Experimente mit Semantischer Suche

Roland Schilling

29. August 2011

Inhaltsverzeichnis

Liste der Abkürzungen

ABox Assertion Box(Repräsentation der Individuen und ihrer Zustände innerhalb einer Ontologie)
AEO Athletics Event Ontology(BOEMIE Ontologie)
API Application Programming Interface(Ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird)
DL Description Logic (Beschreibungslogik)
GIO Geographic Information Ontology(BOEMIE Ontologie)
GPS Global Positioning System
GUI Graphical User Interface
ID Identification(Eindeutiger Identifikator innerhalb einer Datenbank)
IDE Integrated Development Environment
KB Knowledge Base
MCO Multimedia Content Ontology(BOEMIE Ontologie)
URL Uniform Resource Locator
TBox Terminological Box(Konzepte und Individuen innerhalb einer Ontologie)

1 Einleitung

In dieser Arbeit sollen Experimente mit einer Suchmaschine durchgeführt werden, die ihren Index nicht durch reine Indizierung eines Textes, sondern dazu durch Anreicherung mit semantischen Informationen generiert. Für diese Informationen wird auf eine vorhandene Datenbank aus Hintergrundwissen und eine festgelegte Menge an Dokumenten zurückgegriffen. Beides entstammt einem separaten Projekt (1.3) und dient hier lediglich als Datenbasis. Durch die Einbindung einer *Reasoning Engine* (1.5) können diese Informationen für jedes Dokument aus der Dokumentenmenge gewonnen werden. Die Grundlage für diese Arbeit sind die Ergebnisse der Diplomarbeit von Torben Rebhan [Reb09]. In dieser Diplomarbeit wurde ein Programm zur Durchführung und Auswertung semantischer Suchen entwickelt, das hier erweitert und für neue Experimente genutzt werden soll.

1.1 Indexbasierte Suche

Große Suchmaschinen im Internet, die in Bruchteilen von Sekunden Milliarden Internetseiten durchsuchen und die (idealerweise) interessanten Ergebnisse präsentieren können, verdanken ihre Geschwindigkeit dem *invertierten Index*. Dieser Index ist eine große Tabelle, die jedem gesuchten Term eine sortierte Liste von Dokumenten-Identifikationsnummern zuordnet. Eine effiziente Suche in großen Dokumentenmengen wird hierdurch möglich, indem der Index anstelle der Dokumente durchsucht wird. Zum Erstellen dieses Index werden die einzelnen Dokumente, z.B. wortweise, in kleine Einheiten (*Tokens*) zerlegt und in einer großen Tabelle abgelegt. Bei einem sehr großen Dokumentenschatz kann diese Tabelle leicht mehrere 100.000 Spalten und Zeilen enthalten. Vorher können verschiedene Techniken zur Bearbeitung der Tokens angewendet werden. Beispielsweise kann man einzelne Wörter auf ihren Wortstamm reduzieren (*stemming*) und so die Toleranz bei der Suche erhöhen. Die Technik des Stemming wird für jede Sprache anders gehandhabt und basiert auf individuellen Regeln. In einem simplen Beispiel der englischen Sprache würde für die Ablage in einem Index das Wort `Highjumper` auf das Wort `Highjump`, also seine Grundform reduziert. Dies erhöht die Trefferwahrscheinlichkeit einer Suche erheblich. Ebenso werden, wiederum in Abhängigkeit der Sprache, sogenannte *Stop Words* aussortiert. Worte, die durch ihre Häufigkeit in dieser Sprache zu einer Verfälschung des Ergebnisses führen würden.

1.1.1 Vektorraum-Retrieval

Vektorraum-Retrieval wird ein Suchverfahren genannt, bei dem die Menge der Tokens Punkte in einem metrischen Vektorraum repräsentiert. Jedes Dokument entspricht einem Vektor in diesem Raum, der überwiegend Nullen enthält. Ist ein Token in einem Dokument enthalten, hat der entsprechende Vektor an dieser Stelle einen Wert. Gesucht wird, indem eine mathematische Distanz zwischen dem Suchvektor und jedem Dokumentenvektor berechnet wird. Vereinfacht könnte man nach Vektoren suchen, die parallel zu einander verlaufen oder um einen kleine Winkel voneinander abweichen. Dies birgt den

zusätzlichen Vorteil, dass nach dieser Auswertung mit der Distanz zwischen Such- und Dokumentvektor eine Aussage über die Ähnlichkeit vorliegt, der sogenannte *Score*.

