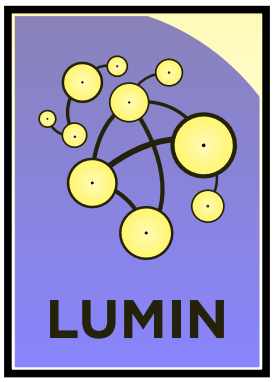
Wissensextraktion aus im Internet verfügbaren Informationsquellen

Kevin Gaunt, Tobias Löffel & Beat Weber

Betreuer: Prof. Dr. Josef Joller

Hochschule für Technik Rapperswil

12. Juni 2009



Abstract

In dieser Bachelorarbeit wird eine Lösung zur automatischen Erkennung von Ontologien aus verschiedenen, im Internet verfügbaren, Quellen präsentiert. Basierend auf einigen vorgegebenen Typdefinitionen, sowie deren Relationen zueinander, kann, durch den Einsatz von Data-Mining, Knowledge-Extraction und Clustering Verfahren, ein Graph von Begriffen über diese Typen aufgespannt werden. Als Referenzquelle für die Wissensbeschaffung dienen die Daten der freien Online-Enzyklopädie Wikipedia. Besondere Schwierigkeit ist es, die Anzahl falsch zugeordneter Begriffe klein zu halten, um so die unvermeidbare Korrekturarbeit für den Menschen möglichst zu minimieren. Ausserdem wird eine Architektur samt Implementation zur Speicherung von graph-basierten Daten vorgestellt und die darin enthaltenen Informationen grafisch aufbereitet und visualisiert. Die einzelnen Software-Komponenten bilden zusammen ein verteiltes System, damit beliebig viele Rechner ihre Leistung zur Verfügung stellen können.

Inhaltsverzeichnis

Teil I

Übersicht

Kapitel 1

Einführung

1.1 Zweck

Dieses Dokument beschreibt die, an der Hochschule für Technik Rapperswil (HSR) entwickelte, Bachelorarbeit Lumin und deren Ergebnisse.

1.2 Gültigkeitsbereich

Die Gültigkeit dieses Dokuments beschränkt sich auf die Bachelorarbeit Lumin, welche im Frühlingssemester 09, vom 16. Februar 2009 bis 12. Juni 2009, durchgeführt wurde.

1.3 Referenzen

Für eine Liste aller verwendeten Dokumente siehe Verzeichnisse auf Seite 114

1.4 Beteiligte Personen

Die Arbeit wurde von den drei Informatik-Studenten Kevin Gaunt, Tobias Löffel und Beat Weber konzipiert und durchgeführt. Betreuender Dozent war Prof. Dr. Josef Joller.

1.5 Übersicht

Im ersten Teil des Dokuments wird die Aufgabenstellung erläutert, sowie eine allgemeine Beschreibung der Bachelorarbeit und deren Resultate dargelegt. Im zweiten Teil werden die technischen Ergebnisse im Detail erläutert. Der dritte Teil umfasst die Software-Projektdokumentation. Im letzten Teil finden sich weiterführende Informationen und Referenzen.

Kapitel 2

Aufgabenstellung

In dieser Bachelorarbeit soll erforscht werden, inwiefern sich semantische Informationen aus vorhandenen Informationsquellen des World Wide Webs extrahieren lassen. Ziel soll sein, ein System zum Aufbau eines Wissensnetzes zu entwickeln und dessen Daten in nützlicher Weise zu visualisieren. Der Umfang der Arbeit umfasst zusätzlich die Architektur des Systems und einer geeigneten Storage Solution. Dabei sollen sich die Studenten mit den Gebieten 'Information Extraction', 'Text Mining' und 'Clustering' (und deren Algorithmen) vertraut machen. Einige Erfahrungen, insbesondere beim Crawling von Webinhalten, können aus der Studienarbeit 'Ziirp' von Kevin Gaunt und Tobias Löffel gewonnen werden.

Mögliche Anwendungen liegen in der Erstellung alternativer Suchmaschinen, die auf den erstellten semantischen Informationen operieren. So wäre es beispielsweise möglich, alle Städte der Schweiz anzuzeigen, welche einen Löwen im Wappen haben, ohne dass dafür eine spezifische Informationsseite indexiert werden muss. Denkbar wäre ausserdem, diese Daten über eine Schnittstelle anderen Anwendungen, besonders nachfolgenden Bachelorarbeiten, zur Aus- und Bewertung zugänglich zu machen.

Bei der Informationsbeschaffung ist es den Studenten freigestellt, ob sie sich auf bereits bestehende Ontologienetze stützen wollen oder ein komplett eigenes Netz aufbauen möchten. Für die Implementierung soll mit Vorteil die Programmiersprache Java verwendet werden.

Kapitel 3

Management Summary

3.1 Ausgangslage

Während das Internet für uns Menschen zu einer unabdingbar wichtigen Quelle für Informationen aller Art wurde, kann der Computer nichts mit dem, auf mittlerweile fast 29 Milliarden Webseiten[1] publizierten Wissen, anfangen. Zu unstrukturiert und komplex ist unsere Sprache, als dass eine Maschine daraus Informationen gewinnen könnte. Genau aus dieser Problemstellung ergab sich die Geburtsstunde des Semantischen Netzes, welches der Erfinder des Internets, Tim Berners-Lee, schon 1999 wie folgt beschrieb:

> "I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize." - Quelle: `http://en.wikipedia.org/wiki/Semantic_Web`

Um die, für den Menschen geschriebenen, im Internet publizierten Artikel für den Computer erfassbar zu machen, muss man diese Informationen in eine abstraktere Struktur umwandeln. Während sich das Gebiet der Computerlinguistik mit der Analyse der Sprache als solches beschäftigt, wird in dieser Arbeit ein anderer Ansatz gewählt, auf den in den folgenden Abschnitten, noch eingegangen wird.

3.2 Zielsetzung

Hauptziel der Arbeit ist es, aus für uns Menschen verständlichen Texten, genügend Informationen zu extrahieren, um es dem Computer zu ermöglichen, einzelne verwandte Begriffe einander zuordnen zu können. Als Beispiel soll erkannt werden, dass die Stadt Zürich mehr mit der Schweiz zu tun hat, als mit dem Vereinigten Königreich. Ein für uns Menschen denkbar einfaches Unterfangen, für den untrainierten, kontextfreien Computer aber, bereits eine grosse Herausforderung.

Ein weiteres Ziel ist die Speicherung dieser extrahierten Informationen in einer Datenbank. Schnell wird klar, dass sich die dabei aufbauende Datenmenge in sehr grossen Sphären bewegen wird. Hier ist also besonders auf eine gut skalierbare, schnelle Architektur zu setzen.

[1] `http://www.worldwidewebsize.com`, Stand: 15.05.2009

Schliesslich sollen die, in dieser Arbeit erzielten Ergebnisse, auch visuell für den Menschen sichtbar gemacht werden. Dies ist besonders für die Nachbearbeitung bzw. Korrektur der automatisch beschafften Informationen nützlich. Ausserdem stellt sich die schwierige Herausforderung, wie man solch grosse Datenmengen überhaupt noch übersichtlich präsentieren kann.

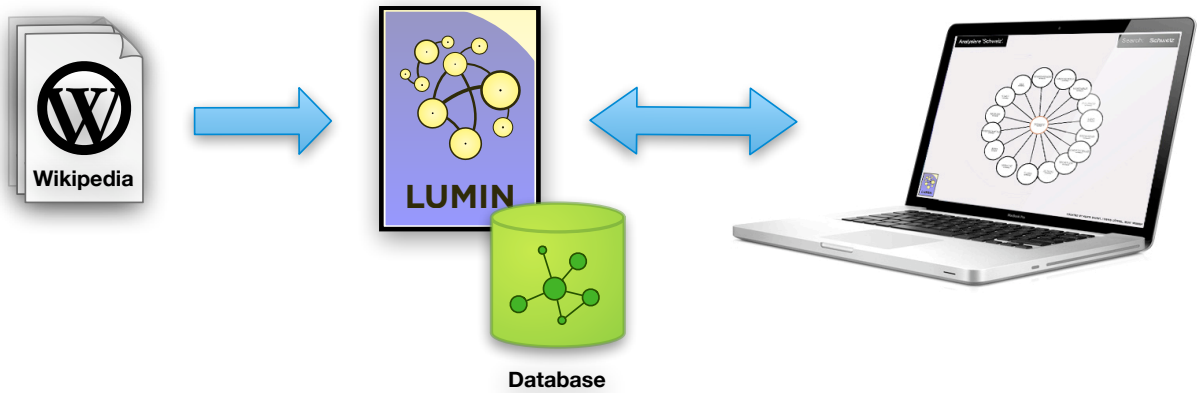


Abbildung 3.1: Vereinfachter Überblick über das System

3.3 Vorgehen

Gerade die benötigte Einarbeitung in die, für die Projektmitglieder, neuartigen Themengebiete, barg ein gewisses Risiko für den Projekterfolg. Deshalb wurde entschlossen, bereits früh im Projektverlauf einen Prototypen zu entwickeln, der die Machbarkeit der Arbeit nachweisen sollte. Aus den, bei dieser Entwicklung gewonnenen Erkenntnissen, lässt sich der Aufwand und Umfang der geplanten Arbeit danach besser abschätzen und planen. In der darauf folgenden Iteration wurde die Architektur des Prototypen überarbeitet und ein Refactoring eines Grossteils des bestehenden Programmcodes durchgeführt.

Während der erste Prototyp bereits eine einfache Datenhaltung aufwies, galt das Augenmerk der zweiten Iteration auf deren Ausbau. Speziell hervorzuheben, ist der Einsatz der graph-basierten Datenbank Neo4j[2], welche sich von den bekannten relationalen Datenbanken wie Oracle[3] oder MySQL[4] unterscheidet.

In der dritten und gleichzeitig letzten Iteration wurde schliesslich die Entwicklung des User Interfaces in den Vordergrund gerückt. Dabei wurde entschieden, möglichst die Struktur der Datenbank zu visualisieren. Neben der Darstellung einzelner Subgraphen, sollte ausserdem auch eine Suchfunktionalität unterstützt werden.

Der wissenschaftlichen Art des Projektes kam ein agiles Vorgehen sehr zu gute. Häufige Teamsitzungen und Besprechungen (angelehnt an Scrum [4]) sorgten für eine ständige Fortschritt- und Qualitätskontrolle. Gerade in einer gut abgestimmten Dreiergruppe, waren ein hohes Mass an Flexibilität und Pragmatismus gefragte Eigenschaften eines jeden Projektmitgliedes.

[2] http://neo4j.org/
[3] http://www.oracle.com/
[4] http://www.mysql.com/

Der Bachelorarbeit ist ein einwöchiger Workshop zur Themenfindung vorausgegangen, dessen Resultate sich oben in der Zielsetzung wiederfinden. Der für die Umsetzung verfügbare Zeitrahmen betrug die üblichen 16 Wochen, wovon 14 aufs Sommersemester fallen und anschliessende 2 Wochen Vollzeitarbeit.

3.4 Ergebnis

Zum Ende der Bachelorarbeit wurden die gesteckten Ziele allesamt erfüllt. Es wurde ein System entwickelt, welches mit einer Fehlerquote von weniger als 5% bei rund 10% vorgegebenen Daten, kontextuell verwandte Begriffe, einander zuordnen kann. Besonders die Beschränkung auf die Enzyklopädie Wikipedia, war besonders wertvoll. Versuche mittels Google-Suchresultaten Informationen zu beschaffen, erwiesen sich, vor allem durch die Multivalenz verschiedener Begriffe, als qualitativ minderwertig. Sämtliche zugeordneten, Begriffe werden in einer graph-basierten Datenbank persistiert, welche gerade auf Lese- bzw. Traversierungs-Operationen optimiert ist. Ein optisch ansprechendes Graphical User Interface (GUI) präsentiert die gewonnen Informationen, auf verständliche Art und Weise. Durch den Einsatz ausschliesslich plattformunabhängiger Technologien, ist das System auf allen - von Java 6 unterstützten Betriebssystemen - einsatzfähig, sogar in heterogenen Konfigurationen.

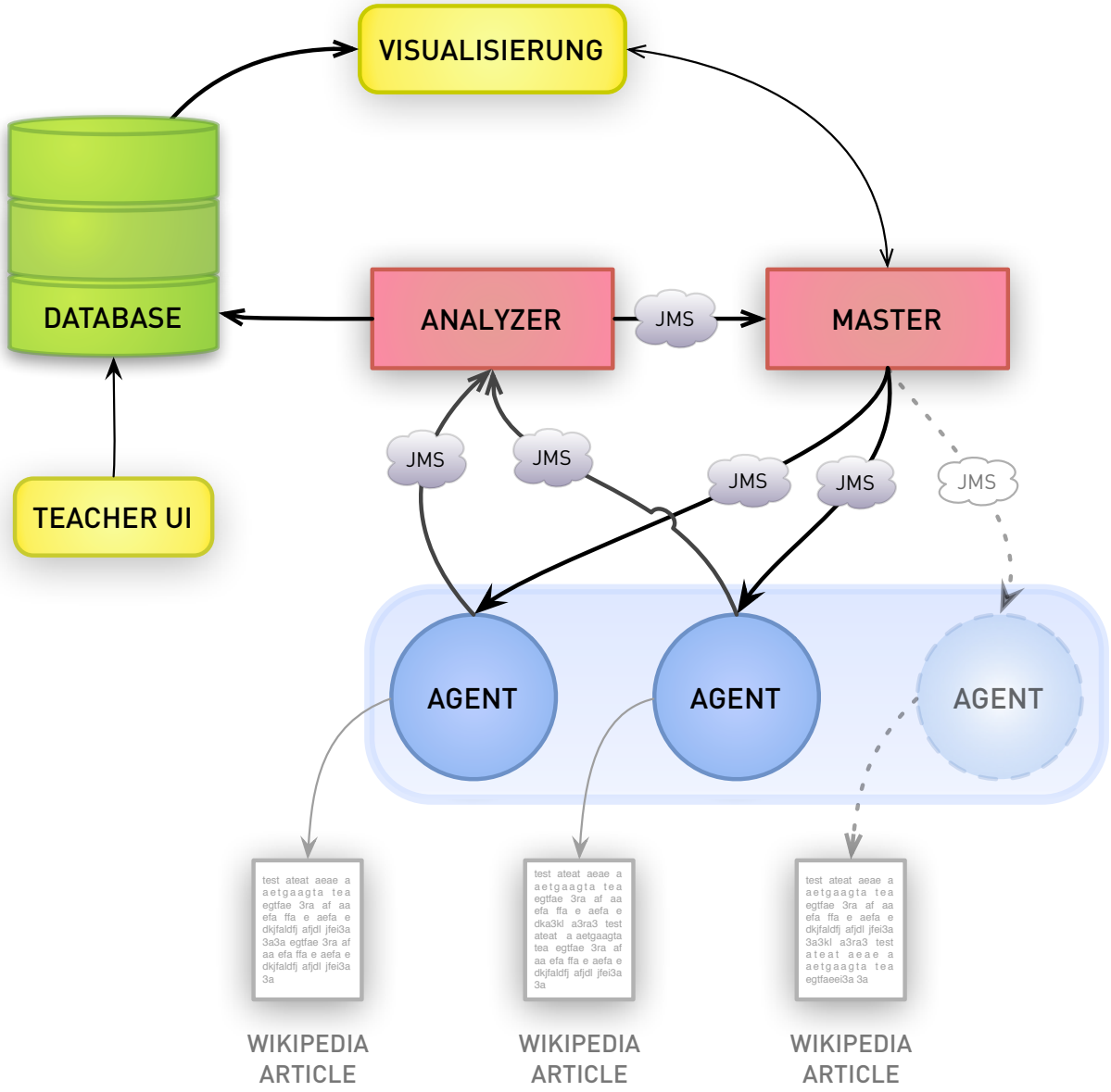


Abbildung 3.2: Übersicht zur konzipierten Lösung

Teil II

Technischer Bericht

Kapitel 4

Einführung

4.1 Problemstellung, Vision

> "The Semantic Web is [...] an extension of the current [Web], in which information is given well-defined meaning, better enabling computers and people to work in cooperation." - Edd Dumbill, 2001, `http://www.xml.com/pub/a/2000/11/01/semanticweb/`

Gemäss aktuellen Schätzungen umfasst das World Wide Web im Mai 2009 mindestens 28.41 Milliarden indexierte Webseiten.[1] Auf jeder dieser Seiten, ist wiederum eine Vielfalt von Wissen verschiedener Qualität publiziert. Mehr als 1.5 Milliarden Menschen[2] haben Zugang zum World Wide Web und sorgen dafür, dass ständig neues Wissen hinzugefügt wird. Die Enzyklopädie Wikipedia zählt mittlerweile mehr als 2.8 Millionen englische[3] und beinahe 900'000 deutsche Artikel,[4] zu ganz unterschiedlichen Themen. Die Auswertung dieses Wissens wird jedoch stets durch den Menschen selbst gemacht, da die publizierten Informationen für den Computer nicht differenzierbar sind. Man stelle sich vor, welche Möglichkeiten sich ergeben, wenn der Computer beginnt, einfache Zusammenhänge selbst zu interpretieren.

Als Beispiel soll in Erfahrung gebracht werden, welche Filme in der Schweiz gedreht wurden. Mit dieser Aufgabenstellung können konventionelle Suchmaschinen, wie Google oder Yahoo, nur etwas anfangen, wenn sich in ihrem indexierten Content genau eine Seite befindet, welche die Antwort auf diese Frage liefert. Würde eine Suchmaschine jedoch verstehen, dass es sich beim Term "Schweiz" um ein Land handelt und der Term "Film" durch einen "Drehort" mit dem Typ "Land", verbunden ist, könnte die Suchanfrage völlig neu gestellt werden. So aufgeschlüsselt, versteht der Computer selbst, dass es sich dabei eigentlich um eine boolesche Operation auf alle Filmtitel und alle Drehorte handelt.

Dieses einfache Beispiel zeigt die grosse Vision des Projektes auf: Das im Internet vorhandene Wissen soweit zu gliedern, dass es maschinen-verstehbar wird. Dies ist, selbstredend, für eine einsemestrige Bachelorarbeit, ein unmögliches Unterfangen. Die Arbeit umfasst deshalb die Erstellung eines ersten Prototypen zur Wissensbeschaffung, zur Datenhaltung und zu deren Visualisierung.

[1] `http://www.worldwidewebsize.com`, Stand: 15.05.2009
[2] `http://www.itu.int/ITU-D/ict/statistics/at_glance/KeyTelecom99.html`, Stand: 2008
[3] `http://en.wikipedia.org/wiki/Wikipedia:Size_comparisons`, Stand: 09.06.2009
[4] `http://de.wikipedia.org/wiki/Wikipedia:Über_Wikipedia`, Stand: 09.06.2009

4.2 Ziele und Unterziele

In den folgenden Abschnitten werden die Ziele der Bachelorarbeit erläutert.

4.2.1 Prototyp: Wissensextraktion

Es soll ein Prototyp zur Wissensextraktion aus einer im Internet vorhandenen Quelle entwickelt werden. Dabei soll aufgrund vorgegebener Typen und Beispielwerten, weitere unbekannte Begriffe einem Objekttyp zugeordnet werden können. Die Skalierbarkeit der Informationsbeschaffung soll ein wichtiger Bestandteil sein. Daher empfiehlt sich der Einsatz eines verteilen Systems.

- Evaluation bestehender Lösungen oder Neu-Entwicklung eines Html-Crawlers.
- Evaluation verschiedener Architekturen von verteilten Systemen.
- Entwicklung einer verteilten Lösung zur Informationsbeschaffung / Informationsextraktion.
- Evaluation verschiedener Clustering-Algorithmen zur Typenzuordnung.
- Implementation eines Clustering-Algorithmus / Clustering-Systems.

4.2.2 Prototyp: Datenhaltung

Es soll ein Prototyp zur Datenhaltung der extrahierten Daten entwickelt werden. Angesichts der anzunehmend riesigen Datenmenge, soll eine möglichst skalierbare Technologie zur Datenspeicherung verwendet werden. Dabei kann gerade die Modellierbarkeit des Problems als gewichteter Graph eine wichtige Rolle spielen.

- Evaluation graph-basierter Datenbanken.
- Implementation der Datenhaltung der Knowledge-Objects.
- Entwicklung eines einfachen Tools, um gespeicherte Daten zu korrigieren.

4.2.3 Prototyp: Visualisierung

Es soll ein Prototyp zur Visualisierung der gespeicherten Daten entwickelt werden. Als Java2D-Framework empfiehlt sich Processing 1.0.[5]

- Entwurf einer Architektur für die Darstellung von Processing-Objekten.
- Entwurf eines Graphical User Interfaces.
- Entwicklung der Visualisierungsplattform.

[5] `http://processing.org/`

4.3 Rahmenbedingungen, Umfeld, Definitionen, Abgrenzungen

Zu Beginn des Projekts wurden umfassende Recherche-Arbeiten durchgeführt, um herauszufinden, ob bereits ähnliche Arbeiten bestehen. Informationen und Projekte zu "Semantischen Netzen" sind im Internet reichlich vorhanden. Ein sehr spannendes Projekt, das es in diesem Zusammenhang zu erwähnen gilt, ist Eyeplorer.[6] Dieses Projekt wird von einer, im Jahr 2008 gegründeten, Firma in Deutschland entwickelt und hat sich darauf spezialisiert, Zusammenhänge zwischen verschiedenen Begriffen und deren Verwandtschaft aufzuzeigen. Mit diesem interessanten Ansatz, hat die Firma den Innovationspreis-IT 2009 im Rahmen der CeBIT[7] in Hannover in zwei Kategorien gewonnen.[8] Eine Ähnlichkeit zu unserer Arbeit zeigt sich darin, dass Eyeplorer ebenfalls auf Daten aus der freien Enzyklopädie Wikipedia zugreift und diese auswertet. Unser Ansatz grenzt sich jedoch von Eyeplorer ab, indem wir nicht in erster Linie Zusammenhänge zwischen Begriffen darstellen wollen, sondern Wert darauf legen, einen Begriff einzuordnen und feststellen zu können, um was es sich handelt. Diese Funktionalität fehlt Eyeplorer gänzlich.

Da der Arbeit seitens des Betreuers grösstmögliche Freiheiten gewährt wurden, mussten zu Beginn die Rahmenbedingungen klar abgesteckt werden. So wurde beschlossen, dass man sich auf Wikipedia als Quelle von Informationen beschränkt und das System vorerst nicht über das Internet öffentlich zugänglich macht.

4.4 Vorgehen, Aufbau der Arbeit

In einem einwöchigen Thema-Workshop zu Beginn des Semesters, haben sich die Projektteilnehmer auf eine Einschränkung des Themas "Kontextanalyse" geeinigt. Obwohl sich die Arbeit relativ einfach auf die drei Themenschwerpunkte Knowledge-Extraction, Datastorage und Visualisierung aufteilen lässt, wurde entschieden, keine definitive Arbeitsteilung einzuführen, sondern nur einen Verantwortlichen pro Themengebiet zu bestimmen. Aufgrund den, in der Studienarbeit 'Ziirp' erarbeiteten Erkenntnisse, soll das Augenmerk auf eine skalierbare Architektur gelegt werden.

Auf den Softwareentwicklungsprozess bezogen, soll analog zur Semesterabeit, auf ein agiles Modell gesetzt werden. Da die Applikation für die HSR entwickelt wird und der Betreuer dem Entwicklungsteam bewusst viel Selbständigkeit zugesteht, wird auf die Simulation eines virtuellen Kunden verzichtet. Das Projekt hat also die spezielle Eigenschaft, dass es durch die Kreativität der Entwickler getrieben werden soll. Den Projektfortschritt sollen regelmässige Teamsitzungen und Code-Reviews sicherstellen. Dazu kommen in grösseren Abschnitten Controlling-Sitzungen, an denen auch der Projektbetreuer teilnimmt. Neben Code-Reviews, werden auch Unit-Tests und verschiedene Metriken als Qualitätskontrolle eingesetzt.

Zunächst gilt es, innerhalb von 4 Wochen, einen ersten, funktionstüchtigen Prototypen zu entwickeln und sich so möglichst schnell ins Thema einzuarbeiten. Gerade weil das Themengebiet für die Projektteilnehmer Neuland darstellt, kommt der Arbeit auch einen experimentellen Charakter zu: Ziele können zwar gesteckt werden, der Arbeitsfortschritt ist jedoch im Voraus schwer abschätzbar.

Nachdem der Beweis der Machbarkeit erbracht worden ist, geht es darum, den ersten Prototypen neu aufzugleisen. Gleichzeitig soll mit dem Entwurf der Datenhaltungs-Architektur begonnen werden.

[6] `http://www.eyeplorer.com/`
[7] `http://www.cebit.de/`
[8] `http://www.vionto.com/news_inno.html`

Hier geht es vor allem auch darum, Alternativen zum klassischen, relationalen Datenmodell zu entdecken. Auch kommt der Anforderung nach einer potentiell sehr grossen Anzahl einzelner Datensätze, eine grosse Bedeutung zu. Im zweiten Prototypen soll demnach die wichtigste Datenhaltungsfunktionalität bereits integriert sein.

Schliesslich gilt der Fokus der letzten Iteration, der Visualisierung der gespeicherten Datensätze. Falls genügend Zeit vorhanden ist, können hier idealerweise Paper Prototyping und User Akzeptanz Tests durchgeführt werden.

Während des ganzen Projektzeitraums soll ausserdem der Typenkatalog ständig gepflegt und erweitert werden. Dies ermöglicht gegen Ende einen Testlauf über ein grösseres Sample der in Wikipedia archivierten, deutschsprachigen Artikel.

In den letzten 2 Wochen des Projektes sollen schliesslich die erzielten Resultate aus- und bewertet werden.

Kapitel 5

Wissensgebiete

5.1 Text Mining

5.1.1 Definition

> Text repräsentiert Wissen. Im Unterschied zu den strukturierten Daten in einer Datenbank, stellen Texte unstrukturierte Daten dar. Mit Hilfe von Text Mining Werkzeugen, können aus digital vorliegenden Texten neue und relevante sachliche und inhaltliche Zusammenhänge extrahiert werden. Text Mining basiert auf statistischen und musterbasierten Verfahren und erlaubt innovative Anwendungen im Wissensmanagement. [1]

5.1.2 Linguistische Analyse von Texten

Die Bedeutung aus natürlichsprachlichen Texten zu erschliessen, ist eine der grössten Herausforderungen des Text Mining. Wörter und Phrasen können einerseits mehrdeutig sein und andererseits, kann man dieselbe Bedeutung durch verschiedene Wörter und Phrasen ausdrücken. Beide Aspekte erschweren die Aufgabe massiv. Text Mining setzt daher die linguistische Erschliessung des Textes voraus.

5.1.3 Verfahren

Die Text Mining Verfahren haben folgende Ziele:

- In Texten implizit vorhandene Informationen sollen explizit gemacht werden.
- Zusammengehörige Informationen aus verschiedenen Texten sichtbar zu machen.

Um dies zu erreichen, werden Methoden der explorativen Datenanalyse und des logischen Schliessens verwendet. Das maschinelle Lernen, sowohl überwacht als auch unüberwacht, spielt bei der Entwicklung solcher Verfahren eine grosse Rolle.

In unserem Projekt verwenden wir ein Verfahren, welches einen Text in einen hochdimensionalen Vektorraum von Termen und deren Häufigkeit überführt. Die Auswertung von Beziehungen zwischen Dokumenten durch diesen Vektor ermöglicht es, Dokumente zu ermitteln, die sich auf denselben Sachverhalt beziehen, obwohl ihr Wortlaut verschieden ist. Die Auswertung der Beziehungen zwischen Termen in dieser Matrix ermöglicht es, assoziative Beziehungen zwischen den Termen herzustellen, die oftmals semantischen Beziehungen entsprechen und in einer Ontologie repräsentiert werden können.

5.2 Clustering

Dieser Abschnitt gibt einen groben Einblick ins Thema Clustering. Dabei wird das Clustering-Verfahren k-means genauer beschrieben, da in dieser Arbeit Teile dieses Verfahrens verwendet werden.

5.2.1 Prinzip

Unter Clustering wird das Zuordnen von einzelnen Objekten zu Clustern (Objektmengen) verstanden. Dabei werden Objekte gleicher Art, zu einem Cluster zusammengefasst. Die entsprechenden Analyseverfahren, lassen sich zur Klassifizierung und / oder zur Erkennung von Mustern in verschiedensten Bereichen, wie der Bildverarbeitung, der Informationsbeschaffung, bei Kunden-Segmentierungsverfahren, oder wie in dieser Arbeit für das Text Mining einsetzen.

5.2.2 Distanz

Für die Bestimmung der Ähnlichkeit der einzelnen Objekte, wird eine Abstandsfunktion verwendet. Die Objekte werden durch einen n-dimensionalen Vektor beschrieben. Dabei bildet das Skalarprodukt, auch inneres Produkt oder Punktprodukt genannt, die Distanz zwischen zwei Vektoren. Multidimensionale Vektoren sind schwer visualisierbar, aber anhand eines zwei-dimensionalen Vektors, kann die Distanz einfach illustriert werden.

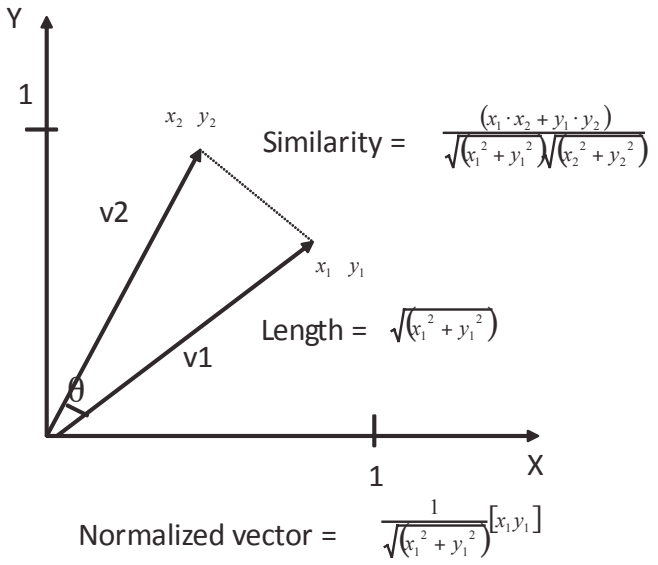


Abbildung 5.1: Berechnung des Distanz-Masses
Quelle: "Collective intelligence in Action"[3], S. 31

5.2.3 Clusteringverfahren

Bei Clusteringverfahren wird zwischen hierarchischen und unterteilenden Verfahren unterschieden. Wobei bei den erstgenannten, noch zwischen agglomerierenden (bottom-up) oder unterteilenden (top-down) Algorithmen differenziert wird.

Weiter unterscheidet man zwischen "harten" und "weichen" Clusteringalgorithmen. Harte Methoden (z. B. k-means), ordnen jeden Datenpunkt genau einem Cluster zu, wobei weiche Methoden, jedem Datenpunkt für jeden Cluster eine Wahrscheinlichkeit zugeordnet wird, mit der dieser Datenpunkt in diesem Cluster liegt. Weiche Methoden sind insbesondere dann nützlich, wenn die Datenpunkte relativ homogen im Raum verteilt sind und die Cluster nur als Regionen mit erhöhter Datenpunktdichte in Erscheinung treten, d.h. wenn es z. B. fliessende Übergänge zwischen den Clustern oder Hintergrundrauschen gibt. Harte Methoden sind in diesem Fall unbrauchbar.

5.2.4 K-means

K-means ist ein klassisches Clusteringverfahren. Es ordnet jeden Punkt demjenigen Cluster zu, dessen Mittelpunkt am wenigsten weit entfernt ist. Dabei wird anhand dieser Zuteilung, der Clustermittelpunkt für jeden Durchlauf neu berechnet. Das bedeutet, dass die Clustermittelpunkte und somit die Anzahl Cluster als Ausgangswert vorgegeben werden muss.