Stark vereinfacht kann folgendes Beispiel dem Verständnis dienen. In diesem Fall ergibt sich der Wert des Dokumentvektors an der Stelle x durch Summierung des entsprechenden Tokens im Dokument. Der Vektor für das Dokument `Die Katze jagt die Maus!` könnte dann als Vektor $(0, \ldots, 2, \ldots, 1, \ldots, 1, \ldots, 1, \ldots)$ repräsentiert werden. Das Wort `die` kommt zweimal vor, die Wörter `Katze`, `jagt` und `Maus` ein mal und sonstige Wörter null mal. Suchanfragen werden genau so behandelt. Eine Suche nach `Jagt die Katze die Maus?` ergibt also den gleichen Vektor und daher eine hunderprozentige Ähnlichkeit zwischen Dokument und Anfrage.

Die Tokens des Sprachraums werden in reeller Anwendung entsprechend ihrer Häufigkeit und in Abhängigkeit der Sprache und eines eventuell vorhandenen Gesamtkontexts gewichtet. Worte wie `der` oder `und` kommen in der deutschen Sprache sehr häufig vor und werden daher schwach gewichtet. Ein Wort wie z.B. `Wählscheibe` kommt deutlich seltener vor (insbesondere heutzutage) und wird höher gewichtet. Hierdurch wird verhindert, dass häufig auftretende Wörter das Ergebnis zu stark beeinflussen. Bevor mit der Indizierung der Dokumente begonnen wird, muss natürlich der Vektorraum definiert werden. Hierzu wird die Menge der Dokumente auf ihre Tokens reduziert. Jeder individuelle Token spannt hier eine neue Dimension auf und enthält eine Gewichtung entsprechend seiner Häufigkeit in Sprache oder Dokumentmenge.

1.2 Semantische Suche

Semantik, auch Bedeutungslehre, nennt man die Theorie oder Wissenschaft von der Bedeutung der Zeichen. „Zeichen" können in diesem Fall Wörter, Phrasen oder Symbole sein. Die Semantik beschäftigt sich typischerweise mit den Beziehungen zwischen Zeichen und Bedeutungen dieser Zeichen [Wik11c]. Unter dem Begriff *semantische Suche* versteht man die Suche unter zuhilfenahme semantischer Informationen über den eigentlichen Inhalt. Als *Zeichen* gelten im Zusammenhang mit der Suche einzelne Wörter in einem Dokument. Sie können miteinander in Beziehung stehen oder Metainformationen[1] enthalten, ohne dass dies aus dem Text hervor geht. Zum Beispiel kann ein Mensch, der in einem Text den Namen eines ihm bekannten Schauspielers liest, Auskunft über den Beruf dieser Person geben, ohne dass diese Information im Text enthalten ist. Hierzu greift er auf Hintergrundwissen zurück, das er im Laufe seines Lebens gesammelt hat. Ähnlich verhält es sich bei der semantischen Suche. Das Hintergrundwissen liegt hierbei in Form einer Datenbank (KB[2]) vor[3] und muss bei der Generierung des Index ausgewertet werden (siehe 1.4 auf Seite 8). Dieser Schritt wird *Extraktion* genannt. Die Extraktion erfolgt in mehreren Schritten:

[1] Als Metadaten oder Metainformationen bezeichnet man allgemein Daten, die Informationen über andere Daten enthalten. Bei den beschriebenen Daten handelt es sich oft um größere Datensammlungen (Dokumente) wie Bücher, Datenbanken oder Dateien. [Wik11a]

[2] Knowledge Base

[3] KBs werden durch Menschenhand angelegt, da die enthaltenen Informationen und Zusammenhänge nicht maschinell auswertbar sind.

- Auswertung des Dokuments hinsichtlich vorhandener Individuen in der KB
- Auswertung der Informationen zu diesen Individuen
- Hinzufügen dieser Informationen zum Dokument (*Anreicherung*)

Das Dokument enthält nun auch alle identifizierten semantischen Informationen, die somit in die Generierung des Index eingehen. Das Wort *Individuum* ist in diesem Fall nicht mit *Person* gleichzusetzen sondern beschreibt ein Ding (engl. Thing), über das die KB weiter Informationen enthält. Diese Zuordnung ist Teil der KB und die Grundlage für die weitere Auswertung.

Betrachten wir als Beispiel eine Internetseite mit einer Nachrichtenmeldung über den Tennisspieler Andre Agassi:

```
He will make his return to competitive tournament
tennis by participating in the 30-and-over Outback
Champions Series event in October at Surprise,Arizona.

Tour spokesman Randy Walker told The Associated
Press on Sunday about Agassi's plans. The tour will
make an announcement Monday.

Agassi,who turns 39 next month,retired after playing in
the 2006 U.S. Open. He walked away from the game
with eight major singles championships and remains
one of only five men to complete a career Grand Slam.

He's the first announced member of what will be an
eight-player field competing for $150,000 in prize money
at the Oct. 8-11 tournament at the Surprise Tennis and
Racquet Complex.

To be eligible for the Outback Champions Series,players
must have reached at least one final at a Grand Slam
tournament,been ranked in the top five or played
singles on a Davis Cup title-winning team. Agassi meets
all three criteria,including having logged 101 weeks
with the ATP's No. 1 ranking,and having helped the
United States claim three Davis Cup championships.

Among others who have played on the senior tour this
year are Pete Sampras and John McEnroe.

Agassi's participation marks another in a series of
returns to the court for him.
```

```
World Team Tennis announced last month he will play in
two matches for the Philadelphia Freedoms in July.

And Agassi also plans to join his wife,Steffi Graf,along
with former touring pros Tim Henman and Kim Clijsters
for exhibition matches at the All England Club to test the
new retractable roof over Centre Court on May 17,five
weeks before Wimbledon starts.
```

In diesem Text finden sich einige Informationen, die ohne Kenntnisse der Semantik der Sprache nicht erkennbar sind. Zum Beispiel das Alter von Andre Agassi. Da dieses jedoch nur implizit erwähnt ist, also ohne das Wort *age*, wird eine Suche in einem rein textbasierten Index nach `+agassi +age` dieses Dokument nicht zurückliefern. Betrachten wir nun, aus einer KB gewonnenen zusätzlichen Informationen zu diesem Dokument:

```
Age PersonName Ranking MLC Country GeographicObject GIO_Thing
Name GeopoliticalArea Date SportsTrial AEO_Thing Gender SportsEvent
Performance HLC SportsEventName Person Male
```

Hier fällt auf, dass die Angabe des Alters richtig interpretiert und `Age` zum Index hinzugefügt wurde. Eine erneute Suche nach `+agassi +age` liefert nun auch das Beispieldokument zurück.

1.3 Das BOEMIE Projekt

BOEMIE ist ein von der EU gefördertes Projekt mit dem Ziel, automatisiert Wissen aus Multimedia-Inhalten zu gewinnen. Die Grundlage hierfür ist eine Sammlung von HTML-Seiten, Bildern und Videos, die sich mit dem Thema Leichtathletik beschäftigen. Das Hintergrundwissen besteht aus drei Ontologien[4]:

- Leichtathletik (AEO[5])
- Multimedia (MCO[6])
- Geographie (GIO[7])

Jede dieser Ontologien enthält das Wissen über die entsprechende Domäne. Alle Konzepte sind abgeleitet von einem definierten Grundkonzept `Thing`, wodurch sich eine Konzepthierarchie ergibt, die Teil dieser Arbeit wird.