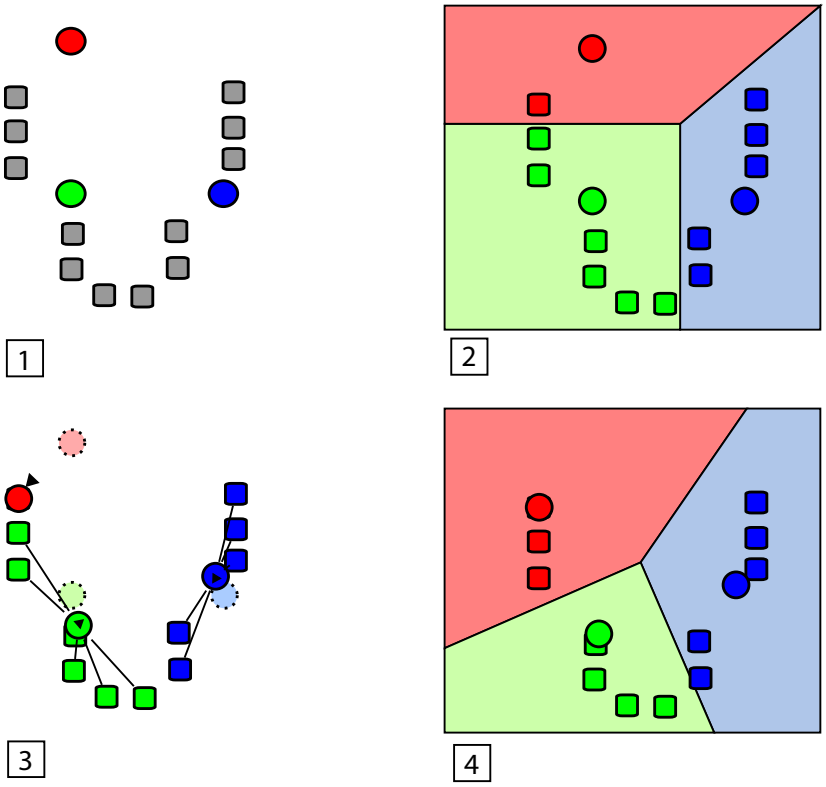


Abbildung 5.2: Clustering-Ablauf
Quelle: `http://de.wikipedia.org/wiki/k-means`

1. Bestimmung von k Clusterzentren.
2. Jedes Objekt wird demjenigen Cluster zugeordnet, dessen Zentrum ihm am nächsten liegt. (Distanz zum Clustermittelpunkt)
3. Für jeden Cluster wird das Zentrum neu berechnet. Basierend auf den neu berechneten Zentren, werden die Objekte wieder analog zu Schritt 2 auf die Cluster verteilt.
4. Wurde bei der Neuverteilung mindestens ein Objekt einem anderen Cluster zugeordnet, wiederhole den Algorithmus beginnend mit Schritt 3.

Da die Clusterzentren zufällig gewählt werden, liefert nicht jeder Durchlauf dasselbe Ergebnis. Zudem kann es vorkommen, dass einzelne Objekte endlos zwischen den Clustern hin und her springen, was eine Beschränkung der Anzahl Iterationen erfordert.

Vorteile des k-means Algorithmus:

- einfache Implementierung
- schnelle Laufzeit, lineare Komplexität $O(k*n*i)$, $k : Clusterzentren; n : Objekte; i : Iterationen.$

Probleme des k-means Algorithmus:

- zufällige Initialisierung der Clusterzentren.
- vorgängige Auswahl der Anzahl der Clusterzentren.
- Beschränkung der Clustergestalt auf elliptische Formen ähnlicher Grösse.
- mangelhafte Robustheit gegen verrauschte Daten und Ausreisser.

5.2.5 Ward Clustering

Ward Clustering ist ein bottom-up hierarchisches Clusteringverfahren. Dabei repräsentiert jeder Punkt (Objekt) einen Cluster. Bei jedem Durchlauf werden die beiden ähnlichsten Cluster zu einem grösseren Cluster zusammengelegt. Das Clustering-Ergebnis entspricht einem Binärbaum. Im Gegensatz zu k-means, muss so nicht vorgängig angegeben werden, wie viele Clusterzentren erwartet werden. Dafür entscheidet die Betrachtungshöhe des Binärbaums darüber, wie viele Clusters existieren. Am oberen Ende des Baumes gibt es noch ein Cluster, der alle Objekte enthält. Am unteren Ende entspricht jedes Element genau einem Cluster. Die Schwierigkeit ist es, die gewünschte Betrachtungshöhe zu finden. Diese liegt irgendwo dazwischen.

Im Gegensatz zu k-means, sind mit Ward auch komplexere Formen, als reine Ellipsen, erkennbar. Dies, weil die Cluster Zugehörigkeit nicht anhand des Clusterzentrums (Radius zum Mittelpunkt) sondern anhand der Distanz zweier Objektmengen (Clustern) ermittelt wird. Dies hat zur Folge, dass ein Ward Clustering rechenintensiver ist. Die Komplexität beträgt $O(n^2)$.

5.2.6 Self-organised clustering in spiking networks (Ausblick)

Ein mögliches Clusteringverfahren, das ohne Kenntnis der Anzahl Cluster und deren Form auskommt, wird in einer Publikation [2] von T. Ott, M. Christen und R. Stoop beschrieben.

5.3 Message Service

In diesem Projekt wird JMS (Java Message Service) als Messaging Service eingesetzt. Nachfolgend werden die beiden grundsätzlichen Ansätze zur Nachrichenübermittlung kurz erklärt.

5.3.1 Publish / Subscribe Prinzip (Topic)

Ein oder mehrere Publisher (Sender) senden Nachrichten an mehrere Subscriber (Empfänger). Dabei erhalten alle Subscriber die identische Nachricht. Die Nachrichten werden (in dieser Umsetzung / Implementation) nicht gespeichert. Meldet sich ein Subscriber nach dem Senden der Nachricht an, so erhält er diese nicht. Bei JMS werden Nachrichten mit einem bestimmten Topic, an den JMS Server gesendet. Clients melden sich für dieses Topic an und beziehen die Nachrichten dazu, über den Server.

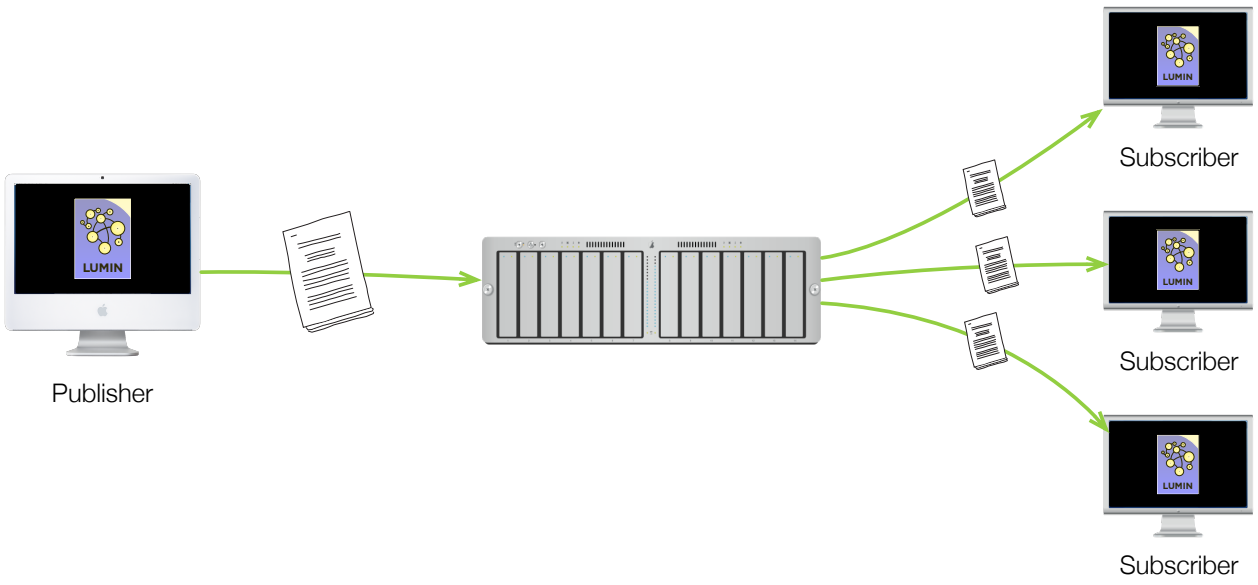


Abbildung 5.3: Publish / Subscribe Messaging

5.3.2 Nachrichtenwarteschlangen Prinzip (Queue)

Queues (Warteschlangen) sind Nachrichtenkanäle, die normalerweise zwischen genau einem Sender und einem Empfänger aufgebaut werden (Punkt-zu-Punkt-Verbindung). Wenn ein Empfänger eine Nachricht erhält, wird diese aus dem Nachrichtenkanal entfernt. Für das Projekt wird die JMS Queue als "Point to Multipoint"- und "Multipoint to Point"-Kommunikation verwendet. Das bedeutet, dass ein Sender an mehrere Empfänger oder im anderen Fall mehrere Sender, an ein Empfänger senden. Im Unterschied zum Publish/Subscribe Prinzip, werden hier jedoch die Nachrichten exakt einmal (exactly-once) von einem Empfänger verarbeitet. Dabei wird sichergestellt, dass die Nachricht auch einen Empfänger erreicht.

Generell kann in verteilten Systemen keinerlei Garantie für eine fehlerlose Kommunikation gegeben werden. Denn falls beispielsweise ein Netzwerk-Knoten dauerhaft ausfällt, ist in jedem Fall auch keine einzige Ausführung möglich.

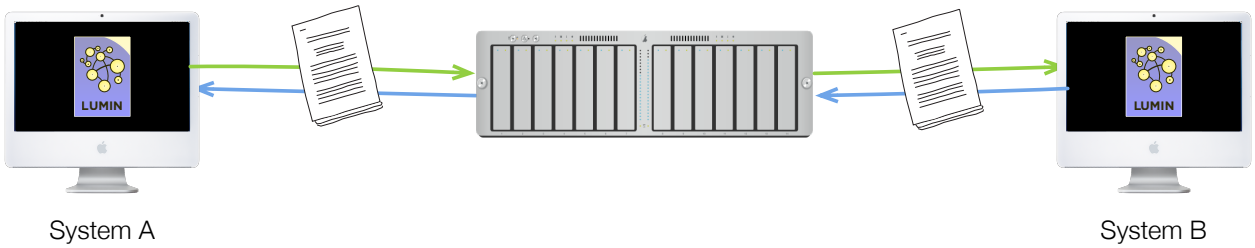


Abbildung 5.4: Point to Point Messaging

Kapitel 6

Evaluation

6.1 JMS Provider

Um die Analyse-Aufgaben auf einzelne Teilsysteme verteilen zu können, wird JMS für die Kommunikation eingesetzt. Dabei kommunizieren alle Systeme über einen zentralen Dienst, den Message Broker. Dieser ist für die Verteilung der Messages zuständig.

6.1.1 Open Message Queue

Es gibt eine ganze Menge von Message Broker Software; Freie, wie auch kostenpflichtige Produkte. Da die API (für die Ansteuerung von Message Oriented Middleware) genormt ist, kann grundsätzlich jeder beliebige JMS Provider verwendet werden. Daher wurde für die Evaluation des JMS Providers nicht besonders viel Zeit aufgewendet.

Entscheid:

Man hat sich für Open Message Queue[1] in der Version 4.3 entschieden.
Open Message Queue ist die Open Source Version von 'Sun's Java System Message Queue'. Ausschlaggebend war, dass Open Message Queue kostenlos ist und alle benötigten Anforderungen zur Verfügung stellt. Zudem könnte es in einem Cluster betrieben werden. Dies wird in dieser Arbeit nicht verwendet, wäre aber zu einem späteren Zeitpunkt denkbar, um den 'single point of failure', den Message Broker, zu eliminieren.
Bei Open Message Queues, können mehrere Clients lesend und schreibend auf eine Queue zugreifen. Die meisten kostenlosen Konkurrenz-Produkte, erlauben jedoch jeweils nur einen Lese- und Schreibprozess. Da in diesem Fall mehrere Clients eine Queue auslesen aber immer nur einer den entsprechenden Auftrag bearbeiten soll, wird diese Funktionalität zwingend benötigt.
Der Entscheid für Open Message Queue erwies sich im Laufe des Projekts als richtig. Es gab damit keinerlei Probleme. Das Produkt lief immer stabil und performant.

[1] `https://mq.dev.java.net/`

6.2 Datenbank-Technologien

6.2.1 Problemstellung

Für die Speicherung der extrahierten Daten, werden aufgrund des grossen Datenumfangs, voraussichtlich massive Datensatzmengen anfallen. Da zu einem Eintrag jedoch verhältnismässig wenig Informationen gespeichert werden müssen, ist es die hohe Anzahl der Einträge, welche das voraussehbare Problem charakterisiert.

6.2.2 Lösungsansätze

Eine Möglichkeit wäre es, eine relationale Datenbank, wie beispielsweise PostgreSQL[2] oder MySQL[3], einzusetzen. Mittels eines Objekt-Relationalen Mappings, könnte man die Java Objekte unkompliziert im relationalen Schema abspeichern. Besonderer Beliebtheit, erfreuen sich die beiden Persistenz-Frameworks Hibernate[4] und JDO[5] (Java Data Objects). Der Einsatz dieser Technologien bietet viele Vorteile, besonders weil diese Frameworks bereits sehr ausgereift und praxiserprobt sind. Andererseits denkt man bei der Problemstellung nicht unbedingt an eine relationale Datenspeicherung. Viel eher würde man die Problemstellung als Graph, als verkettete Menge von Knoten, abbilden. Eine solche Repräsentation wäre gerade für die Abbildung von Listen, Hierarchien und Bäumen von Vorteil. So stiess man auf die graph-basierte Datenbank Neo4j[6] (siehe Kapitel 7.6.2). Ausserdem ist es durchaus möglich, dass sich die Art der angesammelten Daten mit der Zeit ändert. Daher wäre auch ein anpassbares Datenbank-Schema von Vorteil.

[2] http://www.postgresql.org/
[3] http://www.mysql.com/
[4] https://www.hibernate.org/
[5] http://java.sun.com/jdo/
[6] http://neo4j.org/

Kapitel 7

Umsetzungskonzepte

7.1 Text Mining

Dieser Abschnitt beschreibt, wie der vom Parser extrahierte Text gefiltert, analysiert und zu einem Vektor aufbereitet wird. Dieser Vektor ist nötig, damit der Text durch das Clusteringverfahren gruppiert werden kann. Dabei wurde stark nach dem, im Buch "Collective Intelligence in Action"[3] beschriebenen Verfahren vorgegangen. Für die Analyse des Textes, wird die Open-Source-Java-Bibliothek Lucene[1] verwendet.

7.1.1 Filter

Der vom Parser zurückgelieferte Text, wird der Reihe nach durch folgende Filter geschleust.[2]

StandardTokenizer

- Splittet den Text in einzelne Wörter auf. Dabei werden Satzzeichen von den Wörtern am Ende eines Satzes entfernt.
- Splittet Wörter auch zwischen Bindestrichen.
- Erkennt Mail- und Internetadressen als ganzes und speichert diese als Token.

LowerCaseFilter

- Normalisiert die Token zu Kleinbuchstaben

StopFilter

- Eliminiert Stoppwörter anhand einer Stoppwörter-Liste.

LengthFilter

- Eliminiert Wörter, die nicht einer Mindest- oder Maximallänge entsprechen.

[1] `http://lucene.apache.org/java/docs/`
[2] `http://wiki.apache.org/lucene-java`

PorterStemFilter

- Transformiert Token mit dem Porter Stemmer Algorithmus in deren Wortstamm. Die Wörter dürfen dafür keine Grossbuchstaben aufweisen.

Da die deutsche Version von Wikipedia als Datenquelle verwendet wird, definiert eine deutsche Stoppwörter-Liste alle Wörter, die nicht zur Analyse beigezogen werden sollen. Meist versucht man damit sämtliche Präpositionen, und Adverbien herauszufiltern, da diese schlichtweg überall vorkommen können, ohne eine relevante Aussage zu treffen.
Da hier jedoch kein Standardtext analysiert wird, sondern der Text schon vorgefiltert ist und nur aus Nomen besteht, wird als Stoppliste eine, auf die Datenquelle zugeschnittene, Liste verwendet. In diesem Fall enthält diese, Wikipedia-spezifische Wörter, die nicht schon vom Html-Parser gefiltert werden können.
Weiter werden alle Wörter verworfen, die weniger als 3 und mehr als 30 Buchstaben aufweisen. Diese sind für die spätere Gruppierung irrelevant. Zuletzt werden die Wörter mit dem Porter Stemmer Algorithmus (für die deutsche Sprache) in ihre Stammform zerlegt. Dazu wird eine spezielle Implementation für die deutsche Sprache verwendet.

> Der Porter Stemmer Algorithmus ist ein verbreiteter Algorithmus, der Computerlinguistik zum automatischen Zurückführen von Wörtern auf ihren Wortstamm (Stemming) verwendet. Der Algorithmus basiert auf einer Menge von Verkürzungsregeln, die so lange auf ein zu stemmendes Wort angewandt werden, bis dieses eine Mindestanzahl von Silben aufweist. Der ursprünglich für Wörter der englischen Sprache entwickelte Algorithmus, kann relativ leicht für andere Sprachen portiert werden. - `http://de.wikipedia.org/wiki/Porter-Stemmer-Algorithmus`

7.1.2 Vektor-Generierung

Aus den gefilterten Tokens (Wörter) wird nun ein Vektor erzeugt. Dafür wird jedes, nach der Filterung, noch verbleibende Wort, als Attribut verwendet, und ihm die Auftrittshäufigkeit des Wortes hinzugefügt.

Vereinfacht gesehen wird so ein Text als mehrdimensionaler Vektor abgebildet. Das Attribut definiert die Achse (Dimension) und die Auftrittshäufigkeit, die Distanz.

Für die Kategorisierung müssen die Daten noch normalisiert werden. Dadurch wird eine Wertung (Magnitude) zwischen 0 und 1 für jedes Attribut, anhand dessen Auftrittshäufigkeit berechnet. Korrekterweise setzt sich die Wertung aus der normalisierten Auftrittswahrscheinlichkeit eines Wortes zusammen; Denn ein Wort, mit kleiner Auftrittswahrscheinlichkeit, sollte stärker gewichtet werden, als ein Wort, das häufig auftritt.

Diese Auftrittswahrscheinlichkeit wird in der Textanalyse als IDF (Inverse Document Frequency) bezeichnet. Diese wird wie folgt berechnet[3]:

$$idf(t_i) = log\frac{N}{n_i}$$

$$N : Anzahl\ Dokumente$$
$$n_i : Dokumente,\ die\ t_i\ enthalten$$
$$t : zu\ analysierendes\ Wort$$

[3]Understanding IDF: `http://www.soi.city.ac.uk/~ser/idfpapers/Robertson_idf_JDoc.pdf`

In diesem Projekt entspricht ein Termvektor eines Wikipedia-Artikels einem Dokument. Die Berechnung der IDF für jedes Wort, wurde konstant mit dem Wert 1 implementiert. Es wurden zwei Varianten für deren Berechnung geprüft.

1. Berechnung der Auftrittswahrscheinlichkeiten anhand der analysierten Daten. Dabei würden alle verarbeiteten Wörter analysiert und für jedes einzelne, die Auftrittswahrscheinlichkeit berechnet. Das Problem dabei ist, dass dieser Wert erst nach einer grossen Menge von Analysen aussagekräftig wird und sich mit jeder weiteren Analyse verändert. Dadurch müssten alle, bereits analysierten Daten laufend neu analysiert werden.

2. Abfragen von bestehenden Quellen, die Auftrittswahrscheinlichkeiten zu Wörtern liefern, wie z.B: WordNet[4]. Diese Idee wurde jedoch verworfen, da diese Quellen nicht offline verfügbar sind und man sich so von einem weiteren Dienst abhängig machen würde. Zudem sind die Daten unvollständig und daher nicht für alle Wörter Auftrittswahrscheinlichkeiten verfügbar. Weiter wären n Abfragen bei n Wörtern nötig. Was bei einer durchschnittlichen Wörter Anzahl von ca. 1000 pro Begriff, die Analyse extrem verlangsamen würde.

Analysen der Gruppierungs-Resultate haben gezeigt, dass die Wertigkeiten ohne individuelle Auftrittswahrscheinlichkeiten trotzdem zu sehr guten Resultaten führen. Denn für die Gruppierung ist hauptsächlich die Wertigkeit relevant. Die Höhe des Wertes ist nur sekundär.

7.2 Clustering

Für das Clusteringverfahren wurde der k-means Algorithmus als Grundlage gewählt. Dabei wurden die klassischen k-means Problematiken eliminiert und den Bedürfnissen dieses Projekts angepasst. Entstanden ist ein Verfahren, welches auf die vorhandenen Bedürfnisse zugeschnitten ist.

7.2.1 Prototypen-Clustering

Die Clusterzentren werden nicht, wie bei k-means üblich, zufällig gewählt. Bei dieser Variante lassen sich sogenannte Prototypen übergeben, welche die Cluster definieren. Dabei können beliebig viele Prototypen für denselben Cluster existieren. Das Clusterzentrum wird jeweils aus diesen Prototypen gebildet. Für jeden Cluster muss mindestens ein Prototyp angegeben werden. Dadurch wird die k-means-Problematik der zufälligen Clusterzentren Wahl eliminiert. Zudem kann der Cluster und die enthaltenen Elemente, nun anhand des Prototypen identifiziert werden.
Da die Klassifizierung auf den vorgegebenen Prototypen basiert, ist es wichtig, die Wahl dieser Prototypen sehr ausgewogen zu gestalten. Sollen zum Beispiel alle Länder klassifiziert werden,muss ein breiter Mix aus Ländern aller Kontinente, so wie Kleinstaaten, Inselstaaten, usw. als Prototypen verwenden werden. Damit wird definiert, dass alle Länder zu dieser Gruppierung gehören sollen und nicht nur eine Subsektion (z.B. europäische Länder), erfasst werden soll.

Verworfene Alternative

Eine Idee war es, die zufällige Clusterzentrum-Wahl beizubehalten und die zu analysierenden Objekte mit klassifizierten Prototypen zu versehen. Dabei werden die Objekte und Prototypen nach k-means

[4] `http://wordnet.princeton.edu/`

geclustert und die Anzahl Cluster nach der, mehrheitlich optimalen, Formel $k = (n/2)^{1/2}$ berechnet. Danach wird jeder Cluster analysiert und die grösste Anzahl Prototypen desselben Typs, bestimmt dann den Typ des Clusters.

Beispiel: Ein Cluster mit folgenden, zugeordneten Prototypen:
[“Auto”,”Fortbewegungsmittel”] [“Fahrrad”,”Fortbewegungsmittel”] [“Mountainbike”,”Sportart”] [“Strassenbahn”,““] [“Flugzeug”,““] [“Motorrad”,””]

Der Prototyp “Fortbewegungsmittel” überwiegt. Daher würden alle Elemente dieses Clusters als Fortbewegungsmittel klassifiziert. Diese Variante wurde verworfen, da die erzielte Erkennungsrate um ca. 10-15% schlechter war, als mit der im System implementierten. Zudem kann es vorkommen, dass ein Prototypen von anderen Typen überschrieben wird. Da “Mountainbike” im obigen Beispiel der einzige Prototyp für “Sportart” ist, wird er durch “Fortbewegungsmittel” überschrieben.

7.2.2 Clustering Iteration

Bei k-means werden die Objekte solange zugeordnet, bis sich daran nichts mehr ändert (siehe Kapitel 5.2). Während der Analyse der Clustering Resultate hat sich gezeigt, dass sich die besten Resultate genau nach nur einer Clustering-Iteration ergeben. Beim zweiten Durchlauf werden die Resultate schlechter und anschliessend wieder etwas besser, erreichen jedoch nie mehr bessere Werte, als beim ersten Durchlauf. Dies gilt nicht für k-means allgemein, sondern nur für die hier gemachten Anpassungen. Auf Grund dieser Erkenntnisse wird über alle Objekte einmal itereriert und dem entsprechenden nächsten Zentrum zugewiesen. Eine Neuberechnung der Clusterzentren, die für weitere Durchlaufe nötig wären, erübrigt sich somit.

7.2.3 Grenzwerte

Minimale Ähnlichkeit

Beim Clustering wird im Allgemeinen jedes Objekt genau einem Cluster zugeteilt. Das bedeutet, wenn nun für ein Objekt noch kein Prototyp und somit auch kein Cluster besteht, wird es dem ähnlichsten Cluster zugeordnet. Um dies zu verhindern, wurde ein “minSimilarity”-Wert eingeführt. Objekte, die zu weit von einem Cluster-Zentrum entfernt sind, werden als „unknown“ markiert. Damit werden fehleranfällige Zuordnungen zur manuellen Überprüfung schnell erkannt.

Mindestabstand

Für gewisse Objekte ist es schwierig, eine eindeutige Zuteilung zu finden. Beispiel: Ist ein Tanzschuh nun ein Sportgerät oder ein Kleidungsstück? Der Clustering-Algorithmus weist das Objekt dem Cluster mit der grössten Ähnlichkeit zu. Dies ist jedoch nicht immer wünschenswert. Daher wurde ein “minDistance” Wert eingeführt. Dieser regelt wie gross die Distanz zwischen dem ähnlichsten und den zweit ähnlichsten Objekt sein muss. Ist die Distanz zu klein, wird das Objekt als „unknown“ markiert. So können kritische Zuteilungen identifiziert und später manuell überprüft werden.

7.2.4 Abgrenzung

So wie bei k-means, ist es auch mit dieser Implementation "nur" möglich, elliptisch angeordnete Punkte zu einem Cluster zusammenzufügen.

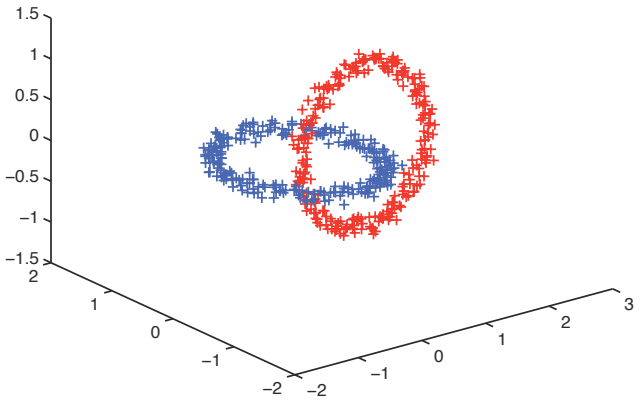


Abbildung 7.1: HLC-Clustering
Quelle: Theorie und Simulation Neuronaler Netze R. Stoop, ETHZ SS 2009

Abbildung 7.1 zeigt, wie mit HLC-Clustering zwei ineinander liegende Ringe gruppiert wurden. So etwas ist mit k-means nicht möglich.

Da in diesem Projekt keine ringförmigen Abbildungen von zusammengehörenden Texten gecluster werden, ist dies nicht weiter tragisch. Die Zusammengehörigkeit stellt sich eher als "Punkte-Haufen" dar. Das bedeutet, Punkte die zusammen gehören, liegen nahe beieinander und müssen nicht anhand ihrer räumlichen Form erkannt werden.

Dies ist genau, was die, verwendete Implementation macht. Sie gruppiert räumlich nahe beieinander liegende "Punkte".

7.3 Verwandte Typen

Die Grundidee des Clustering ist, ähnliche Objekte einer gemeinsamen Gruppe zu zuordnen (siehe Kapitel 7.2). Diese Projekt geht jedoch einen Schritt weiter. Es soll nicht nur erkannt werden, dass "Berlin" eine Stadt oder "Roger Federer" ein Sportler ist, es sollen auch Aussagen wie: "Roger Federer ist ein Schweizer Tennis Spieler" gemacht werden können. Weiter sollen beispielsweise alle Städte Deutschlands angezeigt werden.
Mit herkömmlichem Clustering ist so etwas nicht möglich. Der Unterschied zwischen einer deutschen oder schweizer Stadt, wäre zu gering. Zudem ist eine Gruppierung nach "Schweizer Tennis Spieler" wenig erfolgversprechend. Viel zu viele Beispieldaten müssten vorgegeben werden. Es wären Prototypen für z.B: "Schweizer Tennis Spieler", "Deutscher Tennis Spieler", "Spanischer Tennis Spieler", "Schweizer Rad Spieler", "Schweizer Pferde Sportler" usw. nötig. Ein unmöglicher Aufwand.

Lösung:
Die Beziehung zwischen Elementen verschiedener Cluster wird zusätzlich abgebildet. Dafür ist es nötig, ein Netz mit den gewünschten Relationen zwischen den Typen zu definieren.

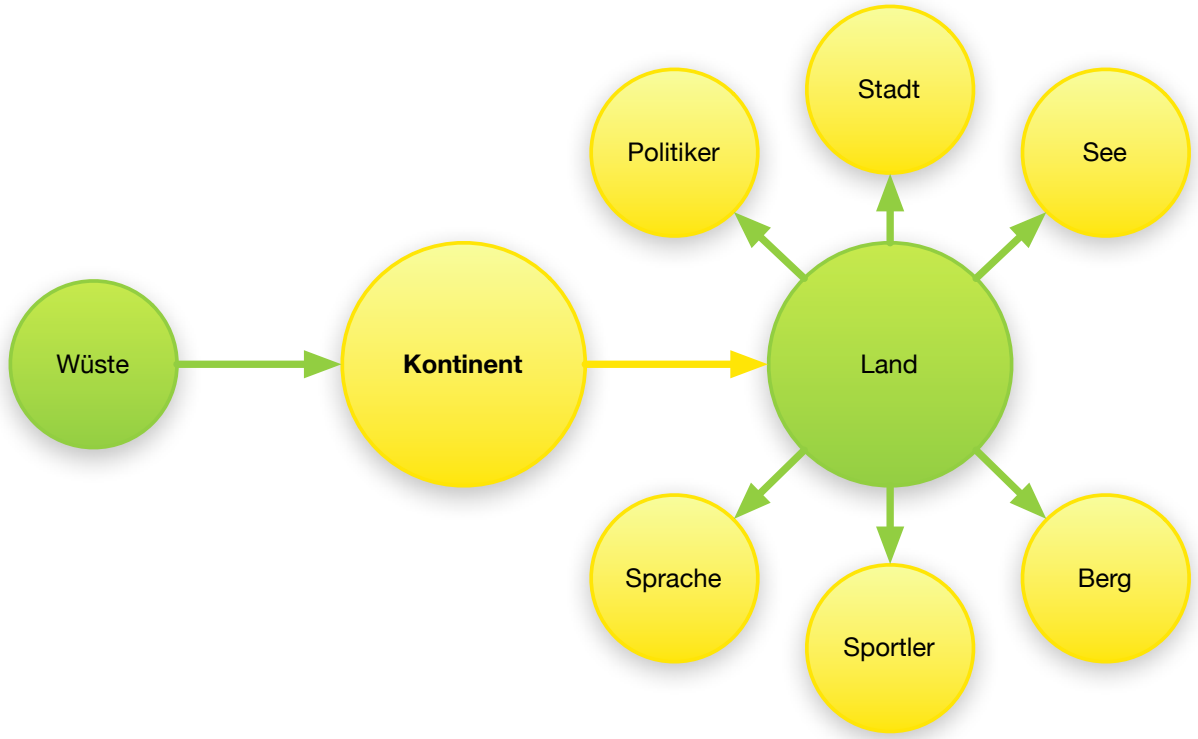


Abbildung 7.2: Ausschnitt der Type-Database

Anhand dieser Vorgaben werden die Beziehungen unter den einzelnen Elementen berechnet. Eine Beziehung ergibt sich durch die Distanz zwischen den einzelnen Elementen. Da sich alle Elemente im selben Vektorraum mit n-Dimensionen befinden, steht grundsätzlich jedes Element mit jedem anderen in Beziehung. Da eine solche "many to many" Beziehung wiederum nichts mehr aussagen würde, beschränkt man sich auf "1 to many" Beziehungen.
Somit kann beispielsweise definiert werden, dass jeder Sportler mit dem entsprechenden Land und jedes Land mit dem entsprechenden Kontinent verknüpft werden soll. Dabei wird, in diesem Fall, jedes Sportler-Element mit dem, ihm am nächsten stehenden Land-Element verknüpft und dieses mit dem, ihm am nächsten gelegenen Kontinent-Element. Für die Verknüpfung wird jeweils das Element mit der minimalsten Distanz verwendet. Dieses System basiert auf der Annahme, dass zusammengehörende Elemente nahe beieinander sind, obwohl sie unterschiedlichen Gruppen angehören.
Beispiel: Das Element "Schweiz" gehört zur Gruppe "Land". Aus der Gruppe "Kontinent" wird das Element mit der geringsten Distanz zum Element "Schweiz" gesucht: "Europa". Jeder andere Kontinent hat eine geringere Ähnlichkeit zur Schweiz. Daraus wird geschlossen, dass die Schweiz ein Land in Europa ist.