[4] Ontologien in der Informatik sind meist sprachlich gefasste und formal geordnete Darstellungen einer Menge von Begrifflichkeiten und der zwischen ihnen bestehenden Beziehungen in einem bestimmten Gegenstandsbereich. Sie werden dazu genutzt, „Wissen“ in digitalisierter und formaler Form zwischen Anwendungsprogrammen und Diensten auszutauschen.[Wik11b]

[5] Athletics Event Ontology

[6] Multimedia Content Ontology

[7] Geographic Information Ontology

1.4 Beschreibungslogik und Ontologien

Die Beschreibungslogik (DL[8]) ist eine Familie von formalen Sprachen zur Repräsentation von Wissen. Ihre Grundlagen sind Konzepte, Rollen und Individuen. Konzepte und Rollen bilden zusammen ein Modell einer Domäne (die Terminologie dieser Domäne), genannt TBox[9]. Die Menge der Individuen und ihrer Beziehungen beschreibt den Zustand der modellierten Domäne und wird ABox[10] genannt. Individuen können über Rollen bestimmte Werte zugewiesen werden. Beispielsweise kann einem Individuum vom Typ *SportsEvent* über die Rolle *hasSportsEventName* ein Name zugewiesen werden. Eine TBox und die zugehörige ABox bilden zusammen eine KB (Abbildung 1). Es gibt verschiedene Arten von DLs, die mit ansteigender Ausdrucksmächtigkeit aufeinander aufbauen:

$\mathcal{AL}$ Attribute Language: Beinhaltet Negation von atomaren Konzepten, Konjunktion, Universalquantifikation und ungualifizierte Existensqualifikation.

$\mathcal{ALC}$ Mit der zusätzlichen Möglichkeit, komplexe Konzepte zu negieren.

$\mathcal{ALCQ}$ Mit der zusätzlichen Möglichkeit, zahlenmäßig einschränkbare Existenzquantoren zu verwenden.

$\mathcal{ALCQ}(\mathcal{D})$ Mit der zusätzlichen Möglichkeit, konkrete Domänen zu verwenden.

Für diese Experimente ist $\mathcal{ALCQ}(\mathcal{D})$ die einzig relevante. Sie beinhaltet atomare Konzepte, Konjunktion, Universalquantifikation, unqualifizierte und qualifizierte Existenzqualifikation, Negation komplexer Konzepte und die Verwendung konkreter Domänen.

1.5 RacerPro

RacerPro steht für ***R**enamed **A**box **C**oncept **E**xpression **R**easoner **Pro**fessional*. Es ist, wie der Name schon sagt ein Beschreibungslogiksystem, also ein Werkzeug zur Interpretation von Ontologien und implementiert $\mathcal{ALCQ}(\mathcal{D})$. RacerPro ist fähig, die Informationen einer KB auszuwerten und Anfragen nach den benötigten Daten zu beantworten. Hierzu bietet es eine Anfragesprache, die genutzt werden kann um semantische Informationen zu den jeweils betrachteten Dokumenten zu gewinnen.

1.6 Apache Lucene

Lucene ist eine hochperformante volltext Suchmaschine mit großem Funktionsumfang. Sie ist komplett in Java geschrieben und kann in Webseiten und Programme eingebunden werden. Ihr Kern ist der *Index*. Ein Index wird über eine Menge von Dokumenten generiert indem der Volltext jedes einzelnen Dokuments ausgewertet wird. In diesem Prozess durchläuft dieser Text eine Reihe von Prozessoren, z.B. für das *Stemming* oder das Aussortieren von *Stopwords* (Wörter, die in einer Sprache so häufig vorkommen, dass sie