7.4 Verteiltes System

Die gesamte Informationsbeschaffung des Lumin Projekts ist als verteiltes System realisiert. Dabei agieren die einzelnen Teile als unabhängige verteilte Komponenten, die über einen zentralen JMS Provider kommunizieren. (siehe Kapitel 6.1)

7.4.1 Idee

Die Datenanalyse in der Studienarbeit 'Ziirp' von Kevin Gaunt und Tobias Löffel, nahm extrem viel Rechenzeit in Anspruch. Analog dazu, müssen in diesem Projekt Daten über komplexe Analysen aufbereitet werden. Freie Rechner-Kapazität steht immer zur Verfügung, wieso also, sollte man diese nicht dafür nutzen? Die Idee, mehrere Agenten zu verteilen, welche die Aufträge entgegennehmen, Daten analysieren und die Daten zur Auswertung weitergeben, wurde schliesslich umgesetzt. Da sich grundsätzlich jeder mit einem Agenten am Projekt beteiligen kann, ist die Umsetzung so gewählt, dass die einzelnen Komponenten keine Kenntnisse voneinander haben müssen. Die Kommunikation verläuft nicht peer-to-peer, sondern über einen zentralen JMS Provider.
Die Kommunikation vom Master zu den Agenten, ist als "Point-to-Multipoint-Queue" realisiert. Die Kommunikation von den Agenten zum Analyzer, als "Multipoint-to-Point-Queue".

7.4.2 Master

Der Master ist der Auftraggeber. Er generiert Requests für die Clients. Hier wird bestimmt, welche Begriffe analysiert werden sollen. Dafür stellt der Master die Aufträge in eine JMS-Queue. Dabei könnte für jeden Agenten-Typen eine Queue verwendet werden. Momentan existiert nur ein Agenten-Typ und daher nur eine Auftrags-Queue.

7.4.3 Agent

Die Agenten "hören" die Auftrags-Queue ab und nehmen die anstehenden Aufträge entgegen, analysiert die Begriffe und liefern das Resultat an den Analyzer weiter. Dabei können beliebig viele Agenten die Aufträge parallel bearbeiten. Diese Flexibilität ist ein grosser Vorteil. Wenn viele Daten analysiert werden müssen, werden einfach mehrere Agenten zugeschaltet. Dabei soll jeder Auftrag pro Agent-Typ genau einmal erledigt werden. Diese Anforderung wird durch die JMS-Queue sichergestellt.

Agenten-Steuerung

Grundsätzlich kann ein Agent von der Maschine, auf der er gestartet wird, gesteuert werden. Es gibt aber auch die Möglichkeit, alle Agenten über den HostMaster zu koordinieren. Dabei sieht der Master den Status aller gestarteten Agenten. Über den Master lassen sie sich starten und stoppen und auch der Agenten-Typ (Informationsquelle), kann geändert werden. Wenn z.B. die Analyse einer Quelle mehr Verarbeitungszeit in Anspruch nimmt, werden einfach einzelne Agenten auf diesen Agenten-Typ umgeschaltet.

Wie vorhin erwähnt, hat keine Komponente Kenntnisse über die Anderen. Für die reine Auftrags-Kommunikation ist dies kein Problem, da kein Agent direkt identifiziert werden muss. Wenn nun aber ein spezieller Agent gestartet werden soll, ist dies der Fall . Für die Steuerung der Agenten wird daher eine UUID (Universally Unique Identifier) für jeden Client verwendet. Steuerungs-Nachrichten werden auch weiterhin nicht an den Client direkt adressiert. Sie gehen an ein spezielles JMS-Topic (siehe

Kapitel 5.3.1). Alle Clients empfangen die Nachrichten zu diesem Topic. Der "HostMaster" versieht die Steuerungs-Nachricht mit der, dem Client entsprechenden UUID. Clients reagieren nur auf Steuerungs-Nachrichten, die ihrer UUID entsprechen.

Somit wird eine Agenten-Adressierung realisiert, ohne dass der Sender den Agent direkt kennen muss. Er benötigt nur seine UUID. Diese propagiert der Client ebenfalls über ein JMS-Topic.

7.4.4 Analyzer

Der Analyzer nimmt die, vom Agenten verarbeiteten Daten entgegen und analysiert diese. Dabei werden die Begriffe anhand der dazugehörigen Vektoren geclustert (siehe Kapitel 7.2).

Der Analyzer schreibt die Resultate direkt in die Datenbank. Dieser Schritt geschieht auf demselben System, könnte jedoch auch noch verteilt werden. Zurzeit laufen Analyse und Datenhaltung noch auf demselben System.

7.5 Visualisierung

Der Entscheid, die Visualisierung des Systems mit Processing[5] zu realisieren, fiel relativ früh im Projekt. Nach kurzer Einarbeitungszeit hat man sehr schnell festgestellt, dass die ganze Processing-Umgebung sehr simpel und vor allem für Gestalter, Künstler und Programmieranfänger ausgerichtet ist. So wurde beschlossen, eine eigene Schnittstelle zu Processing zu entwickeln, die mehr Funktionalität bietet und so sehr einfach in das bestehende Projekt eingebunden werden kann.

7.5.1 Components

Das Konzept sieht vor, dass die ganze Visualisierung mit einzelnen Bausteinen, sogenannten Components, realisiert wird. Diese Components können unabhängig voneinander entwickelt und jederzeit in die Visualisierung eingebunden werden. Da alle Components das gleiche, selbst definierte Interface implementieren, ist die Kompatibilität gewährleistet.

7.5.2 Controllers

Jede Component besitzt einen Controller, der die Ansteuerung deren erlaubt. Dadurch können Nachrichten übermittelt und angezeigt werden oder die Component ein- oder ausgeblendet werden. Ein Controller kann auch von einem anderen Controller aus benutzt werden, damit die Components untereinander kommunizieren können.

7.5.3 Schnittstelle zu Processing

Der Hauptbestandteil der Visualisierung, ist die Schnittstelle zu Processing. Diese Schnittstelle ist in einer Klasse realisiert, die von Processing abgeleitet ist und die Methoden beinhaltet, um die Visualisierung zu erstellen und die Components darauf zu zeichnen. Ausserdem nimmt die Klasse Mouse- und KeyEvents entgegen und leitet diese, an die jeweiligen Components, respektive deren Controller weiter.

[5] `http://www.processing.org/`

7.6 Datenhaltung

Im entwickelten System werden zwei Arten von Datenhaltung unterschieden. Einerseits muss ein analysierter Begriff abgespeichert werden, andererseits werden die Relationen zwischen diesen Begriffen in einer Datenbank gespeichert und bereits persistierte Begriffe mit einer oder mehreren Relationen verknüpft.

7.6.1 Serialisierte Objekte

Für die Speicherung eines analysierten Begriffs wurde beschlossen, dies mit dem frei verfügbaren Simple-Framework[6] zu realisieren. Dieses Framework zeichnet sich dadurch aus, dass auf einfachste Art und Weise ein komplettes Objekt in XML serialisiert und wieder deserialisiert werden kann. Annotations erlauben einen einfachen Umgang, mit den zu serialisierenden Elementen und lassen dem Entwickler grosse Freiheiten, wie er damit umgehen möchte.

Code Listing 7.1 XML-Serialisierungs Schema

```
<clusterItemImpl searchTerm="Indianapolis" type="Stadt" distanceToCentre="0.25433501676351616"
  confirmed="true" unknown="false">
   <tagMagnitudeVector class="TagMagnitudeVectorImpl">
     <tagMagnitudesMap class="HashMap">
       <entry>
         <tagMagnitude class="TagMagnitudeImpl" magnitude="0.03283545516515592">
         <tag class="TagImpl" displayText="koordinaten" stemmedText="koordinaten" hashCode="381210622" />
         </tagMagnitude>
       </entry>
         <entry>
         <tagMagnitude class="TagMagnitudeImpl" magnitude="0.01641772758257796">
         <tag class="TagImpl" displayText="volkszählungsergebnisse" stemmedText="volkszählungsergebniss"
         hashCode="1184918052" />
         </tagMagnitude>
       </entry>
       <entry>
         <tagMagnitude class="TagMagnitudeImpl" magnitude="0.01641772758257796">
         <tag class="TagImpl" displayText="schulbezirk" stemmedText="schulbezirk" hashCode="251520474" />
         </tagMagnitude>
       </entry>
       <entry>
         <tagMagnitude class="TagMagnitudeImpl" magnitude="0.03283545516515592">
         <tag class="TagImpl" displayText="städtepartnerschaften" stemmedText="städtepartnerschaften"
         hashCode="-300769546" />
         </tagMagnitude>
       </entry>
     </tagMagnitudesMap>
   </tagMagnitudeVector>
</clusterItemImpl>
```

Der XML-Ausschnitt 7.1 zeigt, wie ein Begriff mit dazugehörigen Attributen abgespeichert wird. Dabei sind einige Beispieleinträge abgebildet.

[6] `http://simple.sourceforge.net/`

7.6.2 Datenbank

Als Datenbank wird Neo4j[7] eingesetzt. Dies ist eine graph-basierte Datenbank, die einen sehr schnellen Zugriff auf die einzelnen Nodes erlaubt. Aufbauend auf diesem Datenbank-Framework, wurde ein Datenbank-Layer entwickelt, der einen Service anbietet, um auf die Datenbank zuzugreifen. Auf diesem Datenbank-Service setzt der Node-Handler-Service auf. Dieser wird gebraucht, um Objekte in die Term- oder Type-Datenbank zu schreiben und wieder daraus zu lesen. Diese Objekte werden als Plain Old Java Objects (POJO) realisiert, was zu einer niedrigen Kopplung und hoher Kohäsion führt (siehe Abbildung 7.3).

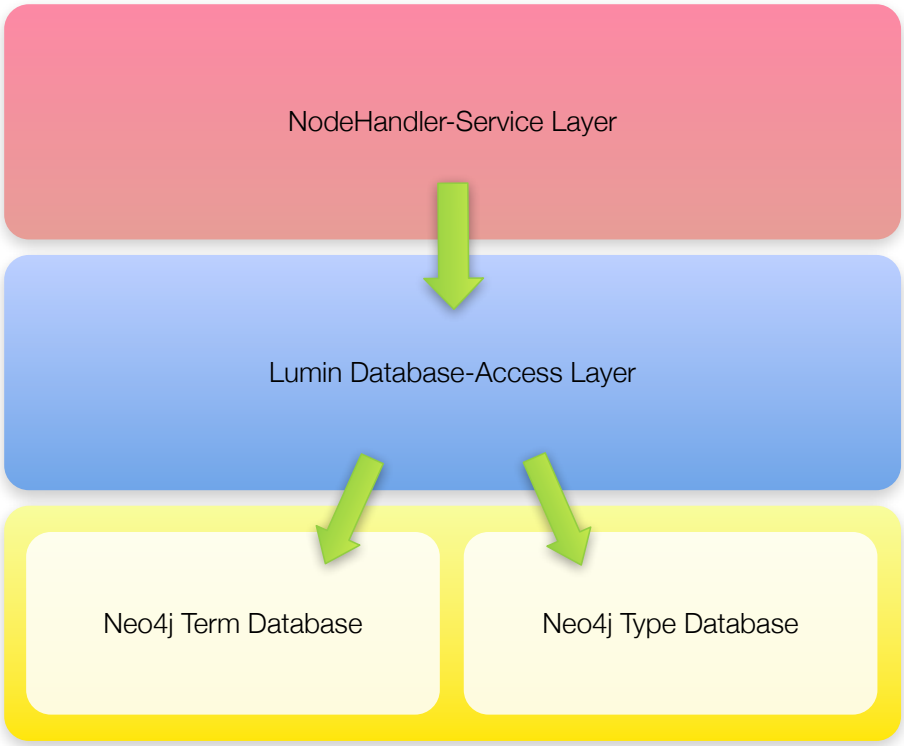


Abbildung 7.3: Illustration der Datenbank Schichten

Wie bereits erwähnt, werden zwei verschiedene Datenbanken genutzt. Die Term-Datenbank beinhaltet die abgespeicherten Begriffe, sowie die Relationen zueinander. In der Type-Datenbank wird das Schema abgespeichert, wie die einzelnen Typen miteinander verknüpft sind. Um die Instanzen der verschiedenen Datenbanken zu erzeugen, wird auf dem Database-Layer eine Factory eingesetzt. Damit wird erreicht, dass der Entwickler, der den Node-Handler-Service benutzt, sich nicht um die Instanzierung der Datenbanken kümmern muss, sondern lediglich die gewünschte Methode aufrufen kann, um entweder mit der Term- oder der Type-Datenbank zu arbeiten.

[7] `http://neo4j.org/`

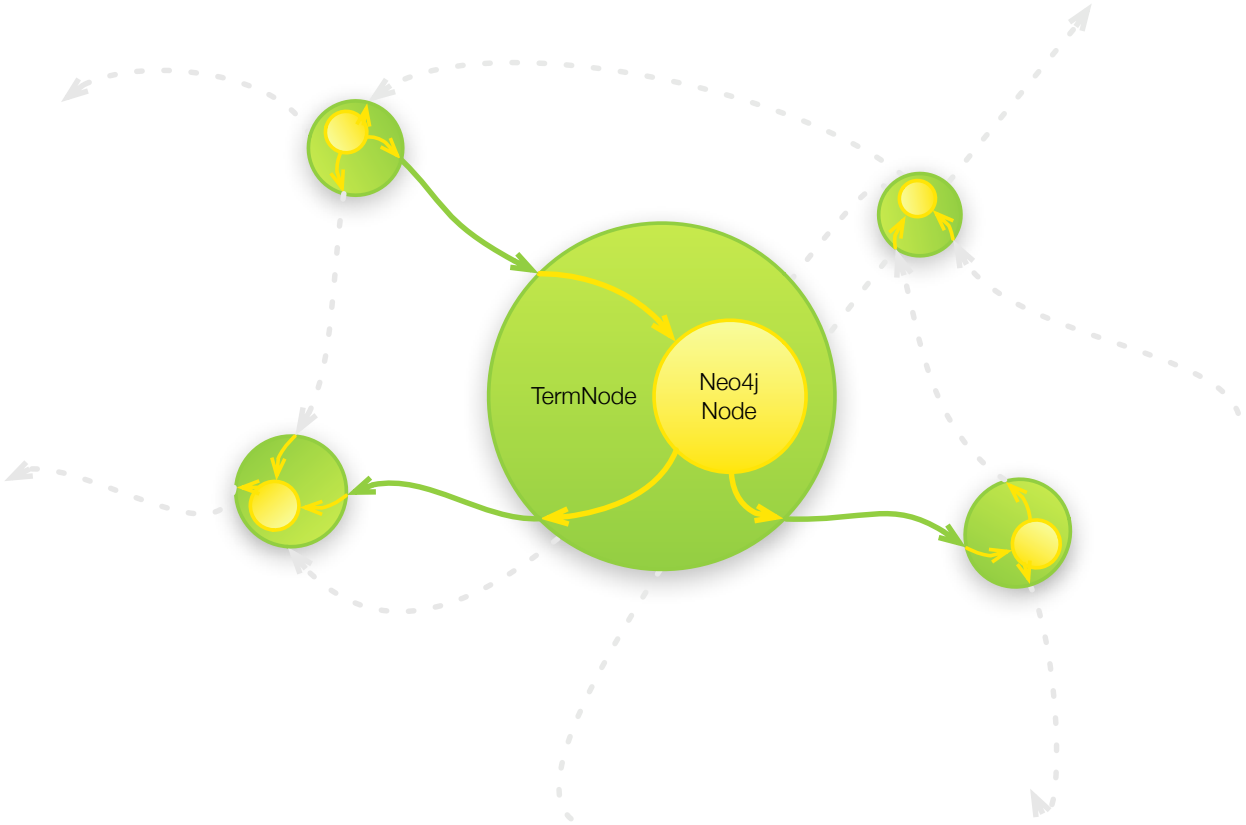


Abbildung 7.4: Ansicht gewrappter Graph-Nodes

7.7 Wikipedia Parser

Um die erforderlichen Informationen zu einem Begriff zu bekommen, wird die dazugehörige Wikipedia-Seite geparst und die relevanten Informationen extrahiert. Ein erster Ansatz bestand darin, die Konvention auszunutzen, dass jede Wikipedia-Seite die gleiche URL, gefolgt vom gesuchten Begriff hat. So kann die Seite zur Schweiz folgendermassen erreicht werden: `http://de.wikipedia.org/wiki/Schweiz`. Schnell wurde jedoch festgestellt, dass diese Methode einige Tücken aufweist. So müssen Begriffe, die aus mehreren Wörtern bestehen, mit einem Underscore geschrieben werden. Wikipedia beachtet auch die Gross- und Kleinschreibung von Wörtern, was nicht überall konsistent gelöst wurde, so dass immer wieder Begriffe auftauchten, die nicht gefunden werden konnten, da die Schreibweise von der Norm abweicht.

Der zweite und nun auch eingesetzte Ansatz, basiert auf einem Http-Request. Dazu wird der Parameter, nach dem gesucht werden soll, einem Post-Request übergeben, der eine Suchanfrage über das Suchformular simuliert. Danach kommt eine Antwort mit dem gewünschten Inhalt zurück, die nun geparst werden kann. Diese Lösung hat den Vorteil, dass das System viel flexibler ist, da die Gross- und Kleinschreibung nicht mehr berücksichtigt werden muss. Ebenfalls ist es nicht mehr nötig, die genaue Schreibweise des Begriffs zu kennen. Wenn nun ein Benutzer einen Begriff eingibt, der von Wikipedia nicht klar zugeordnet werden kann, bekommt er eine Antwort mit verschiedenen Auswahlmöglichkeiten, die Wikipedia vorschlägt. Für die Implementierung des Http-Post-Requests, wurde die frei verfügbare Klasse ClientHttpRequest von Vlad Patryshev verwendet[8].

Für die Analyse eines Begriffes, sind nur die Nomen eines Textes relevant, da die anderen Wörter in einem Grossteil der Fälle, keine wichtigen Informationen beinhalten oder nur Füll- und Bindewörter sind. Der Wikipedia Parser extrahiert deshalb nur die Nomen und verwirft alle anderen Wörter. Daraus müssen noch Wikipedia spezifische Nomen herausgefiltert werden, die überall vorkommen und für die Analyse nicht von Bedeutung sind.

[8] `http://www.myjavatools.com/`

Kapitel 8

Resultate

Nach den verstrichenen 16 Wochen Projektarbeit, beziehungsweise 1130 geleisteten Arbeitsstunden, wurden die Anforderungen, die an diese Arbeit gestellt wurden, erfüllt. Zum jetzigen Zeitpunkt befinden sich über 5500 analysierte Begriffe im System. Dieses System läuft soweit stabil, wie es die Anforderungen an einen wissenschaftlichen Prototypen vorsehen. Die gewählte Architektur der Datenbeschaffung arbeitet verteilt auf beliebig vielen Rechnern, was die Skalierbarkeit erhöht, da bei vielen Suchanfragen entsprechend mehr Agenten eingesetzt werden können, welche die benötigten Informationen beschaffen. Die gesammelten Informationen werden in einer graph-basierten Datenbank gespeichert und dem Benutzer über ein ansprechendes GUI dargestellt.

8.1 Wissensextraktion

Der Prototyp ist in der Lage, beliebige Begriffe zu kategorisieren und diese, mit zusammenhängenden Begriffen zu verknüpfen. Dafür wird der, dem Begriff entsprechende, Wikipedia Artikel geparst, gefiltert und in einen gruppierbaren Vektor abgebildet. Dies ermöglicht dem System, die Ähnlichkeit der Begriffe zu den Beispielgruppen zu erkennen. Dabei muss nicht definiert werden, was ein Begriff beschreibt. Es reicht, einige Beispiele zu einer Kategorie vorzugeben.

Beispielsweise muss nur vorgegeben werden, dass “Italien”, “China”, “Brasilien”, “Neuseeland” und “Senegal” Länder sind. Dann wird bei einer Anfrage zur Schweiz automatisch erkannt, dass es sich dabei ebenfalls um ein Land handelt. Somit lassen sich innert kürzester Zeit, sämtliche Länder der Welt bestimmen.

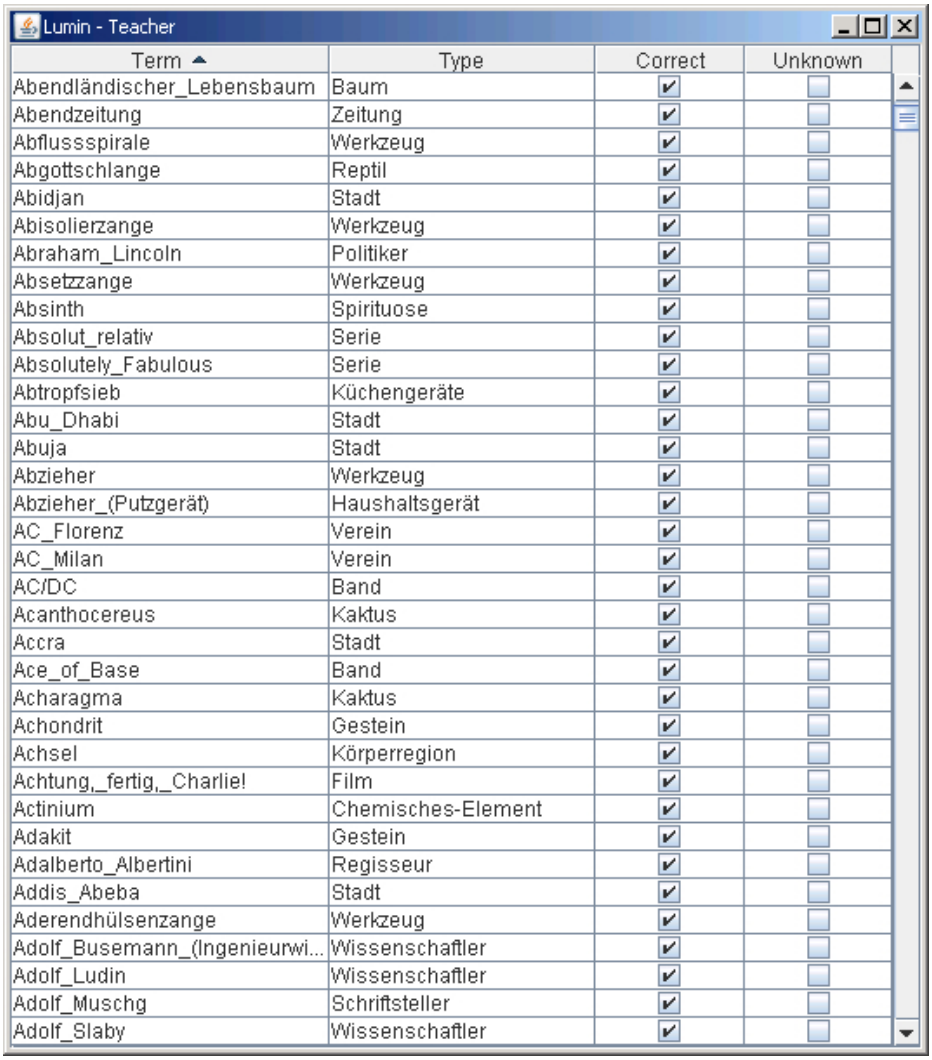
Nebst der reinen Gruppierung, können auch Beziehungen unter den einzelnen Begriffen erkannt werden. Somit ist es nicht nur möglich alle Länder zu erkennen, sondern es werden auch alle Länder Europas mit dem Kontinent “Europa” verknüpft. Das ermöglicht dem System zu bestimmen, dass “Zürich” eine Stadt der “Schweiz” ist und zum Kontinent “Europa” gehört. Dabei lässt sich jeder beliebige existierende Begriff kategorisieren. Voraussetzung ist, dass eine Quelle vorhanden ist, die den Begriff beschreibt.

Das Analyseverfahren ist grundsätzlich unabhängig von der Quelle, solange es sich dabei um Text handelt, der Informationen zum gegebenen Begriff beinhaltet.

8.2 Datenhaltung

Die analysierten Objekte werden in eine XML-Datei serialisiert. Dies bietet den Vorteil, dass die Datei mit jedem XML Viewer angezeigt und inspiziert werden kann. Wenn die Informationen zu einem Begriff wieder gebraucht werden, kann das serialisierte Objekt ganz einfach wieder deserialisiert und ins System eingelesen werden.

Der Term und der Type eines analysierten Objekts, sowie die dazugehörigen Relationen, werden zudem in einer graph-basierten Datenbank gespeichert. Ebenfalls gespeichert, werden die Verknüpfungen zwischen den verschiedenen Begriffen. Es wurde ein eigener Datenbank Layer entwickelt, der von der Datenbank entkoppelt werden könnte, wenn die darunter liegende Datenbank einmal ausgewechselt werden sollte.



Term	Type	Correct	Unknown
Abendländischer_Lebensbaum	Baum	✔	
Abendzeitung	Zeitung	✔	
Abflussspirale	Werkzeug	✔	
Abgottschlange	Reptil	✔	
Abidjan	Stadt	✔	
Abisolierzange	Werkzeug	✔	
Abraham_Lincoln	Politiker	✔	
Absetzzange	Werkzeug	✔	
Absinth	Spirituose	✔	
Absolut_relativ	Serie	✔	
Absolutely_Fabulous	Serie	✔	
Abtropfsieb	Küchengeräte	✔	
Abu_Dhabi	Stadt	✔	
Abuja	Stadt	✔	
Abzieher	Werkzeug	✔	
Abzieher_(Putzgerät)	Haushaltsgerät	✔	
AC_Florenz	Verein	✔	
AC_Milan	Verein	✔	
AC/DC	Band	✔	
Acanthocereus	Kaktus	✔	
Accra	Stadt	✔	
Ace_of_Base	Band	✔	
Acharagma	Kaktus	✔	
Achondrit	Gestein	✔	
Achsel	Körperregion	✔	
Achtung,_fertig,_Charlie!	Film	✔	
Actinium	Chemisches-Element	✔	
Adakit	Gestein	✔	
Adalberto_Albertini	Regisseur	✔	
Addis_Abeba	Stadt	✔	
Aderendhülsenzange	Werkzeug	✔	
Adolf_Busemann_(Ingenieurwi...	Wissenschaftler	✔	
Adolf_Ludin	Wissenschaftler	✔	
Adolf_Muschg	Schriftsteller	✔	
Adolf_Slaby	Wissenschaftler	✔	

Abbildung 8.1: GUI des Korrekturprogrammes

Mit Hilfe des GUIs in Abbildung 8.1 können im System erfasste Einträge bestätigt oder geändert werden. Die Datei wird nach jeder Änderung sofort gespeichert.

8.3 Visualisierung

Es ist ein, auf Processing[1] basierendes, GUI entstanden, das für den Benutzer einfach und intuitiv zu bedienen ist.

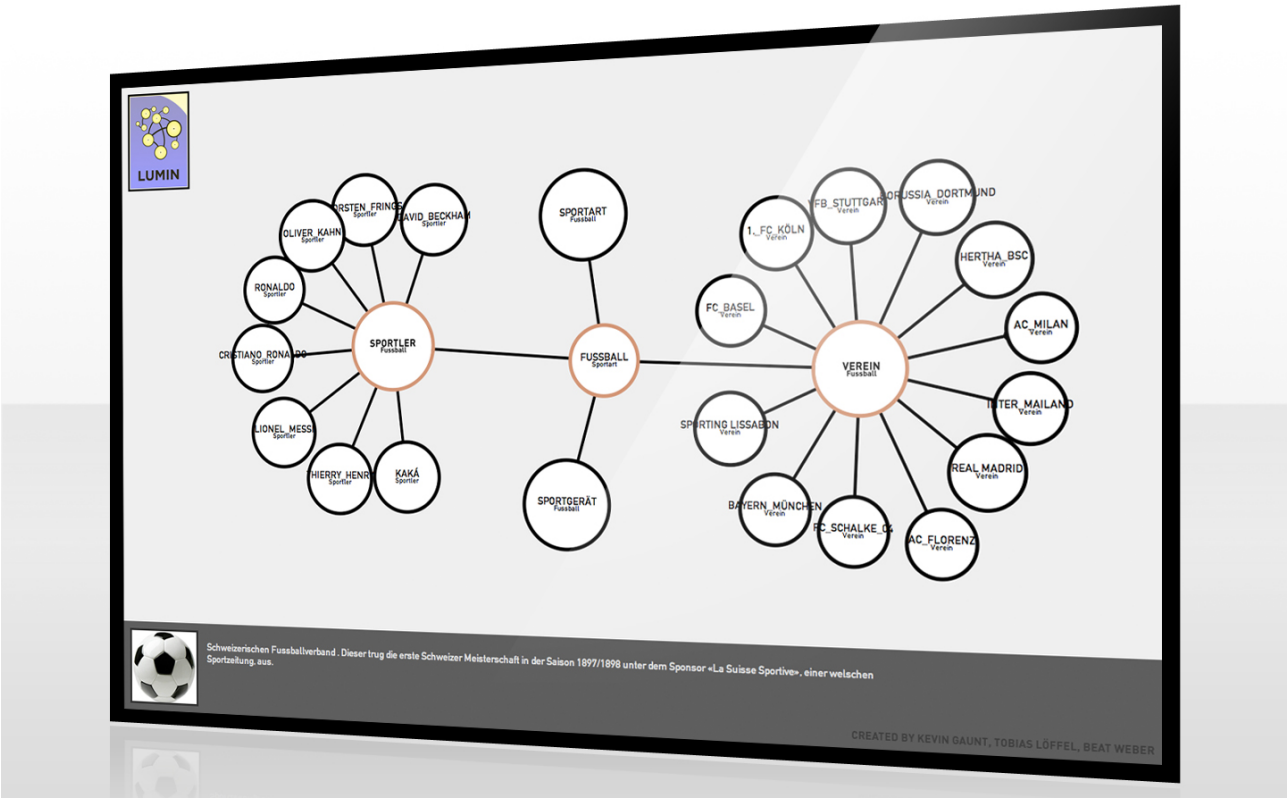


Abbildung 8.2: Screenshot der Visualisierung

[1] `http://www.processing.org/`

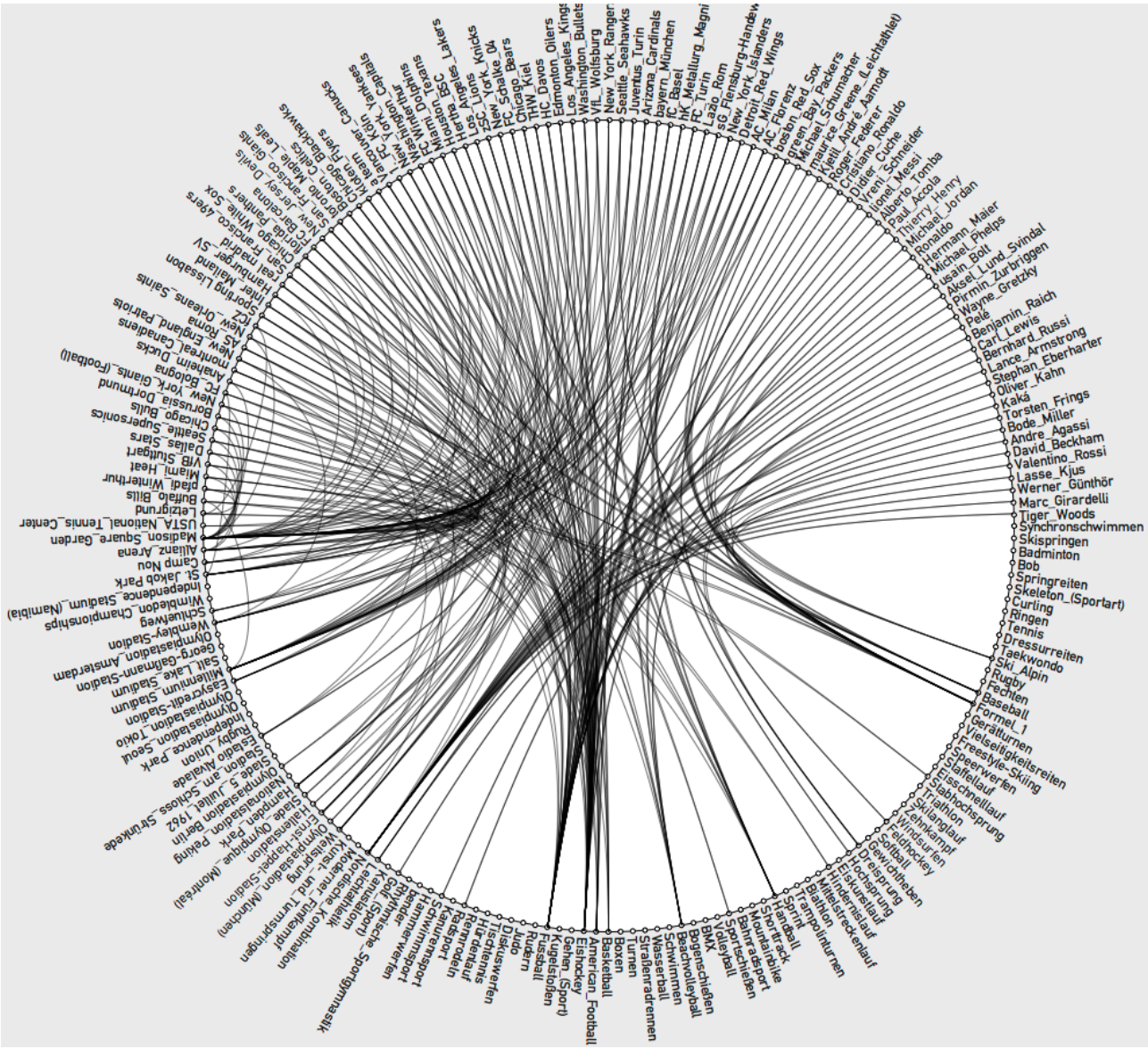


Abbildung 8.3: Alternative Darstellungsweise der verwandten Begriffe

Abbildung 8.3 zeigt eine alternative Darstellungsweise der verwandten Begriffe. Diese Ansicht wurde im Prototypenstadium entwickelt.

8.4 Verteiltes System

Durch die Nutzung von JMS, schliessen sich die einzelnen Teile zu einem unabhängigen, verteilten System zusammen. Durch die so geschaffene lose Kopplung, sind einzelne Komponenten leicht austauschbar. Um die Analyse zu beschleunigen, lassen sich beliebig viele Agenten zuschalten, die Aufträge parallel abarbeiten, was das Ganze sehr skalierbar macht. Ausfälle einzelner Agenten haben dabei keinen Einfluss auf das Gesamtsystem. Durch die Verwendung der JMS-Schnittstelle, liessen sich auch Komponenten in Java fremden Programmiersprachen, ohne grösseren Zusatzaufwand betreiben.

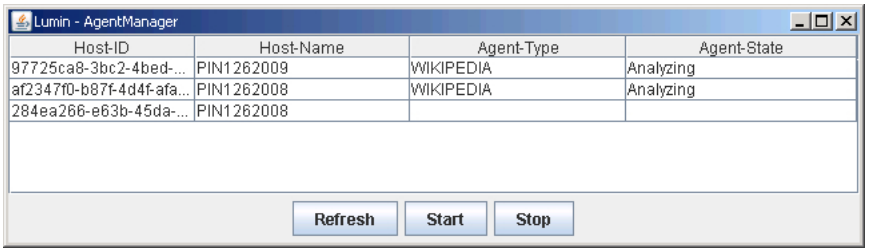


Abbildung 8.4: GUI des Agent Managers

Mit dem GUI in Abildung 8.4 können die verteilten Agents gesteuert werden (siehe Kapitel 7.4.3).

Kapitel 9

Probleme

9.1 JMS-Server

Für die Arbeit wird ein Virtualserver eingesetzt, der in der DMZ der HSR-Domain steht. Dieser Virtualserver wird als JMS-Broker eingesetzt und sollte auch von ausserhalb zugänglich sein. Da jedoch die Informatikdienste der HSR nur die Ports 21, 80 und 8080 nach aussen geöffnet haben, JMS jedoch die Ports 3030 und 3035 benötigt, funktionierte die Kommunikation mit einem Client ausserhalb der HSR-Domäne nicht. Der Informatikdienste war leider nicht gewillt, dafür eine gemeinsame Lösung zu finden. Dieses Problem wurde dann so gelöst, indem der JMS-Server auf die well-known Ports 21 und 80 umkonfiguriert wurde und so die Kommunikation nach aussen ohne Probleme möglich wurde. Da der Virtualserver nicht als Web- oder FTP-Server eingesetzt wird, stellt dieser Eingriff kein Problem dar.

9.2 Mehrfachbeziehungen zwischen Terms nicht möglich

Mehrfachbeziehungen sind in der implementierten Version nicht möglich. Dies hat den Grund, dass die Distanz, die ein Begriff maximal von einem verwandten Begriff abweichen darf, nicht klar definiert werden kann. Ein Schauspieler sollte nach Möglichkeit mit allen Filmen verknüpft werden, in denen er mitgespielt hat. Ein Film wiederum soll mit allen Schauspieler verknüpft werden, die darin mitspielen. Da aber nur die Beziehung in eine Richtung genau definiert werden kann, lässt sich zu jedem Film genau ein entsprechender Darsteller finden. Ein Schauspieler kann so in mehreren Filmen Spielen. Ein Film hat jedoch immer nur einen Darsteller. Da die Anzahl Schauspieler unterschiedlich ist, müsste das System selber eine Grenze ziehen können, ob der ein Film einem Schauspieler zugeordnet wird oder eben nicht. Die Bestimmung dieser Grenze konnte bis jetzt noch nicht realisiert werden, ist aber ein wichtiger Punkt, wenn es um die Weiterentwicklung des Systems geht (siehe Kapitel 11.2.3).

9.3 Datenhaltung

Wenn die Datenbank komplett neu aufgebaut werden soll, besteht das Problem, dass ein Knoten mit einem anderen verknüpft werden soll, der eventuell noch gar nicht gespeichert ist. Als Beispiel könnte man die Schweiz nehmen. Anhand des Schemas in der Type-Datenbank ist bekannt, dass die Schweiz, wenn sie als Land erkannt wird, mit einem Kontinent verbunden werden soll. Wenn nun der Cluster mit

den Kontinenten noch nicht analysiert und in der Datenbank gespeichert wurde, ist es nicht möglich diese beiden Knoten zu verknüpfen. Für dieses Problem gibt es zwei unterschiedliche Lösungsansätze. Entweder man bestimmt die Reihenfolge der Cluster, wie sie analysiert und gespeichert werden sollen. Dies ist jedoch überhaupt nicht geeignet, da sie die dynamische Erweiterung der Typen-Datenbank erschwert und nach jedem Einfügen eines neuen Types angepasst werden muss. Der zweite Ansatz, der auch realisiert wurde, speichert zuerst alle Knoten auf der Datenbank und verknüpft sie erst in einem zweiten Schritt. Dadurch dauert der gesamte Vorgang etwas länger, die Type-Datenbank wird aber in keiner Weise beeinträchtigt und kann weiterhin ganz einfach erweitert werden.

Ein weiteres Problem besteht darin, dass ein Element immer mit einem anderen verknüpft sein muss, wenn es das Schema in der Type-Datenbank so vorsieht. Wenn nun das Stadion eines Fussballvereins noch nicht im System vorhanden ist, wird der Fussballverein trotzdem mit einem Stadion verknüpft, das ihm am nächsten liegt. Dadurch entstehen Fehler in der Datenbank, die unumgänglich sind, bis alle benötigten Elemente gespeichert sind. Da dies sehr schwierig zu realisieren ist, wird es wohl immer Elemente geben, die falsch verknüpft sind, weil der richtige Knoten noch nicht erfasst ist. Dieses Problem kann dadurch gelöst werden, dass so viele Datensätze wie möglich gespeichert werden. Da die Datenbank jede Nacht neu erstellt wird und dadurch die Elemente wieder neu analysiert und verknüpft werden, wird die Anzahl der richtig verknüpften Knoten laufend erhöht.

9.4 Unterschiedliche Zeichensätze der Textquellen

Wenn eine Seite aus dem Internet geparst wird, ist zu Beginn nicht klar, mit welchem Zeichensatz diese Seite codiert ist. Nun trat das Problem auf, dass die Seite mit einem Zeichensatz codiert wurde und mit einem anderen eingelesen und im System abgespeichert wurde. In erster Linie trat das Problem bei Wikipedia Seiten auf, die mit dem Zeichensatz UTF-8 codiert sind. Beim Einlesen und Speichern im System, wird der Windows Standard ISO-8859-1 verwendet, was beispielsweise dazu führte, dass gewisse Sonderzeichen nicht richtig dargestellt werden konnten. Das Problem wurde so gelöst, dass zu Beginn des Parse-Vorgangs einer Internetseite, der Zeichensatz der jeweiligen Seite ausgelesen, diese dann in UTF-8 konvertiert und auch in diesem universellen Zeichensatz gespeichert wird.

Kapitel 10

Entscheidungen

10.1 Datenquellen

Zu Beginn des Projektes bestand die Idee, verschiedene Informationsquellen für die Datenbeschaffung zu nutzen. Verschiedene Tests haben jedoch ergeben, dass die Qualität der einzelnen Quellen starke Unterschiede aufweist. Um die Qualität der verschiedenen Quellen zu testen, wurden verschiedene Agents geschrieben, welche die Informationen von den jeweiligen Seiten geladen und extrahiert haben. Danach wurden die Ergebnisse untereinander verglichen und es stellte sich ganz klar heraus, dass Wikipedia vom Umfang und der Qualität her, die mit Abstand besten Informationen liefert. Aus diesem Grund ist Wikipedia zur Zeit die einzige eingesetzte Quelle. Das System ist aber so flexibel aufgebaut, dass es kein Problem ist, zu einem späteren Zeitpunkt weitere Quellen einzubinden, wenn dadurch noch bessere Ergebnisse erzielt werden können.

10.2 Datenbank

Nach einer Evaluationsphase wurde der Entscheid gefällt, die graph-basierte Datenbank Neo4j einzusetzen. Der grosse Vorteil einer graph-basierten Datenbank besteht darin, dass das Lesen, Löschen und Einfügen von Nodes in konstanter Zeit $O(1)$, sowie das Traversieren von Daten linear ist $O(n)$. Für Schreiboperationen ist die Datenbank etwas langsamer, besonders bei einzelnen Transaktionen. Das Hauptaugenmerk des Projekts liegt aber ganz klar auf der Beschaffung von Daten aus der Datenbank, so dass der Nachteil beim Schreiben vernachlässigt werden kann. Der Hersteller verspricht eine Skalierbarkeit von mehreren Milliarden Elementen, auf einer einzigen Maschine. Die Datenbank kann bei Bedarf auf mehrere Rechner verteilt werden[1].

Falls die Arbeit zu einem späteren Zeitpunkt kommerziell genutzt werden sollte, muss eine entsprechende Lizenz erworben werden. Im Moment wird die Datenbank mit einer freien Lizenz zu nichtkommerziellen Zwecken eingesetzt.

[1] `http://neo4j.org/`

10.3 Clustering

Neben dem implementierten Clustering (siehe Kapitel 5.2.4) wurde auch eine Variante mit dem Hierarchical Agglomerative Clustering (HAC) Algorithmus analysiert. Dabei wurde Ward Clustering (siehe Kapitel 5.2.5) verwendet. Die erzielten Resultate erwiesen sich jedoch als bedeutend schlechter. Zum Einen wurden mehr fehlerhafte Zuordnungen gemacht und zum Anderen, besteht bei hierarchischem Clustering die Schwierigkeit, zu wissen, auf welcher Hierarchiestufe die Cluster gewertet werden sollen.

Für dieses Projekt sollen Elemente nur genau einem Cluster angehören. Daher ist die Hierarchie eher ein Hindernis als ein Vorteil. Weiter zeigte sich in den Versuchen auch die Tatsache, dass hierarchische Clustering Algorithmen viel rechenintensiver sind.

> Hierarchical algorithms don't scale well. If n is the number of items then the order of complexity is n2. [3]

Aus diesen Gründen wurde eine, an k-means angelehnte Clustering Berechnung verwendet. Die Entscheidung erwies sich als richtig. Die Erfolgsrate von mehr als 95% der Tests am Anfang, blieb für den gesamten Projektverlauf konstant. Auch das Clustering, sehr vieler, mehr dimensionaler Elemente, stellt kein Problem dar.

10.4 Visualisierung

Bereits zu Beginn der Arbeit war klar, dass für die Visualisierung ein modernes, bestehendes Framework eingesetzt werden soll. Processing[2] ist ein freies, auf Java2D basiertes Framework für die Erstellung von aufwendigen Darstellungen Informationen jeder Art. Nach ersten Versuchen war schnell klar, dass dies die optimale Lösung für die Visualisierung ist. Da das Processing Konzept für die Arbeit ein wenig zu simpel war, wurde entschieden, einen eigenen Layer zu schreiben, der auf Processing aufsetzt, aber wesentlich mehr Funktionen für die Einbindung des entwickelten Systems bietet.

[2] `http://processing.org/`

Kapitel 11

Weiterentwicklungen

11.1 Mögliche Verbesserungen

11.1.1 Automatische Erweiterung der Typen

Ein erster Schritt zur kontextabhängigen Informationsextraktion, wurde mit unserer Arbeit gemacht. Jedoch bringt gerade die Einschränkung, sich auf vordefinierte Typen zu stützen, einen, in der Realität gewichtigen Nachteil mit sich. Viele Begriffe lassen sich nur schwer den existierenden Definitionen zuordnen. So sind bei jedem Ausbau auf neue Themengebiete, Erweiterungen am Typenkatalog notwendig. In unserer Lösung werden nicht eindeutige Begriffe einem 'Unknown'-Cluster zugeordnet (geschieht nicht mittels Clustering) und werden so, für eine Überarbeitung durch den Benutzer vorgeschlagen. Ein prüfenswerter, interessanter Ansatz wäre es nun, anhand dieser Begriffe eine Modifikation der Granularität des Clustering-Verfahrens durchzuführen. Dies bedingt jedoch eine selbständige Erweiterung des Typenkataloges. Grundsätzlich könnte man auch dieses Problem mittels Clustering der Typenwerte zu lösen versuchen. Als problematisch wird sich jedoch die Suche nach einer geeigneten Wissenressource (wie Wikipedia) erweisen. Erschwerend wirkt ausserdem, dass nicht nur die Typen selbst, sondern auch deren Assoziationen zueinander, gefunden werden müssen. Diese Problemstellungen lassen darauf schliessen, dass dafür in unmittelbarer Zeit, kaum eine Lösung gefunden werden kann.

11.1.2 Direkte Speicherung der Clustering-Werte in Datenbank

In der gewählten Architektur findet sich noch Redundanz bei der Speicherung der Clustering-Ergebnisse. Aufgrund der Anzahl, der beim Clustering einzubeziehenden Tags und deren teilweise geringen kontextfreien Aussagekraft, werden diese in der jetzigen Lösung als XML-Files serialisiert. Problematisch an diesem Ansatz ist, dass diese Dateien ständig verfügbar sein müssen, um deterministische Resultate beim Clustering-Algorithmus zu ermöglichen. Interessant wäre es zu erforschen, inwiefern die Anzahl der Tags, ohne merklichen Qualitätsverlust, reduziert werden kann. Falls sich dies als möglich erweisen sollte, könnten die Clustering-Werte direkt in die Datenbank überführt werden.

11.2 Mögliche Erweiterungen

11.2.1 Extraktion von Properties

Das bestehende System ist darauf ausgelegt, nur die Substantive bzw. Begriffe aus normalsprachlichen Texten zu extrahieren. Wertvoll wäre es nun, auch Zahlen, Synonyme und Einheiten auswerten zu können. Beispielsweise könnte man dem Node "Schweiz" deren Einwohnerzahl (7.7 Millionen) oder deren Fläche (41'285 km^2) zuordnen. Ein möglicher Ansatz besteht darin, "Property"-Nodes in der Datenbank abzuspeichern. Diese Werte würden sich nur durch die Verwendung spezieller Typen, wie "Number", "String" oder "Value", von den anderen Termen unterscheiden. Natürlich müsste auch ein neues Modul zum Aufspüren möglicher Property-Kandidaten, möglicherweise basierend auf den Wikipedia-Tabellen, entwickelt werden. Im Ansatz sind einige, der oben erwähnten Ideen bereits im bestehenden System vorhanden.

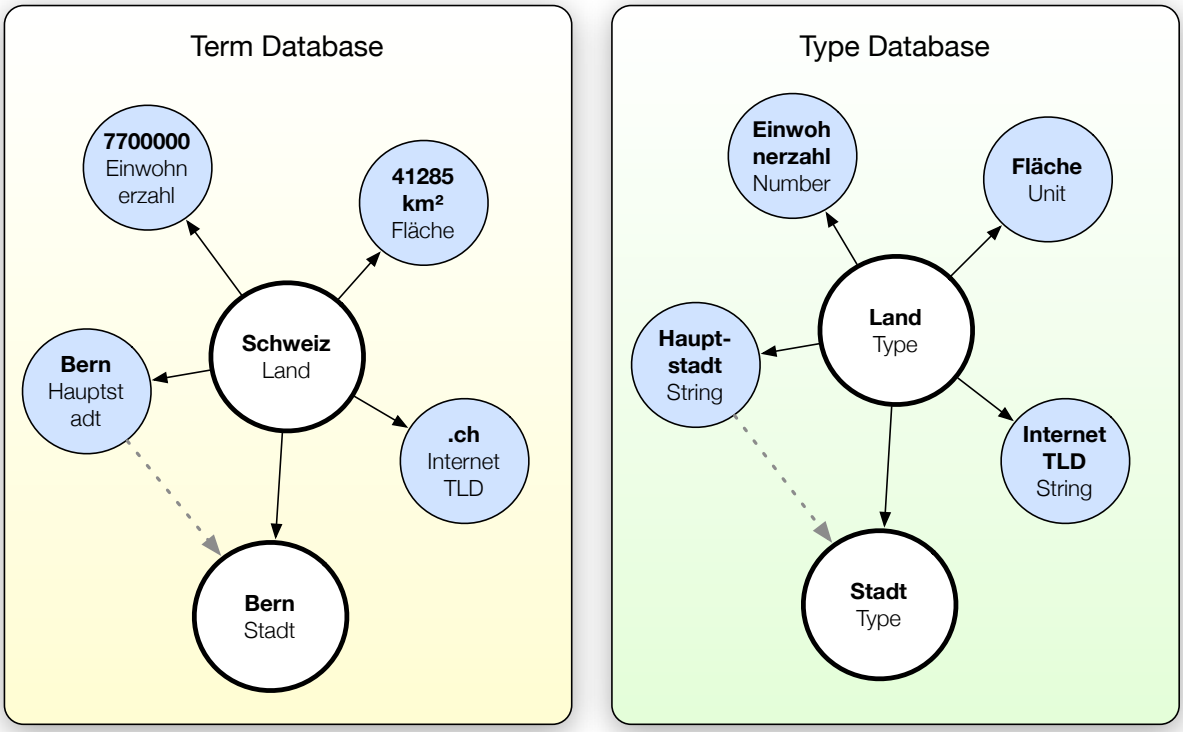


Abbildung 11.1: Erweiterung zur Unterstützung von Property-Nodes auf Datenbankebene

11.2.2 Modellierung von Befehlen

Diese Bachelorarbeit hat sich darauf konzentriert, Substantive und Begriffe gegenseitig in Kontext zu setzen. Besonders interessant wäre es nun, die Abhängigkeiten genauer definieren zu können. Dafür könnte ein Subset, der bekannten Verben in ihrer Infinitivform definiert werden. Beispiele: FLIEGEN -> Abflugort (Flughafen), Ankunftsort (Flughafen), Flugzeug (Flugzeugtyp), Ankunft (Zeit), Abflug

(Zeit). ESSEN -> Gericht (Lebensmittel), Zeit (Zeit), Subjekt (Person), Kalorien (Zahl+Einheit). So könnte man ein Stock von Tätigkeiten modellieren. Falls nun ein assoziativer Term in einem Satz nicht vorkommt, könnte der Computer Vorschläge für diesen bringen.

11.2.3 Mehrfachbeziehungen

Wie in Kapitel 9.2 bereits beschrieben, ist es nicht möglich, Mehrfachbeziehungen zwischen Elementen abzubilden. In einem weiteren Schritt wäre es jetzt spannend, diese Mehrfachbeziehungen, die in der Realität sehr oft vorkommen, im System abzubilden. Die grösste Hürde hierbei, wäre die Bestimmung des Grenzwertes, der gefunden werden muss, ab dem ein Begriff zu einem anderen in Verbindung steht. Eine starre Modellierung, die vorgibt, wieviele andere Elemente ein bestimmtes Element besitzt, kann nicht die Lösung sein, da die Wartung eines solchen Systems viel zu aufwendig würde.

11.2.4 Datenhaltung mit RDF

Das Resource Description Framework[1] (RDF) bietet ein einfaches Datenmodell, mit einer wohldefinierten, formalen Semantik (RDF-Modell), welches auf gerichteten Graphen basiert. RDF ist ein zentraler Bestandteil eines Semantischen Netzes und bietet den Vorteil, dass es durch die formale Repräsentation, von Maschinen auswert- und dadurch maschinell durchsuchbar ist. Die jetzige Datenrepräsentation im System könnte in RDF Datensätze konvertiert und so einem grösseren Publikum zugänglich gemacht werden.

Diese Erweiterung sollte mit relativ wenig Mehraufwand verbunden sein. Leider ist zum jetzigen Zeitpunkt die Dokumentation seitens Neo4j diesbezüglich noch etwas spärlich. Grundsätzlich benötigt man für den RDF-Zugriff einen RDF-Layer und optional auch einen zusätzlichen Sesame/OpenRDF[2] Sail (Storage and Inference Layer). Die entsprechenden Components für Neo4j findet man im SVN-Repository[3]. Nach dem Import dieser Komponenten, würde die Klasse DatabaseService um folgende Zeilen erweitert werden müssen:

Code Listing 11.1 DatabaseServiceRDF.java

```
1  public class DatabaseServiceRDF implements IDatabaseService {
2      private NeoService neoService;
3      private IndexService indexService;
4      private RdfStore rdfStore;
5      private Sail rdfSail;
6
7      public DatabaseService( String databasePath ) {
8          this.databasePath = databasePath;
9          neoService = new EmbeddedNeo( databasePath );
10         indexService = new NeoIndexService( neoService );
11         rdfStore = new VerboseQuadStore( neoService, indexService );
12         rdfSail = new NeoSail( neoService, rdfStore );
13     }
14
15     // [ irrelevant code ommitted ]
16 }
```

[1] http://www.w3.org/RDF/
[2] http://www.openrdf.org/
[3] svn://svn.neo4j.org/components

Selbstverständlich muss auch der ShutdownHook entsprechend angepasst werden, um sicherzustellen, dass die neuen Komponenten wieder in der korrekten Reihenfolge heruntergefahren werden.

Code Listing 11.2 DatabaseServiceRDF.java

```
1  public class DatabaseServiceRDF implements IDatabaseService {
2      public void shutdown() {
3          rdfSail.shutdown();
4          rdfStore.shutdown();
5          indexService.shutdown();
6          neoService.shutdown();
7      }
8
9      // [ irrelevant code ommitted ]
10 }
```

11.2.5 Mehrere Analyse Clients

Das System ist im Moment so aufgebaut, dass zwar die Beschaffung der benötigten Informationen zu einem bestimmten Begriff von beliebig vielen Agenten, auf verteilten Rechnern übernommen wird, die Analyse dieser Informationen jedoch, auf einer zentralen Maschine geschieht. Dies wurde so gewählt, weil die Datenbank Neo4j im Moment noch keine zeitgleichen Zugriffe aus verschiedenen Virtual Machines zulässt. In Zukunft wäre es wünschenswert, dass die Analyse ebenfalls auf beliebig viele Rechner ausgelagert werden kann, damit möglichst viele Zugriffe, quasi zeitgleich abgearbeitet werden können.

Teil III

Software-Projektdokumentation

Kapitel 12

Anforderungsspezifikation

12.1 Funktionale Anforderungen

12.1.1 Funktionen des Wissensextraktions-Prototypen

Die Wissensextraktion, samt Clustering, stellt den grössten Teil der Anforderungen dar. Die Applikation soll als Datenquelle, die freie Online-Enzyklopädie Wikipedia verwenden. Um auf die publizierten Informationen zugreifen zu können, muss entweder ein bestehender oder ein speziell entwickelter Crawler eingesetzt werden. Dabei sollen für entsprechende Begriffe, auch die jeweils relevanten Artikel gecrawlt werden können. Speziell die Multivalenz verschiedener Begriffe, wie beispielsweise "Zürich" (Kanton und Stadt), gilt es zu beachten. Da anzunehmen ist, dass in Zukunft eine sehr grosse Anzahl von Wikipedia-Artikeln ausgelesen werden muss, empfiehlt es sich, eine verteilte Architektur einzusetzen, welche paralleles Crawling mehrerer Seiten unterstützt.

Sind die Wikipedia-Artikel ausgelesen, beginnt die Aufgabe, aus diesen entsprechende, möglichst vielsagende Tags zu extrahieren. Wiederum empfiehlt es sich, dies in einem verteilten Kontext durchzuführen, da sich, wie auch schon beim Crawling, durch paralleles Abarbeiten deutliche Performanzgewinne realisieren lassen.

Durch Clusteringverfahren soll dann überprüft werden, ob sich die gecrawlten Begriffe auf einen der vorgegebenen Typen zuordnen lassen. Dabei muss für deterministisches Clustering stets auf die gleichen, extrahierten Tags zugegriffen werden können.

Falls eine verteilte Architektur gewählt wird, soll bei genügend Zeit, auch eine einfache Möglichkeit für das Remote-Management der verschiedenen Clients gefunden werden.

Nummer	Funktion	Priorität
1	Auslesen von Wikipedia Html-Seiten für gegebene Begriffe.	hoch
2	Verteiltes Crawling der Html-Seiten für bessere Performance.	mittel
3	Herausfiltern von nicht brauchbaren, häufig vorkommenden Wörtern.	hoch
4	Extraktion und Bewertung von Tags aus Plaintext-Content.	hoch
5	Verteilte Tag-Extraktion aus Plaintext-Content.	mittel
6	Konfiguration von Typklassen für das Clustering.	hoch
7	Konfiguration vorgegebener, korrekter Begriffe fürs Clustering.	hoch
8	Clusteringverfahren zur Zuordnung von unbekannten Begriffen.	hoch
9	Speicherung der extrahierten Tags und der Clustering Werte.	hoch
10	Remote-Management der verteilten Agenten.	niedrig

Tabelle 12.1: Funktionale Anforderungen an die Wissensextraktion und deren Prioritäten

12.1.2 Funktionen des Datenhaltungs-Prototypen

Um die extrahierten Informationen auch weiterverwenden zu können, wird eine Datenhaltung benötigt. Dabei sollen alle Begriffe, inklusive deren Attribute und dem zugeordneten Clusterbegriff, gespeichert werden. Jeder Begriff hat auch mehrere verwandte Begriffe, die ebenfalls von der Datenhaltung abgebildet werden sollen. Auch muss der Katalog der vordefinierten Typen und deren Relationen zueinander in irgendeiner Form persistiert werden.

Es ist anzunehmen, dass das Auslesen der Datensätze mit Abstand die häufigste Zugriffsoperation sein wird. Entsprechend soll auch eine Suche nach verschiedenen Begriffen, anhand deren Typen unterstützt werden.

Da das Ressource Description Framework (RDF) als Standard für Ontologie-Netze gilt, soll es mit Vorteil möglich sein, über eine RDF/XML-Schnittstelle auf die Datenhaltung zugreifen zu können.

Nummer	Funktion	Priorität
1	Speicherung von Begriffen inklusive deren Attribute.	hoch
2	Speicherung der Relationen zwischen den Begriffen.	hoch
3	Speicherung des Typenkatalogs und dessen Relationen.	hoch
4	Suchfunktionalität für Begriffe und Typen.	hoch
5	Korrigieren von Datenbank-Einträgen (Begriffe bestätigen).	mittel
6	Zugriffslayer RDF.	niedrig

Tabelle 12.2: Funktionale Anforderungen an die Datenhaltung und deren Prioritäten

12.1.3 Funktionen des Visualisierungs-Prototypen

Um die erzielten Ergebnisse für den Menschen fassbar zu machen, soll eine Visualisierung der Datenhaltung entwickelt werden. Dabei sollen primär die verwandten Typen zur Geltung gebracht werden. Ausserdem soll erkennbar sein, von welchem Typ ein jeder Term ist. Die Navigation durch die gespeicherten Terme kann beispielsweise durch sequentielle Klicks erfolgen.

Durch eine Suchmaske soll der Benutzer nach einem gewünschten Begriff suchen können. Gerade vom Benutzer eingegebene, aber noch nicht im System erfasste Begriffe, könnten direkt den Crawling- / Clustering-Prozess starten. Zu jedem angezeigten Begriff soll zudem die Quelle und ein einfacher Text wiedergegeben werden.

Nummer	Funktion	Priorität
1	Visualisierung der Datenbank-Einträge.	hoch
2	Suchen eines Begriffes oder Typs.	hoch
3	Navigation durch die gespeicherten Begriffe.	hoch
4	Analyse eines neuen, ungespeicherten Begriffs.	mittel
5	Anzeige weiterer Information zum gesuchten Begriff.	niedrig

Tabelle 12.3: Funktionale Anforderungen an die Datenhaltung und deren Prioritäten.