[8] Description Logic
[9] Terminological Box
[10] Assertion Box

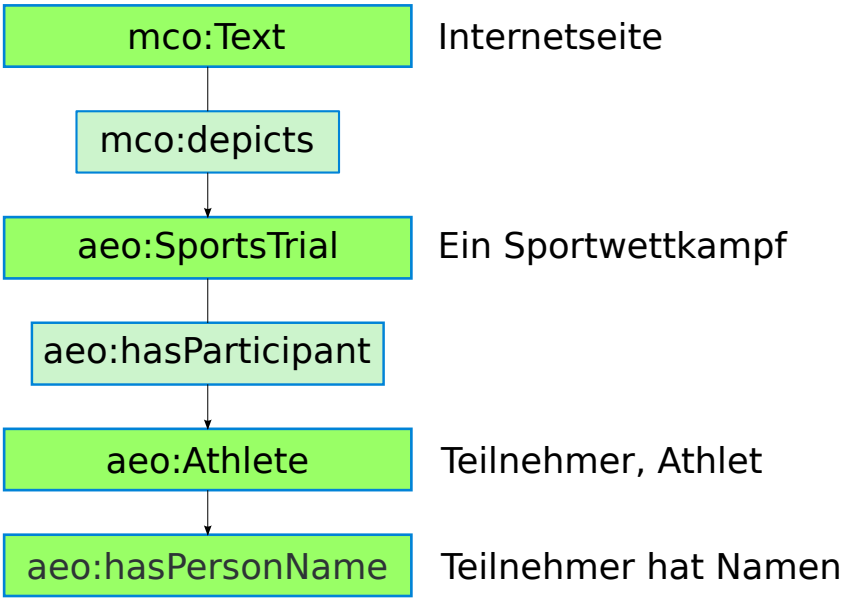

Abbildung 1: Beispiel einer KB (Auszug). In einer betrachteten Internetseite ist ein Sportwettkampf beschrieben, in dem es einen Teilnehmer gibt, der Athlet ist und einen Namen hat. Die Präfixe *mco* und *aeo* geben Auskunft über die Ontologie, in der das jeweilige Konzept beschrieben ist.

für den Index ignoriert werden). Nach dieser Indizierung wird für jedes Dokument eine Repräsentation in Form eines Objekts der Klasse *Document* angelegt und mit *Feldern* gefüllt. Ein Feld ist ein Tupel aus Name und Textwert und kann mit dem Index abgespeichert werden. Felder dienen sowohl dazu, ein Dokument eindeutig zu identifizieren, als auch Daten für das Suchergebnis zurückzuliefern. So können das Datum der Indizierung oder der Ort des Dokuments als nicht-suchbare Felder im Index mit abgelegt und als Teil des Suchergebnisses zurückgegeben werden.

1.7 SemTexSearch

Das Programm zur Auswertung semantischer Indexe, dass in [Reb09] erstellt wurde heißt *SemTexSearch*. Es verwendet im Kern Apache Lucene zur Indexierung und Suche und RacerPro zur semantischen Anreicherung unter Verwendung der BOEMIE KBs. Es besteht aus Bibliothek und Interface und kann sowohl Indexe generieren, als auch in diesen Suchen und einfache Statistiken über *Score*, Relevanz und *Precision* wiedergeben. Außerdem dient es dazu Dokumente über Mapping Dateien mit ihren KBs zu verknüpfen. Suchanfragen und -ergebnisse können in einem XML Format abgespeichert werden.

2 Zielsetzung

Als Basis für die geplanten Experimente diente das Programm *SemTexSearch* von Torben Rebhan [Reb09]. SemTexSearch ist eine prototypische Software, die innerhalb eines Repositoriums von HTML Dateien und ihrer zugehörigen KBs einen Index generieren und in diesem suchen kann. Grundlegende semantische Informationen können bereits ausgewertet werden und es wurden einige Experimente hinsichtlich der veränderten Ergebnisanzahl und der Zahl der relevanten Treffer durchgeführt. Für diese Arbeit sollen nun weitere Experimente durchgeführt und evaluiert werden. *SemTexSearch* lag für diese Arbeit in Quellcodeform vor und konnte für die folgenden Experimente angepasst werden.

2.1 Vorarbeit

Bevor mit der eigentlichen Arbeit begonnen werden konnte, mussten der Code des Programms *SemTexSearch* in Betrieb genommen und die originalen Experimente aus [Reb09] nachvollzogen werden. Obwohl der Quellcode vollständig zur Verfügung stand, erwies sich dieser Teil der Arbeit als unerwartet aufwändig. Seit der Durchführung der Experimente in [Reb09] haben sich die genutzten Bibliotheken, die dem Quellcode nicht beilagen, zum Teil stark verändert. Anpassungen an die neuen APIs[11] endeten in verändertem Programmverhalten, wodurch ein Nachvollziehen der ursprünglichen Experimente unmöglich wurde. Nach Rücksprache mit Herrn Rebhan und einigen Tests, wurden die folgenden Bibiotheken in genannter Version als funktionierende Konfiguration erkannt und für die weiteren Experimente genutzt:

- SemTexSearch
  - xerxesImpl (2.9.1)
  - nekohtml (1.9.9)
  - lucene-core (2.4.0)
  - lucene-snowball (2.4.0)
  - log4j (1.2.*)
- SemTexSearchGui
  - swing-layout (1.0.3)
  - swing-worker (1.1)
  - appframework (1.0.3)

Darüberhinaus wurde in [Reb09] die IDE[12] *Netbeans* als Entwicklungsumgebung genutzt, was Auswirkungen auf die Weiterentwicklung der GUI[13] Elemente hat. Für dieses Experiment wurde im Vorfeld *Eclipse* als Umgebung gewählt, welches keinen internen

[11] Application Programming Interfaces
[12] Integrated Development Environment
[13] Graphical User Interface

GUI Editor zur Verfügung stellt. Der native GUI-Designer von Netbeans erzeugt Metacode in einem XML Format, der von keinem der für Eclipse verfügbaren Plugins interpretiert werden kann. Zwar ist eine Portierung des Netbeans Editor für Eclipse in Arbeit, diese ist aber noch nicht weit genug fortgeschritten, um für dieses Projekt zum Einsatz zu kommen. Daher wurde sich kurzfristig auf eine Kombination aus Netbeans und Eclipse für die weitere Arbeit festgelegt. Netbeans kam hierbei nur zum GUI-Design zum Einsatz, Eclipse wurde zur eigentlichen Entwicklung genutzt.

Die genutzte Reasoning Engine ist das Program *RacerPro*[14]

2.2 Versuchsaufbau

Ziel dieser Arbeit ist es, in einer vorhandenen Umgebung eine Reihe von Tests unter wechselnden Bedingungen durchzuführen und die Ergebnisse zu vergleichen. Da sich die Ergebnisse leicht nachvollziehen lassen und auf sie aufgebaut werden soll, ist eine automatisierte Abwicklung der Such- und Speicheroperationen innerhalb der Software sinnvoll. Zu diesem Zweck wurde das in [Reb09] entwickelte Programm *SemTexSearch* um ein Batch-Modul erweitert, das die automatische Durchführung einer Reihe von Suchen in kürzester Zeit ermöglicht. Dieses Modul arbeitet eine Datei ab, die in jeder Zeile eine Suchanfrage enthält. Jede Anfrage wird auf dem aktuell geladenen Index durchgeführt und das Ergebnis in einem vorgegebenen Ordner gespeichert. Zusätzlich bietet es zwei Schnittstellen, sogenannte *Hooks*. Diese Hooks stellen Punkte im batch Ablauf dar, an denen anderer Code eingefügt werden kann. So ist es z.B. leicht möglich Statistiken zu anzulegen oder Zeitmessungen durchzuführen, ohne den vorhandenen Code umständlich erweitern oder verändern zu müssen.

2.3 Zuverlässiger Code für wissenschaftliche Experimente

Das Herzstück in *SemTexSearch* ist die Klasse *SemanticAdder*. In dieser Klasse findet die gesamte Anreicherung der reinen Dokumentinformationen mit semantischen Informationen aus den BOEMIE KBs statt. Es werden die Individuen des jeweiligen Dokuments extrahiert, ihre Namen erfasst und von diesen Individuen aus weitere Informationen gewonnen. Für Experimente unter wechselnden Bedingungen ist daher eine Architektur von Vorteil, die auf robuste Weise veränderte Implementierungen unterstützt und bestenfalls Snapshots[15] verschiedener Konfigurationen ermöglicht. Daher wurde die anfänglich stark monolithische Struktur dieser Klasse verändert und unter Anwendung des *Decorator*- und des *Factory* Pattern neu strukturiert.