12.2 Use Cases

12.2.1 Use Case Diagramm

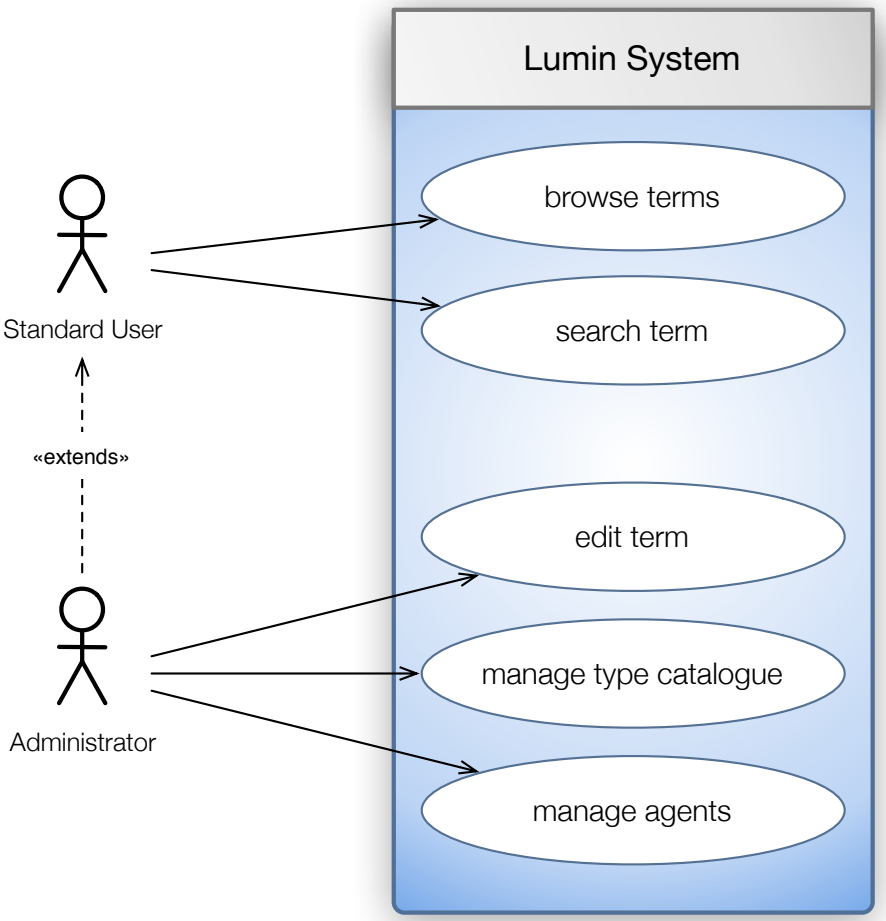


Abbildung 12.1: Use-Case-Diagramm des Lumin Systems

12.2.2 UC01: Search Term

Beschreibung

Ein Benutzer möchte nach einem Begriff suchen, um herauszufinden, was der Begriff für einen Type hat und wie er in Relation zu anderen Begriffen steht.

Vorbedingungen

Zum Begriff, nach dem der Benutzer suchen möchte, besteht ein Eintrag in Wikipedia.

Nachbedingungen

Der Type des gesuchten Begriffs wird dem Benutzer angezeigt und wenn vorhanden, auch die Relation zu anderen Begriffen. Wenn der Begriff noch nicht vorhanden war, ist er nun im System gespeichert.

Ablaufschritte

1. Der Benutzer gibt den zu suchenden Begriff ein und startet den Suchvorgang.
2. Das System überprüft die Datenbank, ob der gesuchte Begriff bereits vorhanden ist.
3. Das System lädt den gefundenen Begriff mitsamt seinen Relationen aus der Datenbank.
4. Der Begriff wird dem Benutzer, mit allen seinen verknüpften Begriffen angezeigt.

Alternative Ablaufschritte

3.a Das System findet den Begriff nicht in der Datenbank und leitet die Suchanfrage an einen Agent weiter.

3.b Das System analysiert den Begriff und gibt den gefundenen Type zurück.

3.c Das System speichert den gefundenen Begriff ab.

4.a Dem Benutzer wird angezeigt, um was für einen Type es sich beim gesuchten Begriff handelt.

12.2.3 UC02: Browse Terms

Beschreibung

Der Benutzer möchte, von einem gefundenen Begriff aus, andere, mit diesem in Beziehung stehende, Begriffe durchsuchen und deren Relationen anzeigen lassen.

Vorbedingungen

Der Begriff, von dem aus die Navigation beginnen soll, wird dem Benutzer angezeigt und besitzt Relationen, die durchsucht werden können.

Nachbedingungen

Der Begriff, bei dem das Durchsuchen endet, wird dem Benutzer angezeigt.

Ablaufschritte

1. Der Benutzer klickt auf einen, mit dem Startbegriff verwandten Begriff.
2. Das System lädt die Informationen zum angeklickten Begriff und zeigt diesen, mitsamt seinen verknüpften Begriffen an.
3. Der Benutzer wiederholt Schritt 1 so oft er will.

Alternative Ablaufschritte

1.b Der angeklickte Begriff hat keine verwandten Begriffe und kann nicht mehr weiter durchsucht werden.

12.2.4 UC03: Edit Term

Beschreibung

Der Administrator möchte den Type eines Begriffs, den er als falsch erkannt hat, korrigieren.

Vorbedingungen

Der Benutzer, der die Korrektur vornehmen will, muss Administrator Rechte besitzen. Der Begriff, der korrigiert werden soll, ist im System erfasst.

Nachbedingungen

Der korrigierte Begriff ist korrekt abgespeichert und wird als bestätigt angezeigt.

Ablaufschritte

1. Der Administrator wählt den zu korrigierenden Begriff aus und ändert den Type.
2. Das System speichert den korrigierten Begriff sofort ab und setzt den Status auf bestätigt.

Alternative Ablaufschritte

2.c Das System kann den Begriff nicht abspeichern und zeigt dem Administrator eine Fehlermeldung an.

12.2.5 UC04: Manage Type Catalogue

Beschreibung

Der Administrator möchte einen neuen Type definieren.

Vorbedingungen

Der Benutzer, der diese Aktion durchführen will, braucht Administrator Rechte. Der neu zu definierende Type ist noch nicht vorhanden. Der Administrator hat eine Liste mit Begriffen erstellt, die zu diesem neuen Type gehören.

Nachbedingungen

Der neu definierte Type ist in der Datenbank vorhanden. Die eingelesenen Begriffe dazu, sind im System gespeichert.

Ablaufschritte

1. Der Administrator macht einen Eintrag für den neuen Type im dafür vorgesehenen XML-Schema.
2. Der Administrator startet die Erstellung der Type-Datenbank.
3. Das System baut die Type-Datenbank neu auf und erstellt alle nötigen Einträge.
4. Der Administrator startet das Lernprogramm.
5. Das Lernprogramm analysiert alle vordefinierten Begriffe, die zum neuen Type gehören und weist sie diesem Type zu.

Alternative Ablaufschritte

5.a Das System kann einen neuen Begriff nicht analysieren oder abspeichern und zeigt dem Benutzer eine Fehlermeldung an.

12.2.6 UC05: Manage Agents

Beschreibung

Der Administrator kann beliebig viele Agents, die sich auf verteilten Rechnern befinden, je nach Bedarf starten und stoppen.

Vorbedingungen

Der Benutzer, der diese Aktion durchführen will, braucht Administrator Rechte. Die verteilten Agents befinden sich auf den jeweiligen Rechnern in einem lauffähigen Zustand.

Nachbedingungen

Die angesteuerten Agents befinden sich in dem Zustand, die der Administrator für sie vorgesehen hat.

Ablaufschritte

1. Dem Administrator werden alle steuerbaren Agents angezeigt.
2. Der Administrator wählt einen Agent aus und setzt den gewünschten Status.
3. Das System steuert den gewählten Agent an und startet oder stoppt ihn.

Alternative Ablaufschritte

3.a Das System kann den gewählten Agent nicht ansteuern und gibt dem Benutzer eine Fehlermeldung aus.

12.3 System-Sequenzdiagramm

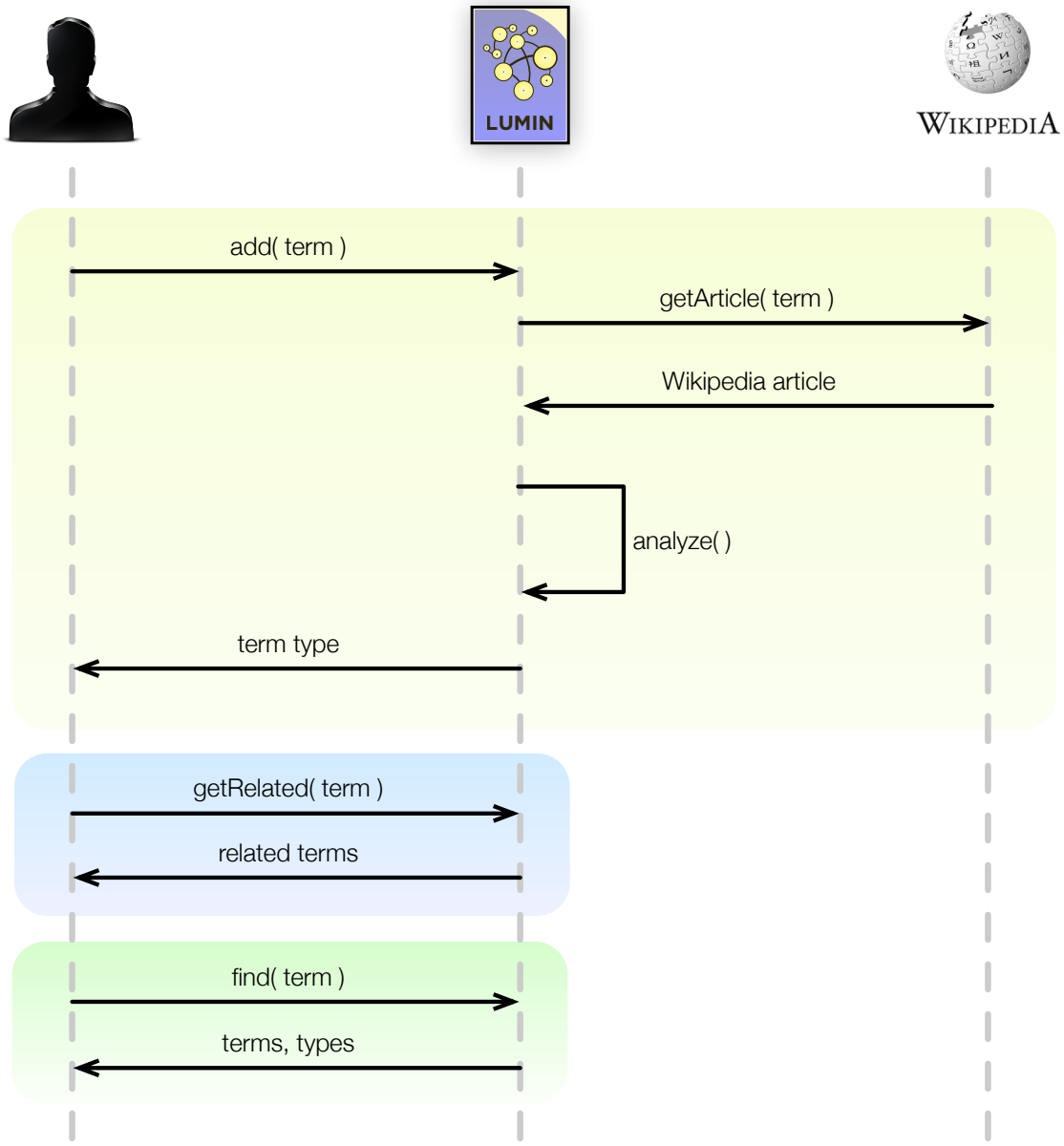


Abbildung 12.2: System-Sequenzdiagramm

12.4 Nicht-funktionale Anforderungen

In den folgenden Abschnitten sind die nicht-funktionalen Anforderungen der Softwarelösung aufgegliedert.

12.4.1 Zuverlässigkeit

Gespeicherte Daten und besonders bereits analysierte Web-Artikel, sollen möglichst wiederhergestellt werden können. Die Systemreife der Software entspricht der allgemein zu erwarteten Reife eines wissenschaftlichen Prototypen. Entsprechende Fehler sollen weitgehend reproduzierbar sein. Die Softwarelösung stellt darüber hinaus keine besonderen Anforderungen an die Zuverlässigkeit der Applikation.

12.4.2 Benutzbarkeit

Die Verständlichkeit und Erlernbarkeit der Applikation ist so zu realisieren, dass ein gebildeter, bereits mit der Materie vertrauter, Wissenschaftler innerhalb einer Woche produktiv damit umgehen kann. Die Bedienbarkeit soll keine, über den normalen Computer-Gebrauch herausgehende, Anforderungen an den Benutzer der Applikation stellen. Die Benutzung des Visualisierungs-Clients soll weitgehend autonom erlernbar sein.

12.4.3 Leistung

Während Angaben zum benötigten Ressourcenbedarf schwer einschätzbar sind, soll zumindest eine Version mit wenigen initialen Datensätzen auf einem handelsüblichen Desktop-Rechner (2 GHz CPU und 2GB RAM) gut lauffähig sein. Bei der Visualisierung gilt eine Framerate über 15 Frames pro Sekunde (fps) als akzeptabel.

12.4.4 Betriebs- und Umgebungsbedingungen

Es wird vorausgesetzt, dass die Softwarelösung unter den aktuellen Betriebssystemen Mac OS X 10.5, Windows XP, Windows Vista und Ubuntu 9.04 voll lauffähig sein soll. Unter sämtlichen Betriebssystemen sollen die unter "Leistung" erwähnten Kriterien erfüllt werden können.

12.4.5 Wartbarkeit und Änderbarkeit

Für eine eventuelle Weiterbearbeitung der Bachelorarbeit oder für mögliche Folgearbeiten, ist es nötig, den bestehenden Code möglichst änderbar zu halten. Dies kann von Vorteil mit einer guten Code-Dokumentation gewährleistet werden. Es werden jedoch keine besonderen Anforderungen an die Wartbarkeit der Softwarelösung gestellt.

12.4.6 Sicherheitsanforderungen

Durch die freie Verfügbarkeit der Informationen auf Wikipedia, ist davon auszugehen, dass keine besonderen Sicherheitsmassnahmen für diese Daten benötigt werden.

12.4.7 Korrektheit

Durch den Prototypenstatus der Lösung können keine definitiven Garantien für die Korrektheit der Daten gegeben werden. Ein gutes Ergebnis für die Begriffszuordnung wird aber bei einer Trefferquote über 75% festgelegt.

12.4.8 Flexibilität

Wo immer möglich, sollen gängige Standards unterstützt werden. Dies zielt vor allem auf die Datenhaltung ab, die in Zukunft auch möglichst mit dem RDF-Standard nachgerüstet werden soll.

Kapitel 13

Analyse

13.1 Domain Model

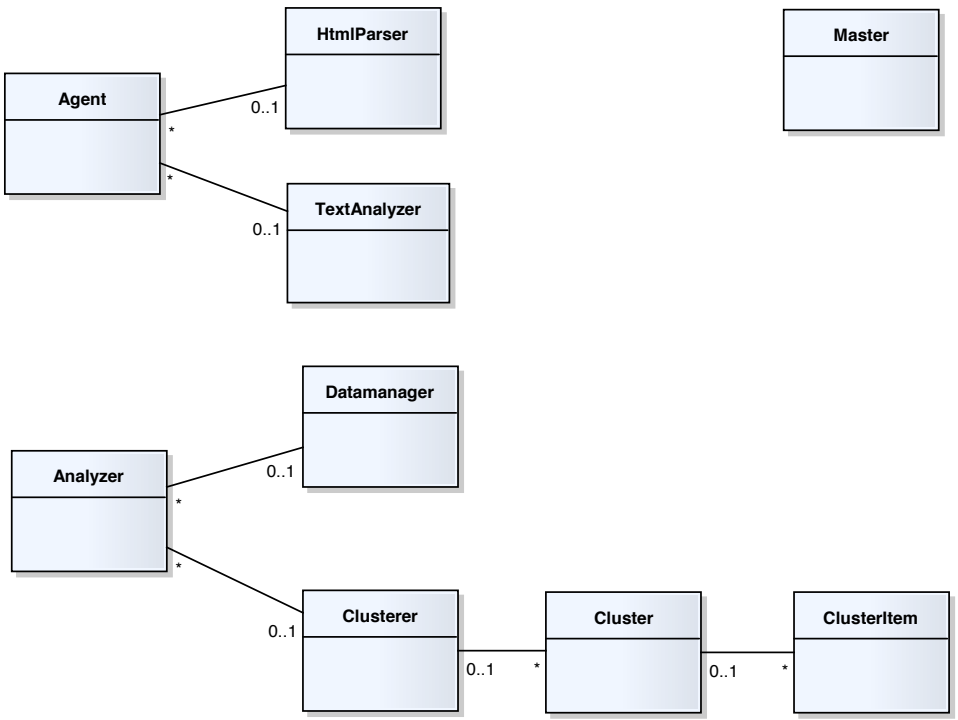


Abbildung 13.1: Domain Model des verteilten Systems

Dieses Domain Model stellt eine Übersicht, über die wesentlichen Objekte dar. Dabei sind die Hauptbestandteile Agent, Master und Analyzer mit ihren Beziehungen abgebildet.

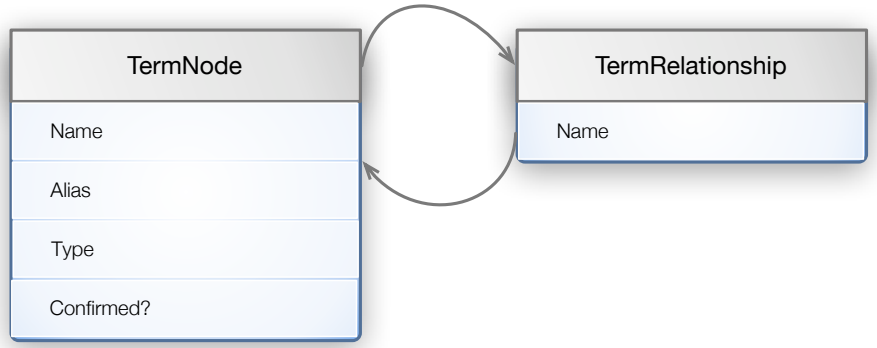


Abbildung 13.2: Domain Model der Datenhaltung

Diese Abbildung zeigt ein Model der Datenhaltung. Jede TermNode kann über eine TermRelationship mit einer beliebigen TermNode verknüpft werden.

Kapitel 14

Design

14.1 Architektur

Die Anforderungsspezifikation erläutert verschiedene Gründe, wieso die Architektur der Informationsbeschaffung bevorzugt verteilt zu lösen ist. Genau dies wurde bei unserem Architekturentwurf berücksichtigt. Im Kapitel Umsetzungskonzepte (siehe Kapitel 7.4), finden sich verschiedene Gründe für den Einsatz von JMS.

Der Wunsch nach dem parallelen Beschaffen von Inhalten und der parallelen Extraktion von Tags aus diesem Inhalt, lassen sich weitgehend kombinieren. Unsere Lösung verwendet eine erweiterbare Agentenarchitektur. Für jede Wissensquelle soll ein eigener, spezifischer Agenten-Typ entwickelt werden, welcher selbständig die benötigten Informationen beschafft. Obwohl die Bachelorarbeit auf eine einzige Quelle eingeschränkt wurde, ermöglicht diese Architektur im Prinzip die Beschaffung von Daten von mehreren Webseiten gleichzeitig. Danach müssen die Resultate noch verglichen und selektiert werden. Die verschiedenen Agenten sind eigenständige Programme und können entsprechend auf verschiedenen Systemen eingesetzt und trotz den verschiedenen Java Virtual Machines mittels Java Messaging System mit den anderen Komponenten im Netzwerk kommunizieren.

Der Master ist dafür verantwortlich, die zu analysierenden Terme an die diversen Agenten zu verteilen. Diese wiederum, melden deren Resultate an den Analyzer, welcher ebenfalls auf einem anderen System, als der Master im Einsatz sein kann. Der Analyzer ist für die Zuordnung der Begriffe zu den Typen zuständig.

Um die Datenbank möglichst zu entkoppeln und dadurch austauschbar zu machen, wurde das Data Access Object (DAO) Pattern implementiert. Damit kann die zugrunde liegende Datenbank, mit minimalen Änderungen am Code gewechselt werden. Bei der Implementierung des NodeHandlerServices, wurde das Facade Pattern verwendet. Dadurch wird eine Schnittstelle zu den Service Methoden realisiert, was die Verwendung des Services vereinfacht und zu loser Kopplung führt. Die GUI Architektur ist an Swing angelehnt, da sich die auf dem GUI befindlichen Komponenten selber zeichnen und Events entgegennehmen können. Der Database-Layer verwendet das Delegation Pattern. Um beispielsweise eine TermNode einzufügen, gibt der Database-Layer die TermNode an den Neo-Service weiter, der die entsprechende Behandlung übernimmt. Nach aussen wird jedoch der Eindruck erweckt, dass der Database-Layer diese Aufgabe übernimmt. Bei der Kommunikation des HostMasters mit dem GUI, wird das Observer Pattern eingesetzt.

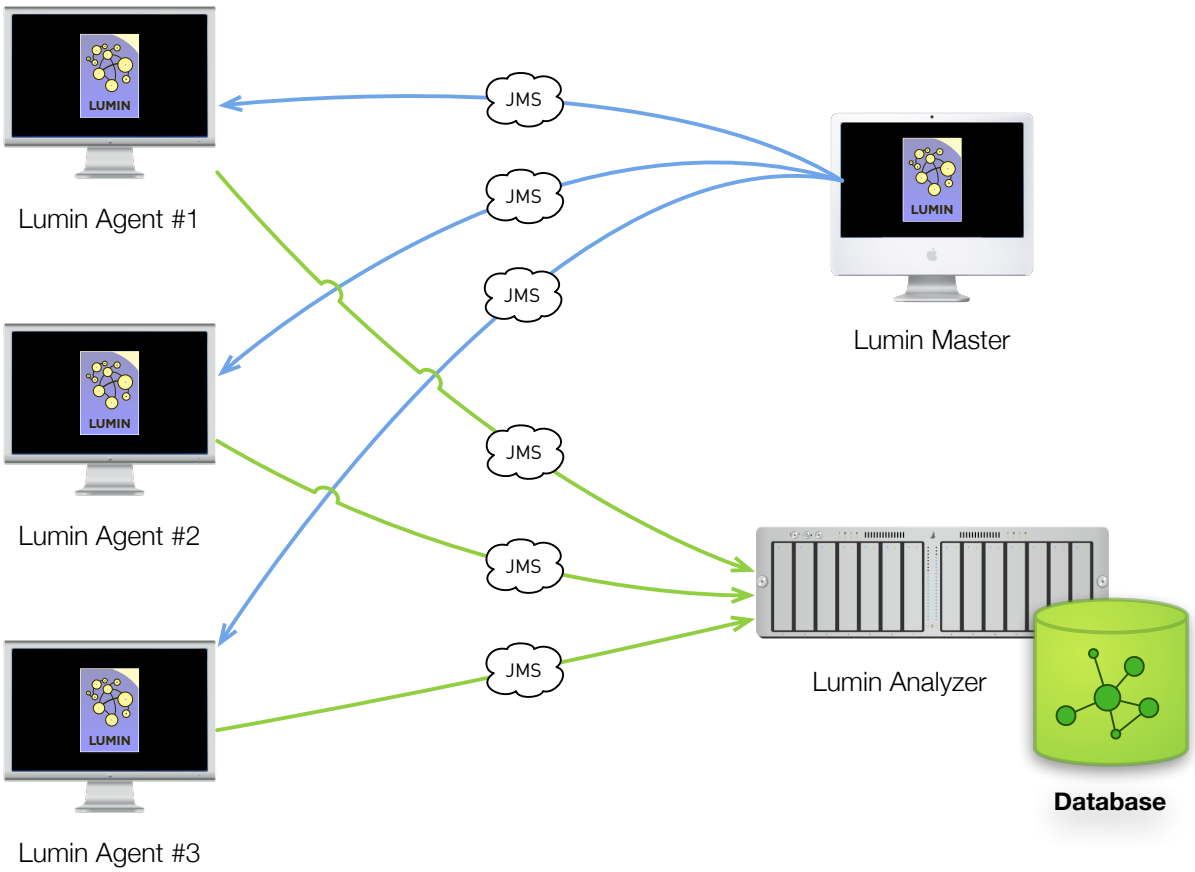


Abbildung 14.1: Übersicht über die verteilte Architektur

14.2 Package- und Klassendiagramme

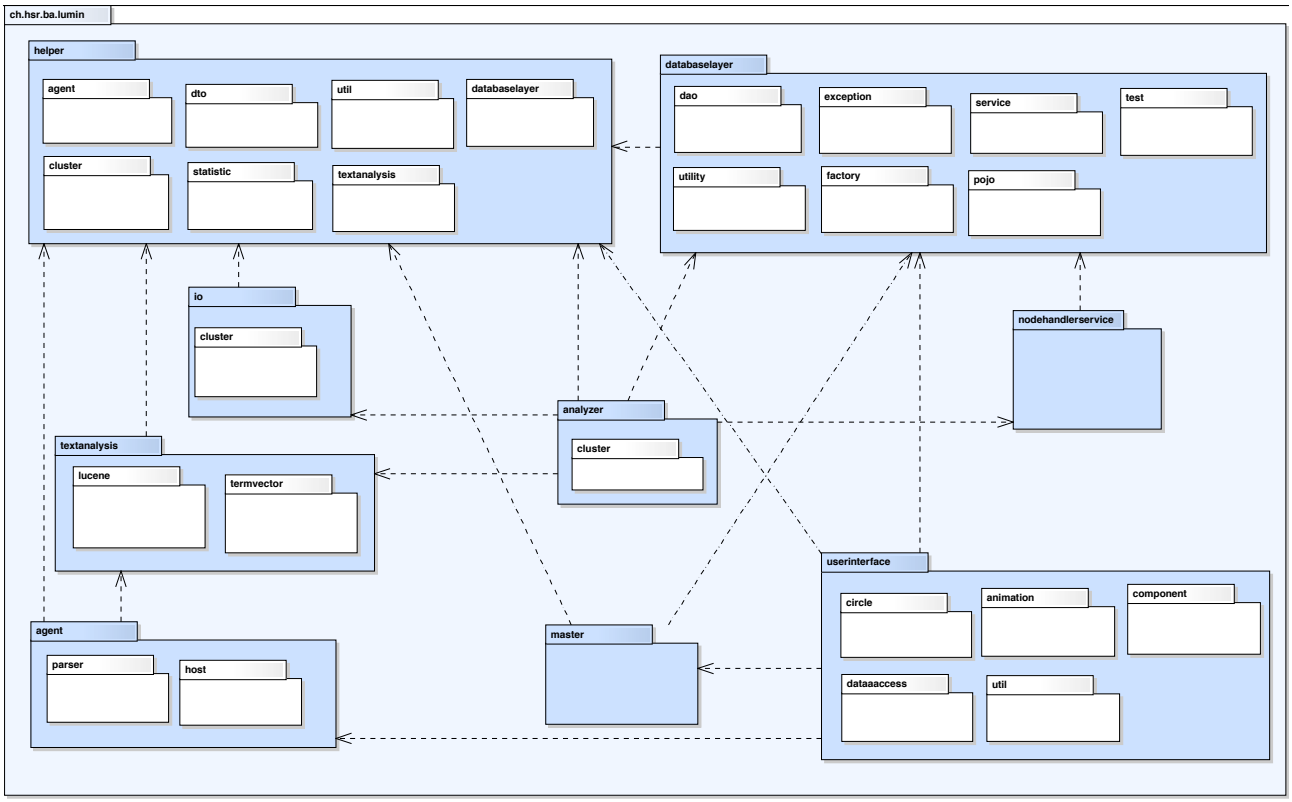


Abbildung 14.2: Package-Diagramm

Das Package-Diagramm zeigt eine Übersicht der Abhängigkeiten der einzelnen Packages. Wesentlich ist, dass die Packages Agent, Master und Analyzer nicht direkt voneinander abhängig sind.

14.2.1 Agent Package

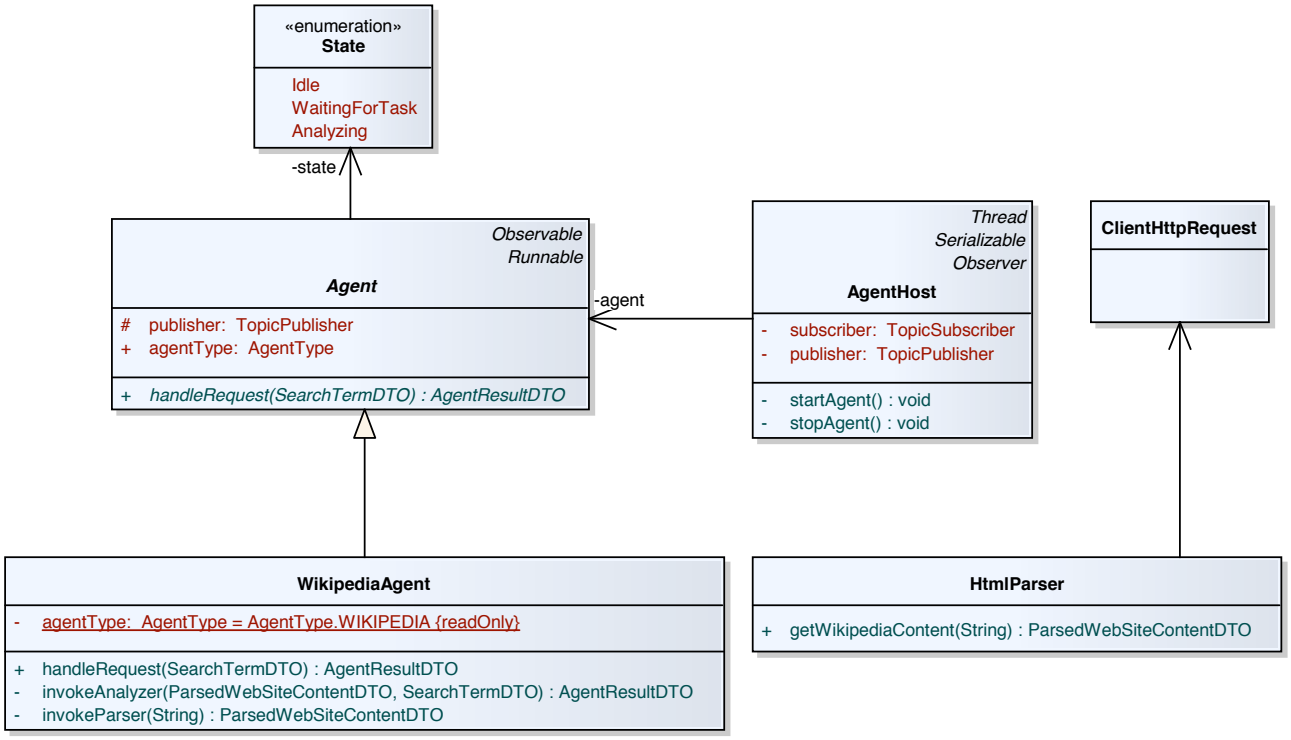


Abbildung 14.3: Klassendiagramm Agent Package

Im Agent Package sind alle, für den verteilten Agent relevanten Klassen, zusammengefasst. Die abstrakte Klasse Agent bildet die nötige JMS-Funktionalität (Empfang von Aufträgen des Masters und Weitergabe der Resultate zur Analyse) ab. Die Spezialisierung, die Klasse WikipediaAgent, definiert in der Methode handleRequest wie eingehende Aufträge abgearbeitet werden. Die Hilfsklasse HtmlParser, beschreibt das Parsen der entsprechenden Website, die als Datenquelle dient.

Die Klasse AgentHost bildet den Host für beliebige Agenten. Damit lassen sich die verteilten Agenten steuern.

14.2.2 Analyzer Package

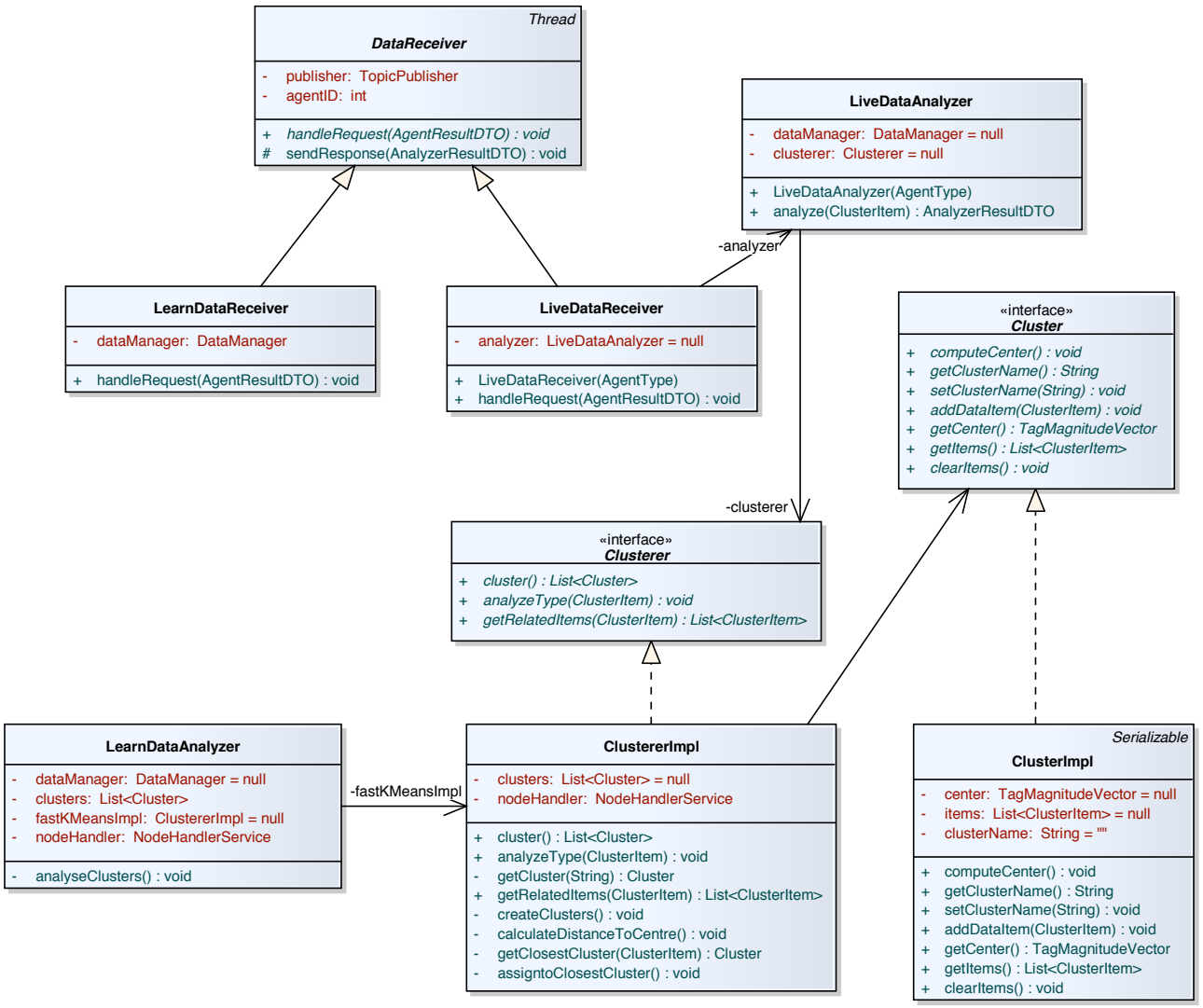


Abbildung 14.4: Klassendiagramm Analyzer Package

Das Analyzer Package umfasst die Kernklassen, die für die Analyse der Agent-Resultate benötigt werden. Dabei definiert die abstrakte Klasse DataReceiver die JMS-Funktionalität (Empfang der Agent-Resultate). Es existieren zwei Spezialisierungen. Der LearnDataReceiver ist für viele Aufträge in kurzer Zeit (Lerndaten) konzipiert. Daher nimmt er nur Aufträge entgegen und speichert diese für die spätere Analyse. Der LiveDataReceiver nimmt Aufträge entgegen und gibt diese zur Realtime - Verarbeitung an den LiveDataAnalyzer weiter.

Die Klasse ClustererImpl implementiert das Clusteringverfahren, nach dem die zu analysierenden Agent-Resultate gruppiert werden.

14.2.3 Databaselayer Package

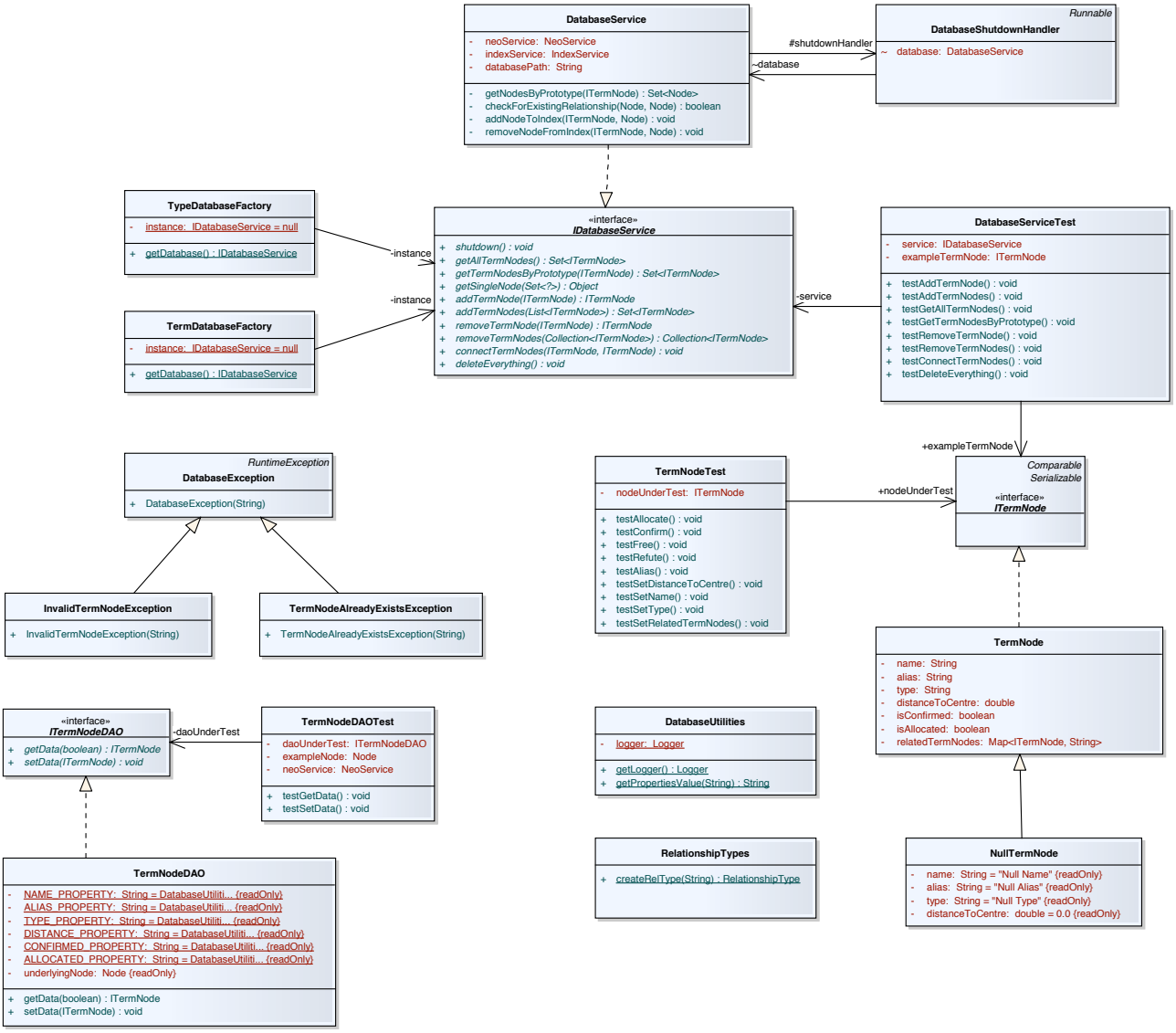


Abbildung 14.5: Klassendiagramm Databaselayer Package

Im Databaselayer Package befinden sich die Klassen, die für den Datenbankzugriff nötig sind. Die Instanz der Datenbank wird in den beiden Factorys TypeDatabaseFactory und TermDatabaseFactory erstellt. Je nachdem, ob man auf die Type- oder Term-Database zugreifen möchte.

Ebenfalls in diesem Package befindet sich das Interface und die Implementierung einer TermNode. Diese TermNode ist das eigentliche Objekt, das in der Datenbank gespeichert wird. Zudem werden eigene Exceptions definiert, um allfällige Fehler besser behandeln zu können.

14.2.4 Helper Package

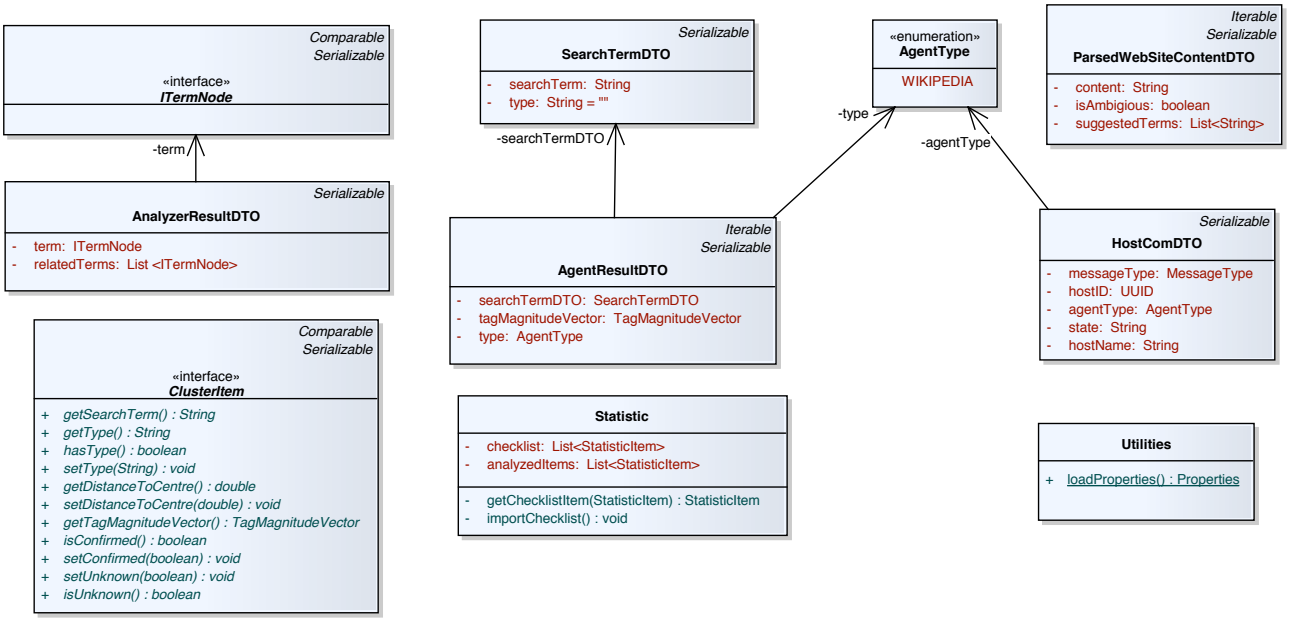


Abbildung 14.6: Klassendiagramm Helper Package

Das Helper Package ist das universal Package, das Hilfsklassen für alle anderen Packages zur Verfügung stellt. Hier befinden sich auch alle vom System genutzten Data Transfer Objects (DTO). Die beiden Interfaces für das ClusterItem, sowie die TermNodes sind ebenfalls in diesem Package vorhanden, da sie in verschiedenen Packages implementiert werden müssen. Die Klasse Utilities wird zum Laden der Property-Files genutzt, sowie für allfällige weitere Aufgaben, die überall eingesetzt werden können. Die Klasse Statistic wird gebraucht, um die im System vorhandenen Terms auszuwerten und anzuzeigen, wieviele richtig zugeordnet wurden.

14.2.5 IO Package

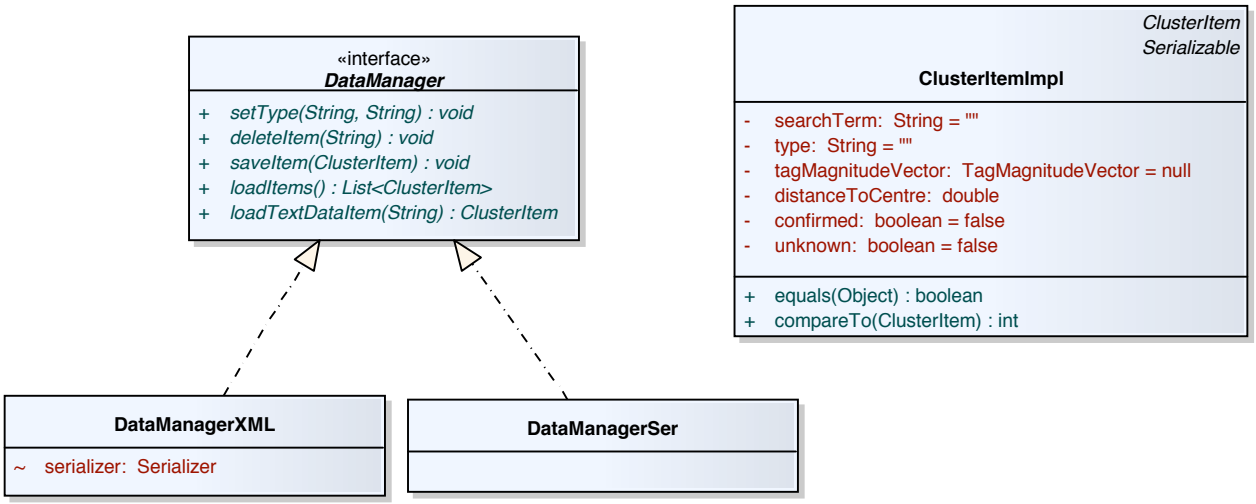


Abbildung 14.7: Klassendiagramm IO Package

Das Input / Output Package ist für das Laden und die Speicherung der ClusterItemImpl-Objekte verantwortlich. Dabei kann zwischen XML-Daten (DataManagerXML) und serialisierten Objekten (DateManagerSer) gewählt werden. Das CluserItemImpl Objekt repräsentiert die, für die Analyse eines Begriffes nötigen Daten.

14.2.6 Maintenance Package

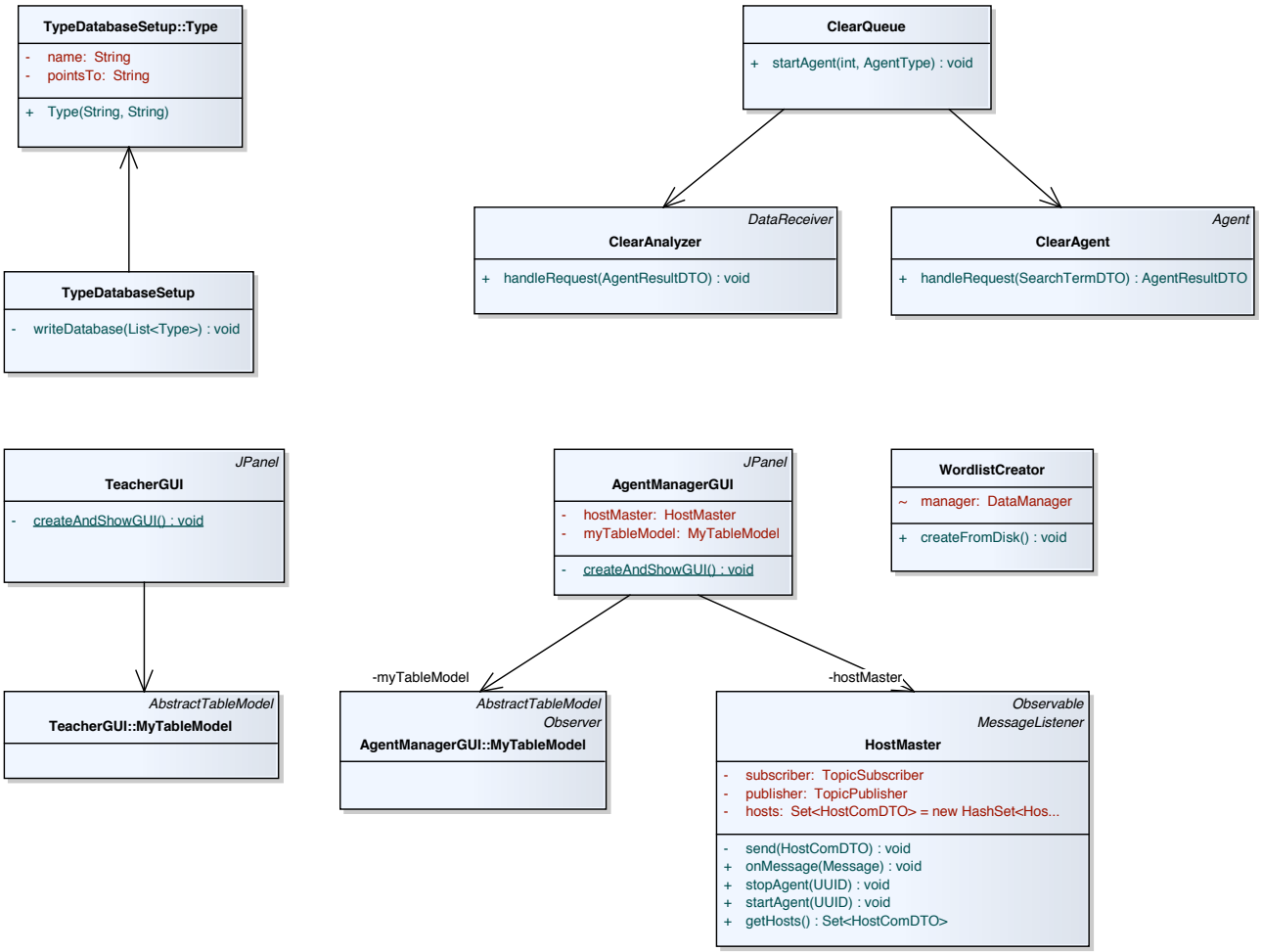


Abbildung 14.8: Klassendiagramm Maintenance Package

Das Maintenance Package ist die "Schaltzentrale" für das System. Mit der Klasse TypeDatabaseSetup kann die Type-Database um neue Types erweitert und wieder neu initialisiert werden. Die Klasse ClearQueue wird gebraucht, um die sich in einer JMS-MessageQueue befindenden Nachrichten zu löschen. Dies wird vor allem dann benötigt, wenn das System abgestürzt ist oder man Tests durchführt, die immer eine leere MessageQueue voraussetzen. Das AgentManagerGUI wird genutzt um die, verteilten Agents zu kontrollieren. Die Agents können über dieses Tool je nach Bedarf gestartet oder gestoppt werden. Der WordlistCreator kann aus allen, in der Datenbank vorhandenen Termen eine Textdatei mit allen Informationen erstellen. Diese Datei kann unter anderem genutzt werden, um die Daten mit einem LearnMaster (siehe Kapitel 14.2.7) in ein frisches System einzulesen.

14.2.7 Master Package

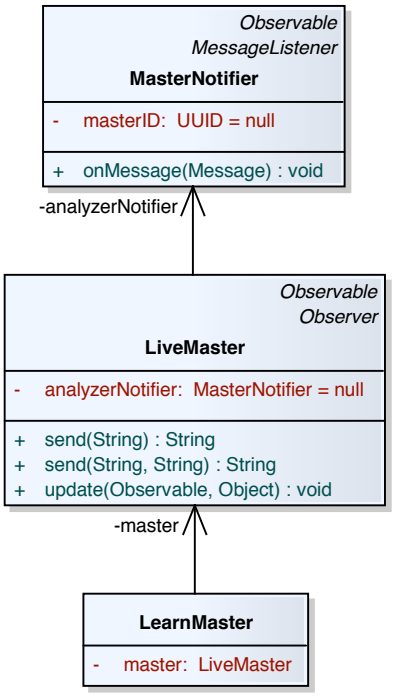


Abbildung 14.9: Klassendiagramm Master Package

Ein Master wird genutzt, um eine Abfrage ans System zu starten. Der LearnMaster kann ganze Textdateien mit Termen durcharbeiten und einen Term nach dem anderen abschicken, um diesen zu analysieren. Er wird ebenfalls gebraucht, um neue Types zu lernen. Dazu kann man ihm angeben, von welchem Type gewisse Terms sind. Diese Terme werden danach mit den Type Informationen im System abgelegt. Der LiveMaster kommt dann zum Einsatz, wenn zu einem Term unmittelbar der dazugehörige Type bestimmt werden soll. Der Term wird im System abgelegt und der bestimmte Type zurückgegeben. Der MasterNotifier wird für die Kommunikation mit dem Analyzer (siehe Kapitel 14.2.2) gebraucht. Er ist von Observable abgeleitet und implementiert das MessageListenerInterface. Damit kann er von JMS angesprochen werden.

14.2.8 NodeHandlerService Package

Abbildung 14.10: Klassendiagramm NodeHandlerService Package

Der NodeHandler setzt auf dem Databaselayer auf und wird benötigt, um Daten aus der Datenbank zu laden, zu löschen, zu ändern oder um neue Nodes einzufügen. Er bietet sowohl Zugriff auf die Term- wie auch die Type-Datenbank.

14.2.9 Textanalysis Package

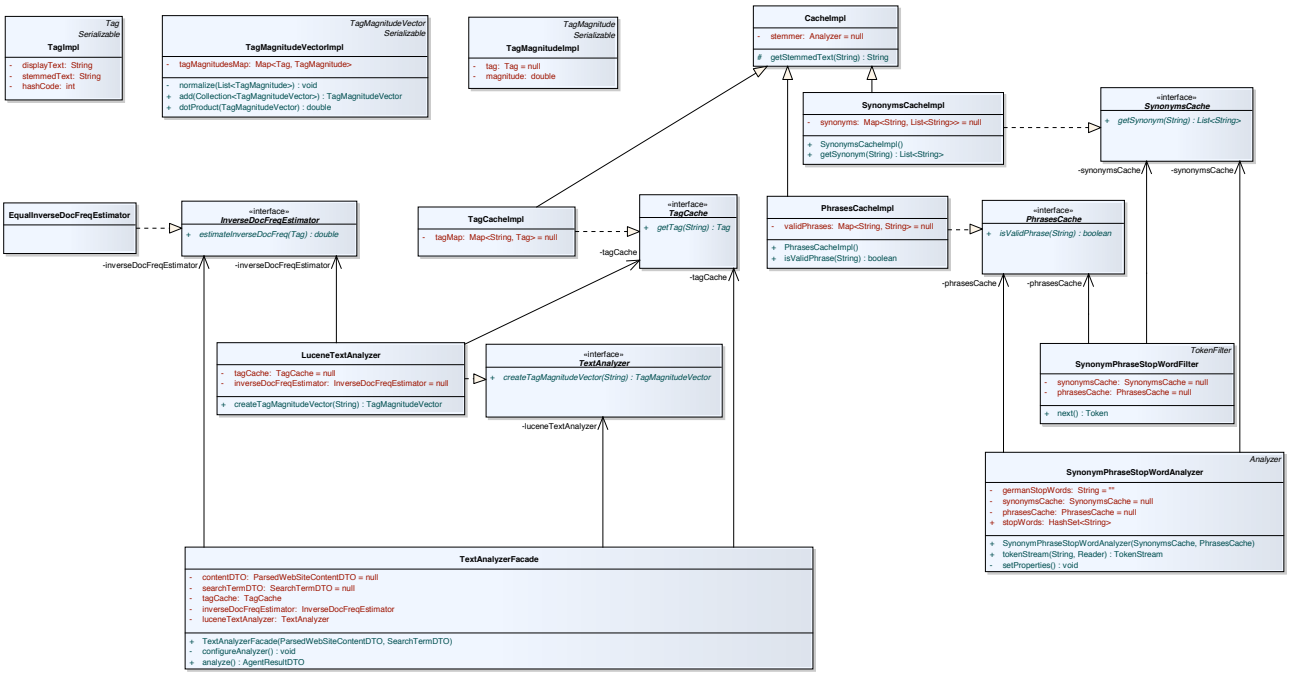


Abbildung 14.11: Klassendiagramm Textanalysis Package

Kernstück des Textanalysis-Package ist die Klasse TextAnalyzerFacade. Diese definiert die Analyse Konfiguration und verbirgt den komplexen Analyseablauf. Für die Analyse wird die Klasse LuceneTextAnalyzer verwendet, die mit Hilfe der Lucene Bibliothek einen Text in einen Daten-Vektor transferiert. Diese Implementation basiert auf einer Beispiel-Implementation aus “Collective Intelligence in Action”[3].

14.2.10 UserInterface Package

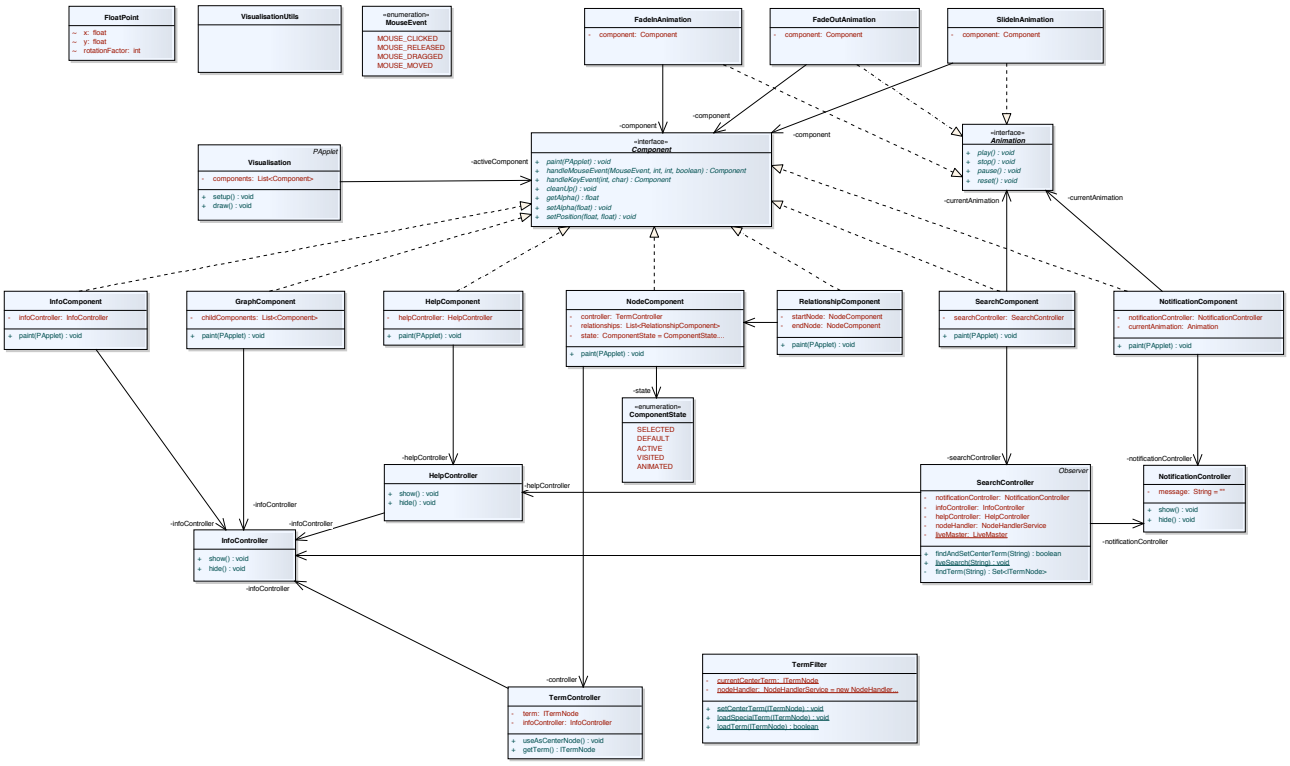


Abbildung 14.12: Klassendiagramm UserInterface Package

Dieses Package wird zur Erzeugung und Kontrolle des Graphical User Interfaces (GUI) benötigt. Erwähnenswert ist hierbei das Component Interface, das die Methoden für alle zu implementierenden Components definiert. Für diese Components werden Controller erstellt, welche die Kontrolle über die Aktionen übernehmen. Die Klasse Visualisation packt alle Components zusammen, delegiert das Zeichnen dieser weiter und stellt sie auf dem GUI dar.

14.3 Sequenzdiagramme

Die folgenden Sequenzdiagramme zeigen die wesentlichen Abläufe einer Suchanfrage. Dabei repräsentiert "JMS: QueueReceiver / QueueSender" die Schnittstelle zu JMS.

14.3.1 Master

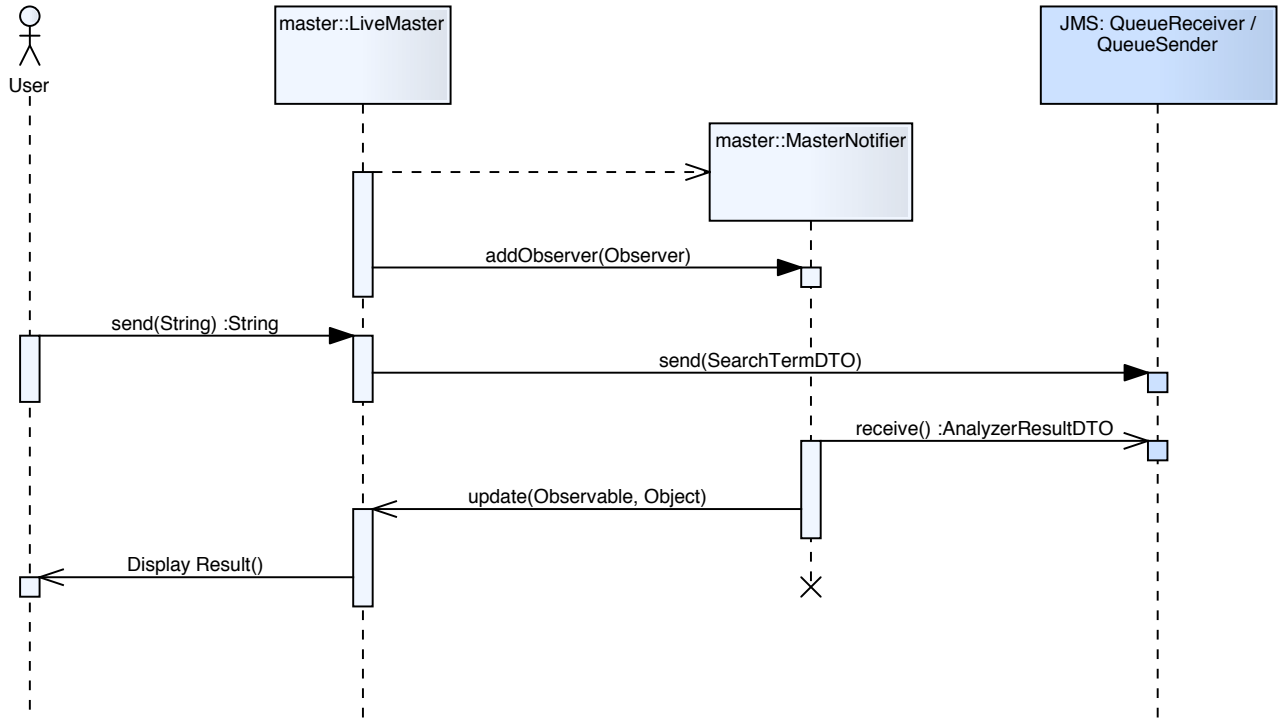


Abbildung 14.13: Sequenzdiagramm des Masters

Ein User stellt eine Suchanfrage an den Master. Dieser sendet die Anfrage an eine JMS Queue. Nach Beendigung des Auftrages, wird dem Master über den MasterNotifier das Resultat mitgeteilt. Der Master zeigt dann dem User das Resultat an. Die Verarbeitung der Suchanfrage wird in den nachfolgenden Sequenzdiagrammen genauer erläutert.