2.3.1 Das Decorator Pattern

Das *Decorator Pattern* [Fre+04] abstrahiert die Eigenschaften einer Klasse und kapselt diese in kleine, separate Klassen. Alle diese Verhaltensklassen implementieren das glei-

[14] Renamed ABox and Concept Expression Reasoner [KG11]

[15] Snapshot, eine Momentaufnahme. In diesem Zusammenhang beschreibt ein Snapshot einen Konfigurationszustand zu einem bestimmten Zeitpunkt. Z.B. die Konfiguration zur Erstellung eines bestimmten Index

che Interface und (in diesem speziellen Fall) erben von der gleichen abstrakten Parent-Klasse. Die Verhaltensklassen sind so entworfen, dass sie zur Instanzierung ein Objekt mit dem gleichen Inteface erwarten. Ihre worker-Methode ruft dann zur Laufzeit die worker-Methode des übergebenen Objekts auf und es entsteht eine Kette kleiner Operationen. Die Felder der inneren Klassen stehen dabei den äußeren zur Verfügung, wodurch eine Kette aus Operationen entsteht, die auf einfache Art erweitert oder in der Reihenfolge verändert werden kann. Das äußerste Objekt wird in dieser Kette als letztes ausgeführt. Eine einfache boolean Variable in jeder dieser Klassen gibt Auskunft über den Erfolg der eigenen `process()`-Methode, sodass die gesamte Kette abgebrochen werden kann, wenn z.B. für ein Dokument keine KB zur Verfügung steht. Der Vorteil dieser Variante gegenüber z.B. dem *Strategy Pattern* [Fre+04] ist die definierbare Reihenfolge der einzelnen Prozessschritte. Sie kann (und muss) fest definiert und leicht variiert werden. Einzelne Schritte, die aufeinander aufbauen, müssen so nicht manuell synchronisiert, sondern können bequem zur Designzeit festgelegt werden.

Die ursprüngliche *SemanticAdder*-Klasse besteht essentiell aus einer großen Methode `process()`, die ihre gesamte Funktionalität beinhaltet. Um SemanticAdder einfach erweitern zu können, wurde diese Klasse wie beschrieben neu entworfen und implementiert.

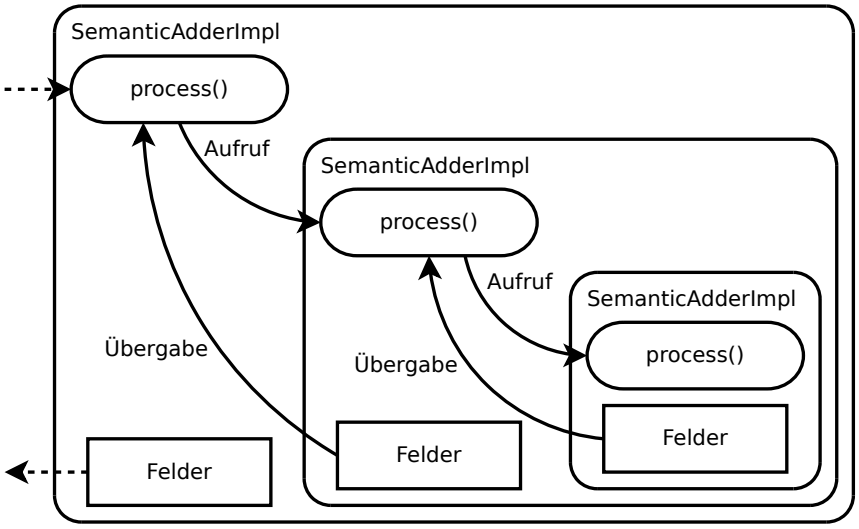

Abbildung 2: Beispielhafte Darstellung des Decorator Pattern am Beispiel der *SemanticAdder* Klasse. Die Methodenaufrufe werden vom äußersten zum innersten Objekt durchgereiht (chaining), während die Felder der inneren den äußeren Objekten für ihre Arbeit zur Verfügung stehen.

2.3.2 Das Factory Pattern

Das *Factory Pattern* [Fre+04] abstrahiert die Instanzierung von Objekten. Es wird umgesetzt durch eine einzelne Klasse, die Methoden für diese Instanzierung bereit hält. In

diesem Fall wird es durch dieses Pattern möglich, *SemanticAdder* Instanzen mit verschiedenen Konfigurationen zu definieren, die über jeweils eine einzelne Methode der *SemanticAdderFactory* Klasse erstellt werden. Durch die Modularität des neuen Designs für die *SemanticAdder*-Klasse gewinnt die Software hiermit an genug Flexibilität, verschiedene Experimente schnell und zuverlässig unter wechselnden Bedingungen durchzuführen. Die im Einzelfall genutzten Konfigurationen von *SemanticAdder* stehen dabei in einzelnen Methoden der Factory zur Verfügung und können für weitere Experimente erneut genutzt werden, ohne den Code selbst verändern zu müssen.