14.3.2 Agent

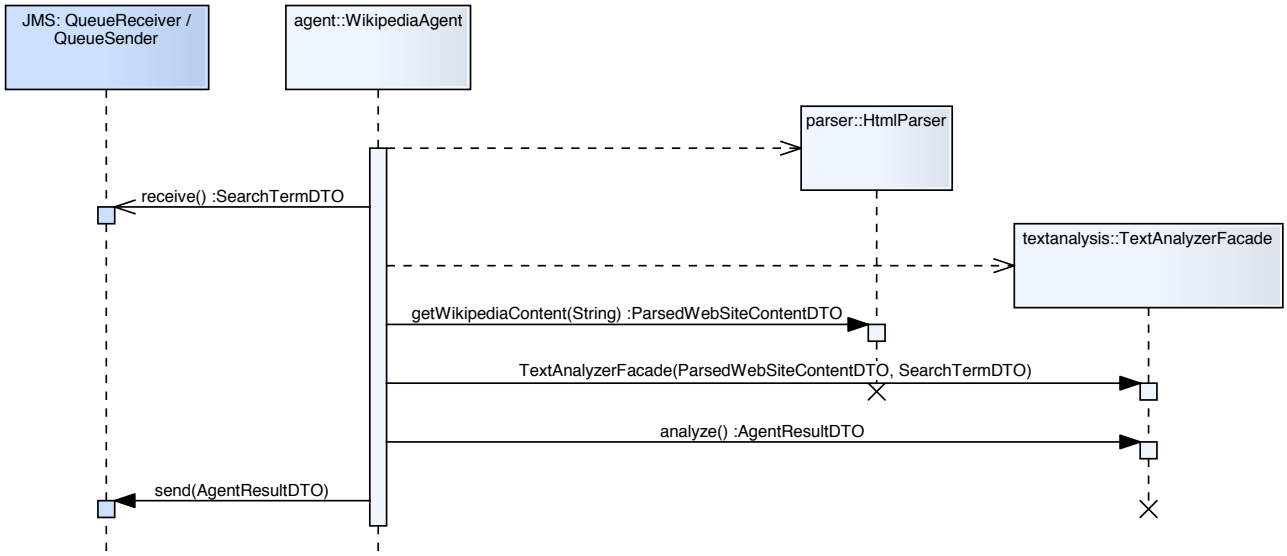


Abbildung 14.14: Sequenzdiagramm des Agents

Sobald ein neuer Auftrag vom Master in die JMS Queue gelangt, startet ein beliebiger Agent die Verarbeitung. Dabei wird zuerst die Datenquelle mit dem Html-Parser geparst. Der resultierende Text wird, mit Hilfe der TextAnalyzerFacade und den, von Lucene [1] zur Verfügung gestellten, Analyse-Methoden zu einem Term-Vektor geformt. Das Resultat wird dann, zur Gruppierung, dem Analyzer weitergeleitet.

[1] `http://lucene.apache.org/java/docs/`

14.3.3 Analyzer

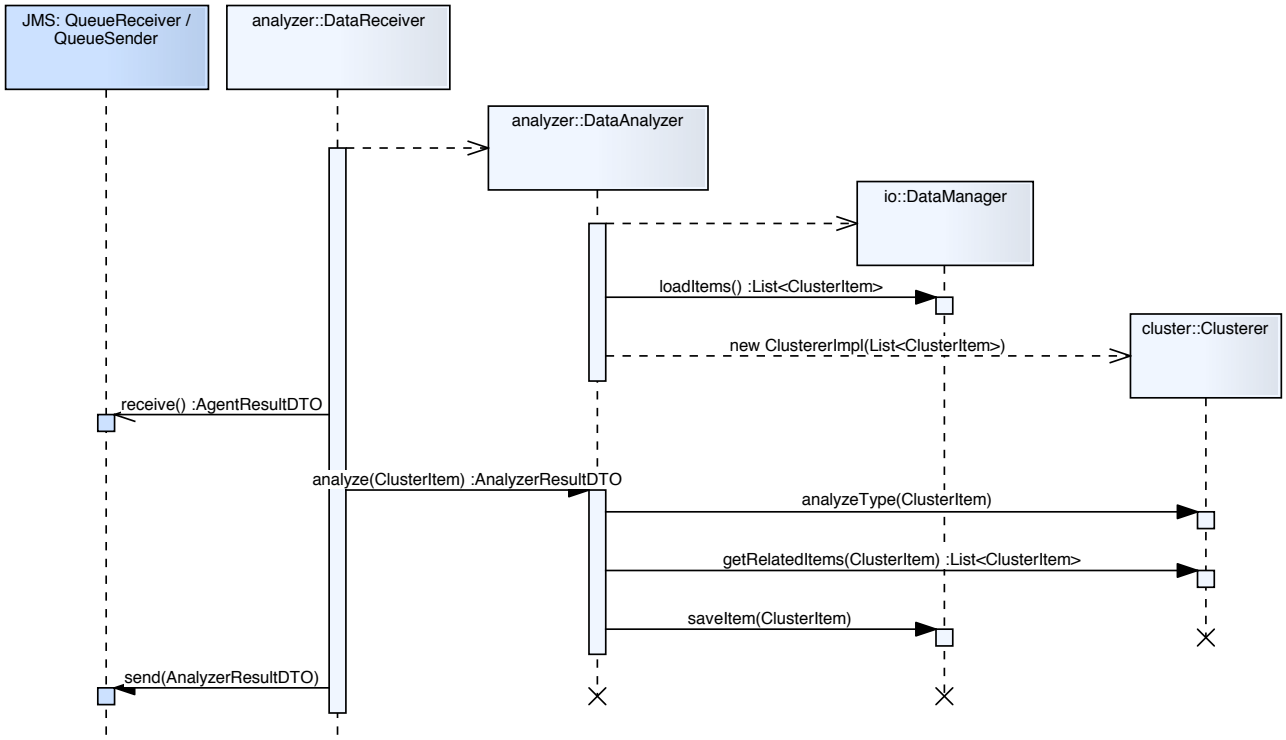


Abbildung 14.15: Sequenzdiagramm des Analyzers

Dieses Diagramm zeigt, wie der Analyzer einen Auftrag vom Agent entgegennimmt, verarbeitet und das Analyse Resultat an den Master zurücksendet.

Die vom DataManager geladene List<ClusterItem>, enthält alle bereits analysierten Terms. Diese dienen dem Cluster als Ausgangslage für das Clustering.

Kapitel 15

Implementation

Dieses Kapitel gibt Einblicke in den Source-Code des Projektes. Die Code Ausschnitte, welche hier gezeigt werden, haben keinen Anspruch auf Vollständigkeit, sondern sollen viel mehr aufzeigen, wie einzelne Probleme gelöst wurden.

15.1 Datenbankzugriff

Code Listing 15.1 DatabaseService.java

```
1  public class DatabaseService implements IDatabaseService {
2      private NeoService neoService;
3      private IndexService indexService;
4      private String databasePath;
5
6      public DatabaseService( String databasePath ) {
7          this.databasePath = databasePath;
8          neoService = new EmbeddedNeo( databasePath );
9          indexService = new NeoIndexService( neoService );
10     }
11
12     @Override
13     public ITermNode addTermNode(ITermNode termNode) {
14         Transaction tx = null;
15         try {
16             tx = neoService.beginTx();
17             termNode.validate();
18
19             if ( getNodesByPrototype( termNode ).size() != 0 ) {
20                 throw new TermNodeAlreadyExistsException();
21             }
22
23             Node node = neoService.createNode();
24             ITermNodeDAO dao = new TermNodeDAO( node );
25             dao.setData( termNode );
26             addNodeToIndex( termNode, node );
27
28             tx.success();
29         } catch ( InvalidTermNodeException ex ) {
30             throw ex;
31         } catch ( TermNodeAlreadyExistsException ex ) {
32             throw ex;
33         } catch ( Exception ex ) {
34             throw new DatabaseException( ex.getMessage() );
35         } finally {
36             tx.finish();
37         }
38         return (ITermNode) getSingleNode( getTermNodesByPrototype( termNode ) );
39         }
40     }
41 }
```

Die Klasse DatabaseService implementiert die Schnittstelle zur Neo4j[1] Datenbank. Im Konstruktor muss dem Service ein String mitgegeben werden, wo sich die Datenbank befindet, mit der gearbeitet werden soll. Ebenfalls im Konstruktor wird der eigentliche NeoService geladen, mit dem auf die Datenbank zugegriffen werden kann. Die Methode addTermNode(ITermNode termNode) zeigt exemplarisch

[1] http://www.neo4j.org/

für alle CRUD Operationen, wie mit einer Node umgegangen wird. Im Code ist zu sehen, dass eine Operation auf der Datenbank, immer in einer eigenen Transaktion ausgeführt wird.

Code Listing 15.2 NodeHandlerService.java

```
1  public class NodeHandlerService implements INodeHandlerService {
2      private IDatabaseService termDB;
3      private IDatabaseService typeDB;
4
5      public NodeHandlerService() {
6          termDB = TermDatabaseFactory.getDatabase();
7          typeDB = TypeDatabaseFactory.getDatabase();
8      }
9
10     @Override
11     public void addDataNode( ITermNode data ) {
12         ITermNode term = new TermNode();
13         term.setName( data.getType() );
14         termDB.addTermNode( data );
15     }
16
17     @Override
18     public void addTypeNode( ITermNode data ) {
19         ITermNode type = new TermNode();
20         type.setName( data.getType() );
21         typeDB.addTermNode( data );
22     }
23
24     @Override
25     public void linkDataNodes( ITermNode source, ITermNode destination ) {
26         termDB.connectTermNodes( source, destination );
27     }
28 }
```

Die Klasse NodeHandlerService setzt auf dem DatabaseService auf. Im Konstruktor werden mittels einer Factory zwei Instanzen der Datenbank geladen. Eine für die Term- und eine für die Type-Database. Nun muss man sich nicht weiter darum kümmern, welche Datenbank man verwenden muss, sondern kann einfach die entsprechende Methode des NodeHandlerServices nutzen. Wenn man eine Node in die Term-Database einfügen will, nutzt man die Methode addDataNode(ITermNode data) und wenn man einen Eintrag in die Type-Database machen will, braucht man addTypeNode(ITermNode type).

15.2 Anbindung von Processing

Code Listing 15.3 Component.java

```
1  public interface Component {
2      public void paint( PApplet canvas );
3      public Component handleMouseEvent( MouseEvent event, int mouseX, int
4              mouseY, boolean override );
5      public Component handleKeyEvent( int keyCode, char key );
6      public void cleanUp();
7      public float getAlpha();
8      public void setAlpha( float alpha );
9      public void setPosition( float x, float y );
10 }
```

Jedes Component, die im GUI realisiert werden soll, muss dieses Interface implementieren. Die Methode paint(PApplet canvas) zeichnet, auf dem entsprechenden Canvas, die gewünschten Objekte dieses Components.

Code Listing 15.4 Visualisation.java

```
1  public class Visualisation extends PApplet {
2      private static final long serialVersionUID = 1L;
3      private List<Component> components;
4
5      public Visualisation() {
6          components = new ArrayList<Component>();
7      }
8
9      @Override public void setup() {
10         ComponentController controller = new ComponentController();
11         components.add( new ExampleComponent( controller ) );
12     }
13
14     @Override public void draw() {
15         for( Component component : components ) {
16             component.paint( this );
17         }
18     }
19
20     @Override public void mouseClicked() {
21         for( Component component : components ) {
22             component.handleMouseEvent( MouseEvent.MOUSE_CLICKED,
23                     this.mouseX, this.mouseY, false );
24         }
25     }
26
27     @Override public void keyPressed() {
28         for( Component component : components ){
29             component.handleKeyEvent( keyCode, key );
30         }
31     }
32 }
```

Beispiel-Implementation der Klasse Visualisation, die von PApplet, der eigentlichen Processing Klasse, abgeleitet ist. Die Methode setup() wird beim Starten als erstes ausgeführt und erzeugt alle gewünschten Components und speichert diese in einer ArrayList. Die Methode draw() wird laufend ausgeführt und zeichnet alle Components, die sich in der Component Liste befinden, aufs GUI. Die Methoden mouseClicked() und keyPressed() nehmen Maus- und Tastaturaktionen entgegen und leiten sie an die jeweiligen Components weiter, wo die Aktionen bearbeitet werden.

Wie man im Source Code erkennen kann, wird dem Konstruktor der Component ein Controller mitgegeben. Über diesen Controller können die einzelnen Components untereinander Kommunizieren und Daten austauschen. Ein einzelner Controller kann mehreren verschiedenen Components übergeben werden.

15.3 Kommunikation zwischen verteilten Rechnern

Code Listing 15.5 DataReceiver.java

```
1  public abstract class DataReceiver extends Thread {
2      public abstract void handleRequest(AgentResultDTO agentResultDTO);
3      private String reciverQueueName, brokerHostName, brokerPort;
4      private String brokerHostNameProperty, brokerHostPortProperty, publisherTopicProperty;
5      private ConnectionFactory connectionFactory = null;
6      private QueueConnection queueConnection = null;
7      private QueueSession queueSession = null;
8      private Queue queue = null;
9      private QueueReceiver queueReceiver = null;
10     private TopicConnection connection = null;
11     private TopicSession session = null;
12     private Topic publisherTopic = null;
13     private TopicPublisher publisher;
14
15     private void initJMS() {
16         setProperties();
17         connectionFactory = new com.sun.messaging.ConnectionFactory();
18         connectionFactory.setProperty(brokerHostNameProperty, brokerHostName);
19         connectionFactory.setProperty(brokerHostPortProperty, brokerPort);
20         queueConnection = connectionFactory.createQueueConnection();
21         queueSession = queueConnection.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
22         queue = queueSession.createQueue(reciverQueueName);
23         queueReceiver = queueSession.createReceiver(queue);
24         connection = connectionFactory.createTopicConnection();
25         session=connection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
26
27         publisherTopic = session.createTopic(publisherTopicProperty);
28         publisher = session.createPublisher(publisherTopic);
29         queueConnection.start();
30     }
31
32     protected void sendResponse(AnalyzerResultDTO
33             analyzerResultDTO ) throws JMSException {
34         publisher.send(session.createObjectMessage(analyzerResultDTO));
35     }
36
37     public void run() {
38         Message message = null;
39         while (!isShutdown()) {
40             message = queueReceiver.receive();
41             ObjectMessage oms = (ObjectMessage) message;
42             if (ObjectMessage instanceof AgentResultDTO ){
43                 AgentResultDTO analyzerResultsDTO=(AgentResultDTO)
44                     oms.getObject();
45                 handleRequest(analyzerResultsDTO);
46             }
47         }
48         if(queueConnection != null) {
49             queueConnection.close();
50         }
51     }
52 }
```

Dies ist ein Beispiel der JMS-Implementation der abstrakten Klasse DataReceiver. Spezialisierungen müssen die abstrakte Methode handleRequest(AgentResultDTO agentResultDTO) implementieren. Dabei wird definiert, wie die entsprechenden Resultate behandelt werden.

15.4 Analyse von zusammengehörigen Begriffen

Code Listing 15.6 TagMagnitudeVectorImpl.java

```
1  @Root public class TagMagnitudeVectorImpl implements TagMagnitudeVector, Serializable{
2
3  @ ElementMap private Map<Tag, TagMagnitude> tagMagnitudesMap;
4
5  public double dotProduct(TagMagnitudeVector o) {
6      Map<Tag, TagMagnitude> otherMap = o.getTagMagnitudeMap();
7      double dotProduct = 0.;
8      for ( Tag tag : tagMagnitudesMap.keySet() ) {
9          TagMagnitude otherTm  = otherMap.get(tag);
10             if (otherTm != null) {
11                 TagMagnitude tm = tagMagnitudesMap.get(tag);
12                 dotProduct += tm.getMagnitude() * otherTm.getMagnitude();
13             }
14     }
15     return dotProduct;
16 }
```

Diese Implementation des TagMagnitudeVector zeigt, wie das Skalarprodukt (dotProduct) von zwei Vektoren berechnet wird. (Erstellt, anhand der Beispiel-Implementation aus dem Buch “Collective Intelligence in Action”, [3]). Dabei wird das Skalarprodukt aus der Grösse (Magnitude) der Vector Tags gebildet. Die Annotations “@Root” “@ElementMap” definieren die, für die XML-Serialisierung benötigten Elemente.

15.5 Clustering

Code Listing 15.7 ClusterImpl.java

```
1  public class ClusterImpl implements Cluster {
2  private TagMagnitudeVector center = null;
3  private List<ClusterItem> items = null;
4  private String clusterName = "";
5
6  public void computeCenter() {
7      if (this.items.size() == 0) {
8          return;
9      }
10
11     List<TagMagnitudeVector> tmList = new ArrayList<TagMagnitudeVector>();
12     for ( ClusterItem item : items ) {
13         tmList.add( item.getTagMagnitudeVector() );
14     }
15
16     List<TagMagnitude> emptyList = Collections.emptyList();
17     TagMagnitudeVector empty = new TagMagnitudeVectorImpl( emptyList );
18     this.center = empty.add(tmList);
19 }
```

Dieser Ausschnitt zeigt, wie aus einzelnen Items eines Clusters, dessen Zentrum gebildet wird.

Code Listing 15.8 ClustererImpl.java

```
1  public class ClustererImpl implements Clusterer {
2  private boolean reassignAll = true;
3  private List<ClusterItem> textDataSet = null;
4  private List<Cluster> clusters = null;
5
6  private void createClusters() {
7      ClusterImpl cluster = null;
8      String lastType = "";
9
10     //Sortierung nach Type
11     Collections.sort(textDataSet);
12     for (ClusterItem item : textDataSet) {
13         if ((item.isConfirmed()) || (item.hasType() && !reassignAll)) {
14             if (!item.getType().equals(lastType)) {
15                 if (cluster != null) {
16                     cluster.computeCenter();
17                     clusters.add(cluster);
18                 }
19                 cluster = new ClusterImpl(item.getType());
20                 lastType = item.getType();
21             }
22             cluster.addDataItem(item);
23         }
24     }
25     if (cluster != null) {
26         cluster.computeCenter();
27         clusters.add(cluster);
28     }
29 }
30
31 private Cluster getClosestCluster(ClusterItem item) {
32     Cluster closestCluster = null;
33     double hightSimilarity = 0.0;
34     double similarity = 0.0;
35     Vector<Double> similaritylist = new Vector<Double> ();
36     for (Cluster cluster : this.clusters) {
37         similarity=cluster.getCenter()
38             .dotProduct(item.getTagMagnitudeVector());
39         if (hightSimilarity < similarity) {
40             hightSimilarity = similarity;
41             closestCluster = cluster;
42         }
43     }
44 }
45
46 private void assignToClosestCluster() {
47     Cluster newCluster = null;
48     for (ClusterItem item : this.textDataSet) {
49         if (!((item.isConfirmed()) || (item.hasType() && !reassignAll))) {
50             newCluster = getClosestCluster(item);
51             if (newCluster != null) {
52             item.setType(newCluster.getClusterName());
53                 item.setDistanceToCentre(newCluster.getCenter()
54                     .dotProduct(item.getTagMagnitudeVector()));
55                 newCluster.addDataItem(item);
56             }
57         }
58     }
59 }
```

Dieser Ausschnitt zeigt die Implementation des Clustering-Algorithmus. (Erstellt, anhand der Beispiel-Implementation aus dem Buch "Collective Intelligence in Action", [3]). Die Funktion CreateClusters() baut aus den vordefinierten Typen die Cluster auf. Dafür werden alle Items verwendet, die als "Confirmed" markiert sind. Je nach Option können bereits klassifizierte Items ebenfalls zur Clusterbildung beigezogen werden. Diese werden nicht mehr neu analysiert, was zu einer schnelleren Analyse führt.

GetClosestCluster(ClusterItem item) liefert zu jedem Item, den ihm am nächsten liegenden Cluster. Dazu wird das Skalarprodukt zwischen den Clusterzentren und dem Item berechnet.

AssignToClosestCluster() weist alle, dem Clustering-Algorithmus übergebenen Items, dem nächsten Cluster zu. Dabei wird für das jeweilige Item, der Type (entspricht dem Clusternamen), sowie die Distanz zum Clusterzentrum (DistanceToCentre) gesetzt.

Kapitel 16

Testing

16.1 Softwaretests

16.1.1 Komponententests

Verschiedene Komponententests wurden mit JUnit[1] durchgeführt. Diese waren besonders zu Beginn der Entwicklungsphase wertvoll, wurden jedoch bei der Umstellung auf eine neuere Version der Datenhaltung noch nicht aktualisiert.

16.1.2 Systemtests

Die folgenden Testcases wurden auf zwei verschiedenen Testsystemen durchgeführt. Falls abweichende Resultate entstehen sollten, werden diese jeweils speziell vermerkt.

Systemtestumgebung

Das erste Testsystem umfasst ein Macbook Pro mit 2GB RAM, Dual-Core 2.4 GHz Prozessor, NVIDIA GeForce 8600M 256MB Grafikchip, 15.4" Bildschirm und dem Betriebssystem Mac OS X 10.5. Auf dem Testrechner wurde Sun's Java 1.6.0_07 installiert. Das zweite Testsystem umfasst einen Windows XP Rechner mit Dual-Core 2.6 GHz, 4GB RAM und ATI FireGL V5200 mit 256MB Grafikchip.

Bei beiden Systemen kommt als JMS-Server ein Virtual Server der HSR, mit 512MB RAM und Windows XP zum Einsatz. Darauf ist die freie JMS-Server-Software Open Message[2] installiert.

[1] `http://www.junit.org`
[2] `https://mq.dev.java.net/`

Testfälle Wissensextraktion

#	Testfall	Erwartetes Resultat	Resultat
1	Crawlen von Datenquellen	Für einen beliebigen Begriff kann eine Datenquelle gecrawlt werden. Daraus resultiert eine textuelle Begriffserklärung.	OK
2	Text in einzelne Wörter zerlegen	Der Text ist frei von Satzzeichen in einzelne Wörter zerlegt.	OK
3	Nomen aus Text extrahieren	Der eingegebene Text enthält nach der Filterung nur noch Nomen.	OK
4	Stoppwörter aus Text filtern	Alle in der Stoppwort-Liste enthaltenen Wörter sind aus dem Text entfernt.	OK
5	Term Vektor aus einem Text erzeugen	Term Vektor wird erzeugt und enthält alle, im Text vorkommenden Wörter und deren Wertigkeit.	OK
6	Clusterbildung anhand von Prototypen	Aus mehreren Prototypen werden die jeweiligen Cluster Typen gebildet.	OK
7	Term clustern. Eine Liste mit Termen dem Clustering Algorithmus übergeben.	Die Terme werden den entsprechenden Cluster zugewiesen. Die Zuweisung entspricht den zuvor definierten Erwartungen, mit einer Erfolgsquote von mehr als 75%.	OK
8	Determinismus Test, Test #7 wird mehrmals wiederholt.	Alle Testdurchläufe führen zum selben Resultat.	OK
9	Verwandte Typen. Eine Liste mit Termen dem Clustering Algorithmus übergeben.	Zu jedem Term werden die verwandten Terme ausgegeben. Dies entspricht den zuvor definierten Erwartungen, mit einer Erfolgsquote von mehr als 75%.	OK
10	Mehrere Terme, für die keine Prototypen existieren werden dem Clustering Algorithmus übergeben.	Es wird erkannt, dass die Terme keiner vorhandenen Gruppe entsprechen und die Terme werden speziell markiert.	OK
11	Bei einem Teil der in Test #10 analysierten Terme wird der Typ definiert. Die Restlichen werden nicht verändert. Die Analyse wird nochmal gestartet.	Die zuvor markierten Termse werden nun den Entsprechenden Typen zugewiesen. Die markierten Terme haben sich reduziert.	OK

Tabelle 16.1: Resultate des Systemtests des Wissensextraktions-Prototypen

Testfälle Verteiltes System

#	Testfall	Erwartetes Resultat	Resultat
1	Mehrere Agenten auf demselben System starten.	Agenten starten problemlos und sind im Agent Control GUI sichtbar.	OK
2	Mehrere Agenten auf unterschiedlichen Systemen starten.	Agenten starten problemlos und sind im Agent Control GUI sichtbar.	OK
3	Agent Starten. Suchbegriff mit dem Master absetzen.	Der Auftrag wird von Agenten verarbeitet.	OK
4	Mehrere Agenten starten. Suchbegriff mit dem Master absetzen.	Der Auftrag wird von GENAU einem Agenten verarbeitet.	OK
5	Sämtliche Agenten stoppen. Suchbegriff mit dem Master absetzen. Master stoppen. Agenten starten.	Der Auftrag wird vom Agenten verarbeitet, sobald dieser gestartet wird.	OK
6	Mehrere Agenten starten. Suchbegriff mit dem Master absetzen. Analyzer starten.	Der Auftrag wird vom Agenten verarbeitet. Sobald der Analyzer gestartet wird, analysiert er das Resultat des Agenten. Der Master empfängt das Analyseresultat das Analyzers.	OK
7	Mehrere Agenten manuell starten und stoppen.	Statusänderung der Agenten ist im Agent Control GUI sichtbar.	OK
8	Agent remote starten und stoppen.	Ausgewählter Agent startet und stoppt sich.	OK

Tabelle 16.2: Resultate des Systemtests des Verteilten Systems

Testfälle des Visualisierungs-Prototypen

#	Testfall	Erwartetes Resultat	Resultat
1	Start der Applikation mittels Doppelklick.	Applikation startet innerhalb von 10 Sekunden im Vollbildmodus.	OK
2	Laden des Default-Terms beim Applikationsstart.	Applikation startet mit dem Default-Begriff 'Schweiz'.	OK
3	Navigation zum nächsten Term mittels Klick auf Term.	Der angeklickte Term wird im Zentrum neu angezeigt und seine verwandten Terme geladen.	OK
4	Navigation zurück zum ursprünglichen Term.	Es ist möglich, zum ursprünglichen Term zurückzukehren, dessen Ansicht mit der Default-Ansicht übereinstimmt.	OK
5	Eingabe eines bekannten, eindeutigen Suchterms.	Der Suchterm wird mit seinen Relationen direkt im GUI dargestellt.	OK
6	Eingabe eines bekannten, aber mehrdeutigen Suchterms.	Der erstbeste Suchterm wird angezeigt, eine Auswahl der Möglichkeiten wird in einer Liste dargestellt.	OK
7	Eingabe eines unbekannten, eindeutigen Suchterms.	Meldung, dass Term unbekannt und dass Analyse gestartet wird. Resultat wird nach Ablauf der Analyse angezeigt.	OK
8	Eingabe eines unbekannten, mehrdeutigen Suchterms.	Meldung, dass Term unbekannt und dass Analyse mit Auswahl verschiedener Möglichkeiten gestartet werden kann.	OK
9	Eingabe eines Suchterms mit Umlaut.	Suche funktioniert trotz Umlaut-Zeichen.	OK
10	Eingabe eines Teils des Suchterms.	Suche listet verschiedene Treffermöglichkeiten, analog zur Anzeige bei Mehrdeutigkeit, auf.	OK
11	Starten der Suche ohne Suchbegriff.	Suche wird nicht gestartet. Stattdessen Anzeige, dass Suchterm mindestens 3 Zeichen umfassen muss.	OK
12	Starten der Suche mit Suchbegriff < 3. Zeichen	Suche wird nicht gestartet. Stattdessen Anzeige, dass Suchterm mindestens 3 Zeichen umfassen muss.	OK
13	Performanz bei ca. 15 angezeigten Termen messen.	Framerate sinkt nicht unter 15 frames per second (fps).	OK
14	Performanz einer Suchabfrage eines bekannten Terms messen.	Suchabfrage dauert nicht länger als 5 Sekunden.	OK
15	Performanz einer Suchabfrage eines unbekannten Terms messen.	Suchabfrage und Analyse dauern nicht länger als 20 Sekunden.	OK
16	Chaostests.	Erfolgen soweit üblicher Erwartungen. Gewisse Fehler bei bestimmten Suchtermen sind nicht auszuschliessen.	OK

Tabelle 16.3: Resultate des Systemtests des Visualisierungs-Prototypen

Testfälle Datenhaltung

#	Testfall	Erwartetes Resultat	Resultat
1	Speicherung eines Terms, mit den minimal benötigten Attributen (Name, Alias, Type).	Speicherung des Terms gelingt, sofern der Term nicht schon besteht.	OK
2	Speicherung eines Terms, mit weniger als den minimal benötigten Attributen.	Speicherung des Terms gelingt nicht. Entsprechende Meldung wird ausgegeben.	OK
3	Speicherung eines Terms, mit mehr als den minimal benötigten Attributen.	Speicherung des Terms gelingt, sofern der Term nicht schon besteht.	OK
4	Speicherung eines bereits existierenden Terms.	Speicherung des Terms gelingt nicht. Entsprechende Meldung wird ausgegeben.	OK
5	Speicherung eines nur in Type, Alias und Name identischen Terms.	Speicherung des Terms gelingt nicht, da Kombination zwischen Type, Alias und Name als unique gilt.	OK
6	Auslesen aller Terme.	Alle Terme werden geladen, inklusive dem 'Root'-Knoten.	OK
7	Auslesen bestimmter Terme aufgrund eines gewissen Suchobjektes.	Terme, welche den Suchkriterien entsprechen, werden zurückgegeben.	OK
8	Auslesen aller Typen.	Alle, in der Typ-Datenbank gespeicherten Typen, werden zurückgegeben.	OK
9	Verbinden zweier Typen miteinander	Typen werden verbunden, sofern keine bestehende Verbindung besteht.	OK
10	Verbinden zweier Terme miteinander.	Terme werden verbunden, sofern keine bestehende Verbindung besteht.	OK
11	Verbinden eines Terms mit einem nicht-existenten Term.	Versuch schlägt fehl. Entsprechende Meldung wird geloggt.	OK
12	Verbinden eines Types mit einem nicht-existenten Type.	Versuch schlägt fehl. Entsprechende Meldung wird geloggt.	OK
13	Löschen aller, in der Term-Datenbank gespeicherten Terme.	Alle Terme ausser dem Root-Knoten und deren Relationen werden gelöscht.	OK
14	Löschen aller in der Type-Datenbank gespeicherten Terme.	Alle Typedefinitionen und deren Relationen werden gelöscht.	OK

Tabelle 16.4: Resultate des Systemtest des Datenhaltungs-Prototypen

16.2 Usability-Tests

16.2.1 Evaluation des Beta-Prototypen

Testbeschreibung

Eine Testperson, deren Computerkenntnisse von uns als mittelmässig eingestuft wurden, hatte sich bereit erklärt, während einer Stunde einen von uns vorbereiteten Test zu durchlaufen. Die gestellten Testfälle sind mit dem entsprechenden Resultat tabellarisch aufgeführt. Das Entwicklerteam hat sich während des Tests aufmerksam beobachtend verhalten. Das Verhalten der Testperson wurde mit Notizen aufgezeichnet. Auf eine Videoaufnahme oder Screenüberwachung wurde aus Zeitgründen verzichtet.

#	Beschreibung	Resultat
1	Starten der Applikation. Doppelklick auf 'Visualisierung.jar'.	Funktioniert wie erwartet. Testperson erkennt, wenn Applikation bereit für ihre Eingaben ist.
2	Suche des Begriffes "Schweiz".	Klick auf 'Search', der nicht notwendig wäre, aber ansonsten keine weiteren Probleme. Die Testperson erkennt ausserdem, dass der Begriff nach dem gesucht werden soll, bereits derselbe ist, der bei Applikationsstart angezeigt wird.
3	Suche des Begriffes "Zürich".	Die Testperson gibt Zeichen ein, aber das Suchformular erscheint erst, nachdem sie mit der Maus in den oberen rechten Bildschirmrand gefahren ist.
4	Navigieren zum Begriff "Deutschland".	Die Testperson versucht sich auf Anhieb per Mausklick durch den Graphen zu navigieren und findet den Node Deutschland nach ein paar Klicks.
5	Chaostests mit eigenen Suchbegriffen.	Wiederum mehrmals das gleiche Problem mit dem versteckten Suchfeld wie bei Testcase 4. Ausgabe bei Begriffen mit Multivalenz nicht ganz klar, was jetzt zu tun ist.
6	Beenden der Applikation.	Die Testperson sucht einen Knopf oben rechts, um die Applikation zu beenden. Da dieser dort nicht vorhanden ist, sucht sie in den anderen Bildschirmecken. Schliesslich versucht sie es mit der Taste 'ESC', was auch die gewünschte Wirkung erzielt.

Tabelle 16.5: Resultat der Evaluation des Beta-Prototypen durch eine externe Testperson

Testresultat

Bei der Benutzung des Beta-Prototypen, sind noch einige Mängel aufgefallen. Beispielsweise, hat die Testperson die Funktionsweise des Scrollrades (zur Festlegung der Tiefe der Node-Stufe), nur durch Zufall entdeckt und war bei dessen Resultat entsprechend verwirrt. Ausserdem wurde versucht, direkt über die Tastatur einen Suchbegriff einzugeben, ohne dass das dafür notwendige Component sichtbar

war. Obwohl die Suche nach der erwarteten Bestätigung mit der “Enter”-Taste funktionierte, war sich die Testperson nicht über die eingebenen Zeichen im Klaren. Ausserdem hatte die Testperson etwas Probleme, die Applikation zu beenden. Erst nach einigen Sekunden wurde ihr klar, dass die ’ESC’-Taste die Applikation beendet.

Daraus folgende Anpassungen

Es wurde entschieden, die Scrollrad-Funktion wieder auszubauen. Die Testperson kam mit dieser Steuerung nicht zurecht und konnte auch mit dem Konzept nichts anfangen. Das Suchfeld soll neu spätestens beim Drücken einer Taste angezeigt werden, nicht erst wenn der Benutzer sich mit der Maus über dem Label ’Search’ befindet. Ausserdem sollte ein ’Close’-Button eingeführt werden, um dem Benutzer das Schliessen der Applikation zu vereinfachen.

Kapitel 17

Projektmanagement

17.1 Prototypen, Release, Meilensteine

Das Projekt umfasst drei Prototypen und fünf Meilensteine. Dabei decken sich die ersten drei Meilensteine mit dem Release der drei Prototypen. Meilenstein vier ist ein feature freeze und der fünfte und gleichzeitig letzte Meilenstein, ist die Beendigung der Dokumentation und die Abgabe der Arbeit.

17.1.1 Prototypen Planung

Prototyp: Wissensextraktion

- Evaluation bestehender Lösungen oder Neu-Entwicklung eines Html-Crawlers.
- Evaluation verschiedener Architekturen von verteilten Systemen.
- Entwicklung einer verteilten Lösung zur Informationsbeschaffung / Informationsextraktion.
- Evaluation verschiedener Clustering-Algorithmen zur Typenzuordnung.
- Implementation eines Clustering-Algorithmus / Clustering-Systems.

Prototyp: Datenhaltung

- Evaluation graph-basierter Datenbanken.
- Implementation der Datenhaltung der Knowledge-Objects.
- Entwicklung eines einfachen Tools, um gespeicherte Daten zu korrigieren.

Prototyp: Visualisierung

- Entwurf einer Architektur für das Darstellen von Processing-Objekten.
- Entwurf eines Graphical User Interfaces.
- Entwicklung der Visualisierungsplattform.

17.1.2 Meilensteine

Meilenstein	Datum	Beschreibung
M1	16.03.09	Prototyp Wissensextraktion ist abgeschlossen.
M2	20.04.09	Prototyp Datenhaltung ist abgeschlossen.
M3	11.05.09	Prototyp Visualisierung ist abgeschlossen.
M4	01.06.09	Feature freeze: Alle Features sind implementiert und getestet.
M5	12.06.09	Abgabe: Projekt abgeschlossen und alle Abgaben erledigt.

Tabelle 17.1: Meilensteine

17.2 Team, Rollen und Verantwortlichkeiten

Das Projektteam besteht aus den drei Mitgliedern Kevin Gaunt, Tobias Löffel und Beat Weber. Regelmässig werden an Scrum[4] angelehnte Besprechungen zur Koordination der Abläufe, sowie der Aufgabenverteilung abgehalten. Jedes Teammitglied ist für seinen Bereich verantwortlich. Entscheidungen werden jedoch gemeinsam getroffen. Dabei ist jedes Teammitglied gleichberechtigt.

Name	Verantwortlichkeiten
Kevin Gaunt	GUI, Visualisierung der Daten, Datenhaltung
Tobias Löffel	Webcrawling, Datenquellen, Text Mining
Beat Weber	Verteiltes System und Kommunikation, Clustering

Tabelle 17.2: Teammitglieder und Verantwortlichkeiten

Während des gesamten Projektes, wird das Team von einem Betreuer beratend unterstützt. Zusätzlich wird die Arbeit von einem Experten geprüft.

Name	Funktion	E-Mail Adresse
Prof. Dr. Josef Joller	Betreuender Dozent	jjoller@hsr.ch
Matthias Lips	Experte	matthias.lips@hslu.ch

Tabelle 17.3: Liste der externen beteiligten Personen

17.3 Risiken

Die folgende Risikoanalyse zeigt die Bewertung möglicher Risiken. Die Massnahmen zur Reduktion der Risiken fliessen in die Aufwandschätzung der einzelnen Arbeitsschritte ein. Die Summe der gewichteten Schäden wird in die zeitliche Reserve einberechnet.

Risiko Bewertungen								
Risk ID	**Risiko**	**Auswirkung**	**Massnahme**	**Kosten der Massnahmen in Stunden**	**Max. Schaden in Stunden**	**Wahrscheinlichkeit des Eintreffens**	**Gewichteter Schaden in Stunden**	**Priorität**
R01	Ausfall von Teammitglied	Verzögerung, Zusätzliche Einarbeitungszeit	Besprechungen, jeder kann Arbeiten des Andern übernehmen, Tätigkeiten kommunizieren	10	50	15%	8	Hoch
R02	Graph basierte Datenbank Technologie nicht ausgereift genug	Umstellung auf relationale Datenbank	Prototyping Intensives Testen, Evaluation	15	60	20%	12	Hoch
R03	Wahl falsches Clustering Algorithmus	Fehlerhafte Zuordnungen	Analyse mehrerer varianten, Intensives Technologie Studium	15	50	50%	25	Hoch
R04	Problem mit Verteiltem System	Rückbau Verteilung, Performance Einbussen, kein Parallelbetrieb	Intensives Technologie Studium	10	60	15%	9	Hoch
R05	Neu Technologien (für Teammitglieder)	Hohe Einarbeitungszeit, Erschwerte Aufwandsschätzung, Funktionalität teils fraglich	Prototyping, Intensives Technologiestudium	15	70	30%	21	Hoch
R06	Zu optimistische Aufwandsschätzung	Arbeiten dauern länger als angenommen	Zusätzliche Reserve	40	40	60%	24	Hoch
R07	Fehler in Software Architektur	Verzögerungen, aufwändiges refactoring	Reviews, Milestones, Teamsitzungen, Testing, Prototyping	10	20	10%	2	Mittel
R08	Datenverlust	Verlust des Projektes oder Teilen davon	SVN	5	720	1%	7	Klein
R09	Übrige unvorhergesehene Risiken	Beliebige (je nach Risiko)	Zusätzliche Zeitreserven	40	40	80%	32	Sehr Hoch
	Total Kosten in Arbeitspaketen enthalten			160				
	Total Rückstellungen						140	

Tabelle 17.4: Risikoanalyse

17.4 Prozessmodell

Für das Projektmanagement werden mehrere Prozessmodelle kombiniert und dabei die brauchbaren und sinnvollen Konzepte der jeweiligen Modelle verwendet. Die Abläufe entsprechen dabei am ehesten dem agilen Softwareentwicklungs-Prozess Scrum[4]. Die Teammitglieder organisieren ihre Arbeit weitgehend selbst und wählen auch die eingesetzten Software-Entwicklungswerkzeuge und -Methoden. Dabei wurden Ziele, Probleme und geplante Arbeitsschritte mindestens einmal pro Woche, in Teammeetings besprochen. Nach grösseren Arbeiten wurden nach Bedarf zusätzliche Besprechungen geführt, um das weitere Vorgehen zu planen. Kombiniert wurde dies mit dem, von Extreme Programming übernommen Wertesystem: Kommunikation, Einfachheit, Feedback, Mut und Respekt.

Kapitel 18

Projektmonitoring

18.1 Soll-Ist-Zeitvergleich

Arbeitspaket	Soll-Stunden	Ist-Stunden
Ideenfindung	60	50
Technologiestudium	100	90
Implementation	530	665
Testing	60	65
Dokumentation	220	260
Reserve	140	0
Total	1110	1130

Tabelle 18.1: Soll-Ist-Zeitvergleich gesamt

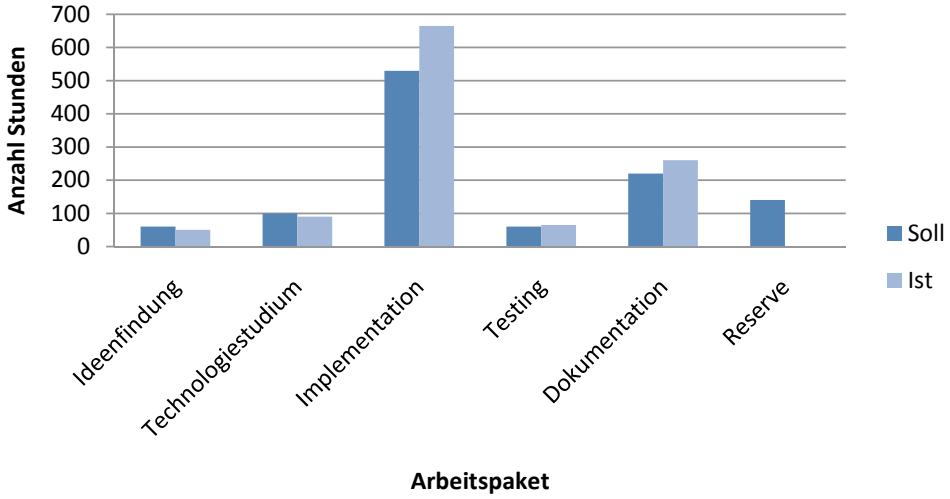


Abbildung 18.1: Soll-Ist-Zeitvergleich gesamt

Arbeitspaket	Soll-Stunden	Ist-Stunden
Verteiltes System	40	60
Text Mining	120	110
Clustering	100	110
Datenhaltung	120	140
Visualisierung	150	245
Total	530	665

Tabelle 18.2: Soll-Ist-Zeitvergleich Implementation

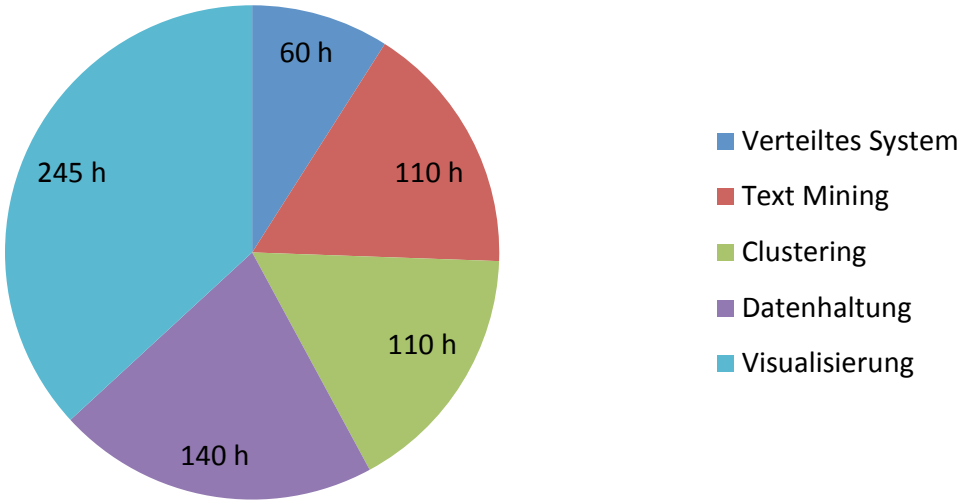


Abbildung 18.2: Soll-Ist-Zeitvergleich Implementation

Name	Zeitaufwand
Kevin Gaunt	384
Tobias Löffel	374
Beat Weber	372
Total	1130

Tabelle 18.3: Arbeitsaufwand pro Person pro Woche

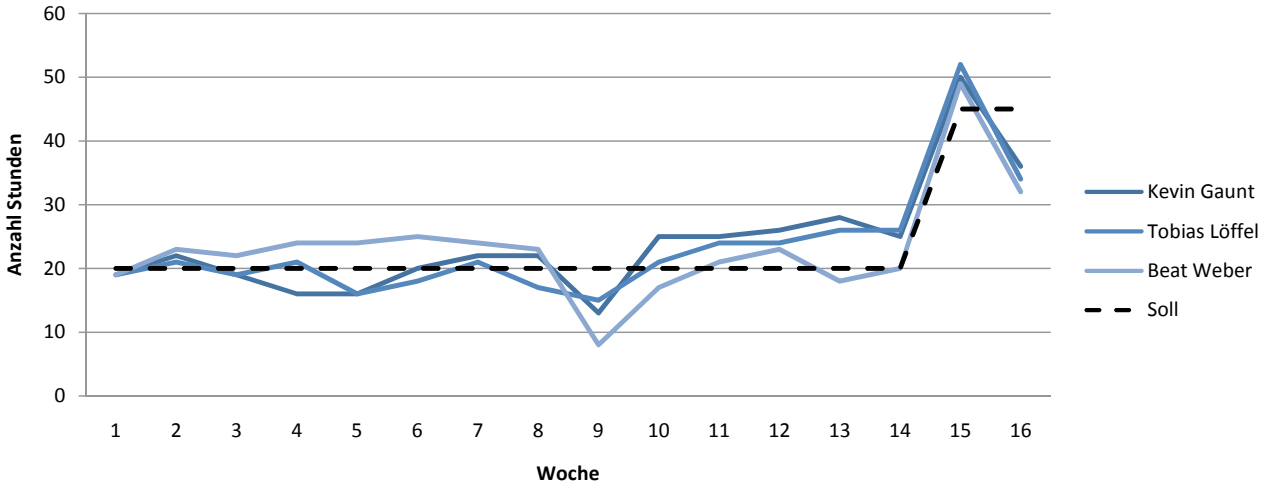


Abbildung 18.3: Arbeitsaufwand pro Person pro Woche

Dank einer guten Aufwandschätzung und Zeitplanung, wichen die effektiv benötigten Stunden nicht beträchtlich von den budgetierten Stunden ab. Im Laufe des Projekts wurde erkannt, dass die anhand der Risikoanalyse eingeplanten Reserven ausreichen. Zum Einen wirkten die dort geplanten Massnahmen, zum Anderen, trafen einzelne Risiken nicht ein. Daher wurde entschieden, die Reserve in die Implementation zu investieren, um den Umfang der Arbeit, durch einige Features zu erweitern.

18.2 Codestatistik

Paket	Anzahl Klassen	Anzahl Methoden	Anzahl Codezeilen
Agent	5	68	797
Analyzer	7	41	626
Helper	9	64	577
IO	3	40	328
Maintenance	11	41	738
Master	3	10	235
Databaselayer	3	82	896
NodeHandlerService	3	17	192
Textanalysis	13	55	631
Tools	2	3	138
UserInterface	32	290	2665
Total	**91**	**711**	**7823**

Tabelle 18.4: Codestatistik
Stand: 10.06.2009

Teil IV

Anhang

Kapitel 19

Persönliche Berichte

19.1 Kevin Gaunt

Nachdem Tobias und ich bereits die Studienarbeit erfolgreich mit Herr Joller realisiert hatten, entschlossen wir uns dazu, uns wiederum mit einem eigenen Themenvorschlag zu bewerben. Schon vor Beginn dieser Arbeit war klar, dass der Fokus weg von der Entwicklung einer weiteren Webapplikation und vermehrt in Richtung der Informationsgewinnung aus Internet-Daten gehen soll. Durch einen Fachartikel über das semantische Netz wurde mein Interesse auf den Aufbau von Wissensnetzen gelenkt. Als sich sowohl meine Kommilitonen, wie auch unser Betreuer, bei einem ersten Gespräch dazu sehr interessiert zeigten und Herr Joller uns auf die Schwierigkeiten eines Projekts dieser Grössen- und Komplexitätsordnung aufmerksam machte, war mein Ehrgeiz definitiv geweckt.

Die erste Phase des Projektes war, den Anforderungen und dem fehlenden Grundwissen entsprechend, wissenschaftlich geprägt. Parallel dazu empfanden wir es als wichtig, möglichst rasch einen Machbarkeits-Prototypen voranzutreiben, bei dem wir unsere ersten Clustering-Gehversuche unternehmen konnten. Gerade die Möglichkeit, auf ein neues Thema eingehen zu können und die Herausforderungen der Projektvision, stellten für mich einen grossen Produktivitätsboost dar. Natürlich birgt eine weitgehend selbständige Arbeit stets auch einige Klippen und Risiken, die wir aber während den 16 Wochen meist grosszügig umschiffen konnten.

Der Entschluss keine strikte Aufgabentrennung einzuführen, ermöglichte allen Teammitgliedern, die Arbeit an verschiedenen Komponenten des Endproduktes. Besonders bei der Entwicklung der Architektur der Persistenzschicht über Neo4j und der Architektur der Visualisierung mittels Processing, konnte ich meine Stärken ausspielen und gleichzeitig in anderen Gebieten von den Stärken anderer profitieren.

Die Zusammenarbeit im Team verlief äusserst zufriedenstellend. Mit kurzen, prägnanten Sitzungen wurde der Arbeitstag begonnen und die Pendenzen oder Probleme diskutiert. Bemerkenswert ist auch, dass wir keine wirkliche Projektleitung bestimmten und trotzdem innerhalb der Dreiergruppe auf das gleiche Ziel zu arbeiteten. Abgesehen von kleineren, situationsbedingten Fluktuationen beim Arbeitsaufwand während der individuellen Freizeit, gab es keine nennenswerten Unterschiede beim Arbeitseinsatz. Die Unterstützung des Dozenten entsprach jener während der Studienarbeit, wobei besonders die motivierende Sitzung bei der Themenfindung hervorzuheben ist. Ausserdem war es sehr angenehm, das Vertrauen zur kompetenten und selbständigen Realisation der Arbeit zu geniessen.

Die Bachelorarbeit ermöglichte einen Einblick in viele verschiedene und vor allem auch neuartige Bereiche der Informatikwissenschaft. Beispielsweise konnte ich bei der Konzeption des verteilten Sys-

tems, auf das, an der HSR vermittelte Wissen im Modul 'Verteilte Softwaresysteme' zurückgreifen. Bei anderen Themen, wie etwa beim Data-Mining oder beim Clustering, waren meist Recherchen notwendig, welche auch im Bezug auf meine zukünftige berufliche Laufbahn, ganz neue Welten eröffneten.

Am Ende des Projektes bleiben die vielen Erfahrungen und das Wissen, ein grösseres und herausforderndes Projekt selbständig und erfolgreich abgewickelt zu haben. Auch die potenziellen Weiterentwicklungen versprechen einiges an interessanter Arbeit.

19.2 Tobias Löffel

Bereits bei meiner Studienarbeit 'Ziirp', die ich mit Kevin Gaunt realisiert habe, haben wir ein selbst eingebrachtes Thema bearbeitet. So entstand nach Abschluss der Studienarbeit auch der Wunsch, ein neues Projekt in einem ähnlichen Themengebiet, zum Thema unserer Bachelorarbeit zu machen. Unser Betreuer, Herr Joller liess uns von Anfang an freie Hand, ein eigenes Thema auszuarbeiten. Zuerst war die Idee, die Resultate unserer Studienarbeit weiter zu bearbeiten und auf dem Gebiet der Kontextanalyse eine Bachelorarbeit zu absolvieren. In einem einwöchigen Workshop zu Beginn der Arbeit, haben wir intensiv diskutiert, verschiedenste Ideen geprüft und wieder verworfen, bis wir dann auf Semantische Netze und deren automatische Generierung gestossen sind. Von Anfang an waren wir völlig fasziniert von diesem Thema und haben sofort damit begonnen, uns mit Büchern in die Materie einzulesen.

Da die Arbeit als Fortsetzungsarbeit unserer Studienarbeit gedacht war, wurde sie für ein Zweierteam ausgeschrieben. Der Studienarbeitspartner von Beat Weber studidaherert jedoch noch ein Jahr länger und so kamen wir auf die Idee, eine Dreiergruppe zu gründen. Dies bot uns die Möglichkeit, ein vom Umfang her grösseres Projekt zu bearbeiten, als dies als Zweiergruppe möglich gewesen wäre. Eine grössere Arbeit in einem Dreierteam zu absolvieren, heisst auch, die Arbeiten so zu koordinieren, dass möglichst klare Schnittstellen zwischen den Projektpartnern definiert werden und sich der administrative Aufwand möglichst in Grenzen hält. Dieser Balanceakt zwischen zuviel und zuwenig Kommunikation ist uns sehr gut gelungen. Dies kommt auch daher, dass ich meine beiden Partner schon seit Studienbeginn kenne und wir auch in unserer Freizeit Dinge zusammen unternehmen.

Während der ganzen Arbeit gewährte uns Herr Joller grösste Freiheiten. So konnten wir unsere Kreativität und Ideen voll ausnutzen. Viel Freiheit kann jedoch schnell zuviel Freiheit bedeuten. Im Projekt gab es immer wieder kurze Phasen, in denen wir andere Studenten beneideten, die eine klare Aufgabenstellung zu bearbeiten hatten und immer genau wussten, wo der Weg hingehen würde. Wir haben für uns zwar ebenfalls Ziele und Meilensteine definiert, waren jedoch so frei, dass wir die Ziele jederzeit ändern konnten, was zu relativ viel Mehraufwand führte, den wir rückblickend jedoch gerne auf uns genommen haben. Ich denke, mit einer starren Vorgabe, wäre unser Projekt nicht möglich gewesen und nie solch ein Resultat entstanden, wie wir es nun vor uns liegen haben. Zum guten Gelingen hat auch beigetragen, dass wir jeden Arbeitsschritt sehr kritisch hinterfragt haben. Dazu haben wir uns mindestens einmal pro Woche zu einer Sitzung getroffen, bei der alle berichteten, an was sie gerade arbeiten und wo die Probleme liegen. So konnten in der Diskussion Lösungen für viele Probleme gefunden werden. Ebenfalls sehr sinnvoll war das Codereview, das wir in der Mitte des Projektes durchführten, um anschliessend den Code gemeinsam zu refactoren. So bekam jeder einen Einblick in die Arbeit der Anderen.

Ich blicke auf ein sehr spannendes Semester zurück, das sehr lehrreich und wertvoll für meine weitere berufliche Zukunft war. Ich bin stolz, auf das, was wir in den vergangenen 16 Wochen erreicht haben und möchte keine der gemachten Erfahrungen missen.

19.3 Beat Weber

Die Bachelorarbeit startete mit einem einwöchigen Thema-Workshop, um unsere Idee genau zu spezifizieren. Die Vision war ein System zu entwickeln, das in der Lage ist, anhand beliebiger Informationsquellen im Internet Begriffe zu kategorisieren und die Begriffe mit weiteren verwandten Begriffen zu verknüpfen.

Aber wie soll ein System beispielsweise erkennen, dass die Schweiz ein Land ist, dieses Land in Europa liegt und Städte wie Bern, Basel und Zürich zur Schweiz gehören?

Eine Aufgabe, die für einen Menschen kein Problem darstellt, soll durch eine Software gelöst werden. Ein hochgestecktes Ziel, in Anbetracht dessen, dass dieses Gebiet für uns alle Neuland bedeutete. Ungewiss war dabei, was für Techniken einzusetzen sind, wie man diese anwendet und ob sie wirklich für diese Problemstellung funktionieren. Zudem nahmen wir uns vor, die einzelnen Komponenten des Systems zur besseren Skalierbarkeit, verteilt zu betreiben.

Im Verlauf der Arbeit lieferte die Analyse immer häufiger brauchbare Resultate. Die besten Verfahren und Techniken kristallisierten sich langsam heraus. Durch kontinuierliche Verbesserung und Verfeinerung der Analyse und Gruppierungs-Methoden, deckten sich die Kategorisierungen unseres Systems zu über 95% mit dem erwarteten Resultat. Eine Erfolgsquote, die wir zu Beginn nie erwartet hätten.

In der Planung und Durchführung des Projekts gewährte uns unser Betreuer maximalen Freiraum. Unserer Kreativität waren dadurch keinerlei Grenzen gesetzt. Dies macht die Arbeit jedoch nicht einfacher. Es ist aufwendiger seine Ziele und Termine selbst zu planen, als wenn man einfach nach einer strikten Anforderungsliste jeden einzelnen Punkt der Reihe nach abarbeiten muss. Dies erwies sich aber als sehr guter Lern- und Erfahrungspunkt.

Die Arbeiten im Team verliefen ausgesprochen gut. Obwohl jeder sehr selbständig an den jeweiligen Themen arbeitete, klappte die Koordination hervorragend. Jeder wusste immer um den Stand der Arbeiten der Anderen Bescheid und konnte bei Bedarf, Teile davon übernehmen. Technische Probleme konnten im Team schnell gelöst werden, da die Kommunikation untereinander gut funktionierte.

Da es sich bei unserem Projekt um eine eigene Idee handelt, merkte man, dass wir alle drei hoch motiviert waren. Es hat mir Spass gemacht, eine Lösung für ein spannendes, zu Beginn mir noch unbekanntes Themengebiet, zu entwickeln. Ich bin froh, dass wir uns für dieses Thema entschieden haben und ich mir dadurch ein breites Wissen in diversen neuen Bereichen erarbeiten konnte.

Kapitel 20

Glossar

Agent Komponente, des in dieser Arbeit entwickelten verteilten Systems. Zerlegt Begriffserklärungen von definierten Datenquellen in einen Term-Vektor.

Analyzer Komponente, des in dieser Arbeit entwickelten verteilten Systems. Analysiert Term-Vektoren der einzelnen Begriffe und weist sie verwandten Gruppen zu.

Crawler Ein Crawler ist ein Programm, welches das Internet durchsucht und Webseiten analysiert. Webcrawler werden vor allem von Suchmaschinen eingesetzt. Weitere Anwendungen sind das Sammeln von RSS-Newsfeeds, E-Mail-Adressen oder anderer Informationen.

Deterministisch Ein deterministischer Algorithmus ist ein Algorithmus, bei dem nur definierte und reproduzierbare Zustände auftreten. Ein Ablauf führt unter den gleichen Voraussetzungen, immer zum selben Resultat.

Exactly-once Begriff aus der Fehlersemantik. Ein Request wird im Fehlerfall noch einmal verschickt, Duplikate werden gefiltert.

HTML-Parser Ein Parser ist ein Programm, das für die Zerlegung und Umwandlung einer beliebigen Eingabe in ein, für die Weiterverarbeitung brauchbares Format, zuständig ist. Häufig werden Parser eingesetzt, um im Anschluss an den Analysevorgang, die Semantik der Eingabe zu erschließen und daraufhin Aktionen durchzuführen.

Information Extraction Unter Informationsextraktion, versteht man die Anwendung von Verfahren aus der praktischen Informatik, der künstlichen Intelligenz und der Computerlinguistik, auf das Problem der automatischen, maschinellen Verarbeitung von unstrukturierten Informationen mit dem Ziel, Wissen zu gewinnen.

JMS Java Message Service (JMS) ist eine, durch den Java Community Process genormte Programmierschnittstelle (API), für die Ansteuerung von Message Oriented Middleware aus einem Client heraus, der in der Programmiersprache Java geschrieben ist.

Master Komponente, des in dieser Arbeit entwickelten verteilten Systems. Zuständig für die Auftragsverteilung an die Agenten.

Ontologie Unter Ontologie versteht man eine explizite formale Spezifikation einer Konzeptualisierung.

Semantische Informationen Semantik, auch Bedeutungslehre, nennt man die Theorie oder Wissenschaft von der Bedeutung der (sprachlichen) Zeichen.

Semantisches Netz Ein semantisches Netz, ist ein formales Modell von Begriffen und ihren Beziehungen (Relationen). Es wird in der Informatik und der künstlichen Intelligenz zur Wissensrepräsentation genutzt. Gelegentlich spricht man auch von einem Wissensnetz. Meist wird ein semantisches Netz durch einen verallgemeinerten Graphen repräsentiert. Die Knoten des Graphen stellen dabei die Begriffe dar.

Single point of failure Unter einem Single point of failure versteht man einen Bestandteil eines technischen Systems, dessen Ausfall den Ausfall des gesamten Systems nach sich zieht.

Term Ein Wort, Ausdruck, Begriff oder auch eine Bezeichnung.

Term-Vektor Ein Vektor aus Wörtern mit dazugehöriger Wertigkeit, der einen Term Beschreibt.

TermNode Speicherform eines Terms in der Datenbank.

Text Mining Der Begriff Text Mining bezeichnet die automatisierte Entdeckung relevanter Informationen aus Textdaten. Mit statistischen und linguistischen Mitteln erschliesst Text-Mining-Software aus Texten Informationen, die die Benutzer in die Lage versetzen soll, ihr Wissen zu erweitern.

Well known ports Bestimmte Applikationen verwenden Portnummern, die fest zugeordnet und allgemein bekannt sind. Sie liegen üblicherweise von 0 bis 1023 und werden als well known ports bezeichnet

Quelle für das Glossar: `http://de.wikipedia.org/`

Kapitel 21

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] G. Heyer, U. Quasthoffe, T. Wittig, "Text Mining: Wissensrohstoff Text - Konzepte, Algorithmen, Ergebnisse", W3L, März, 2006

[2] Ott T., Christen M., Stoop R., "An Unbiased Clustering Algorithm Based on Self-organisation processes in Spiking Neural Networks. Proceedings of NDES'06", p. 143-146, 2006 (`http://www.ini.uzh.ch/~tott/ndes06tott.pdf`)

[3] Satnam Alag, "Collective Intelligence in Action", Manning, Oktober, 2008

[4] Schwaber Ken, Beedle Mike, "Agile Software Development with Scrum", Prentice Hall, Februar, 2002

[5] Grigoris Antoniou, Frank Van Harmelen, "A Semantic Web Primer", Mit Press, 31. März 2008

[6] Daniel T. Larose, "Discovering Knowledge in Data", Wiley & Sons, 14. Dezember 2004

[7] Zdravko Markov , Daniel T. Larose, "Data Mining the Web", Wiley & Sons, 25. April 2007

Kapitel 22

Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selbst und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.

Rapperswil, 12. Juni 2009

Kevin Gaunt Tobias Löffel Beat Weber