Listing 1: Beispiel für eine Factorymethode. Diese Methode erstellt die Kette von SemanticAdderImpl Objekten, die für die Erstellung von Index 5 genutzt wird.

```
1  public static SemanticAdderImpl
2  getIndex5SemanticAdder( RacerConnection connection,
3                          boolean textOnly,
4                          int depth,
5                          Properties owlFilesForDocuments) {
6      RootSemanticAdder
7      root = new RootSemanticAdder( connection,
8                                    textOnly,
9                                    depth,
10                                   owlFilesForDocuments );
11     IndividualSemanticAdder
12     ind = new IndividualSemanticAdder( connection,
13                                        textOnly,
14                                        depth,
15                                        owlFilesForDocuments,
16                                        root );
17     RelatedIndividualSemanticAdder
18     rel = new RelatedIndividualSemanticAdder( connection,
19                                               textOnly,
20                                               depth,
21                                               owlFilesForDocuments,
22                                               ind );
23     NamesSemanticAdder
24     names = new NamesSemanticAdder( connection,
25                                     textOnly,
26                                     depth,
27                                     owlFilesForDocuments,
28                                     rel );
29
30     return names;
31 }
```

3 Durchführung

3.1 Nachvollziehen der bekannten Ergebnisse

Nach erfolgreicher Inbetriebnahme und Erweiterung der Software um die genannten Funktionen bestand der erste Schritt aus dem Nachvollziehen der ursprünglichen Experimente. Die ersten Tests mit den Repositorien des BOEMIE[16] Projekts [BOE11] verdeutlichten, dass diese, zumindest in der vorliegenden Form, nicht die Basis für die Experimente in [Reb09] waren. Das BOEMIE Projekt verwendet für die Referenzierung der Dokumente innerhalb seiner Repositorien IDs[17], die vor der Verarbeitung ohne die BOEMIE-tools[18] in URLs[19] umgewandelt werden müssen. Herr Rebhan war so freundlich, für diese Experimente seine Originaldaten zu Verfügung zu stellen, die eine Teilmenge des BOEMIE HTML Repositoriums sind. Mit diese Daten war es möglich, die Ergebnisse aus [Reb09] vollständig nachzuvollziehen. Hierzu musste ein Fehler in der Dokumentation korrigiert werden, die Anfrage 12 lautete `+"javelin world record"~10 +women` anstatt `+"javelin world record" 10 +women`. Nach erfolgreicher Reproduktion der Ergebnisse aus [Reb09] konnte mit der Umsetzung neuer Experimente begonnen werden. Im Kern diese Arbeit geht es um ein Experiment in dem die semantischen Informationen durch Einbeziehung der Konzepte, die ein jeweiliges Individuum instanziert, angereichert werden. Diese Ergebnisse sollen betrachtet und bewertet werden.

3.2 Kriterien zur Bewertung von Ergebnissen

Nach erfolgter Suche und Auswertung der Relevanz einzelner Treffer, soll eine Messung der Qualität der Anfrageergebnisse durchgeführt werden. In der Fachliteratur gibt es dazu mehrere Metriken, von denen zwei hier zur Anwendung kommen. Für weitere Informationen empfielt sich die Lektüre von [Abr10] und [Reb09].

3.2.1 Precision und Average Precision

Die *Precision* ist eine Metrik zur Evaluierung der Qualität der erhaltenen Suchergebnisse. Sie betrachtet die Gesamtsumme der gefundenen Dokumente und und die Anzahl der relevanten Dokumente und stellt ein Verhältnis auf. Sei D_r die Anzahl der relevanten, und N die Anzahl aller Ergebnisse. Die Precision P ist dann definiert als:

$$P = \frac{D_r}{N} \tag{1}$$

P ist also nichts weiter als das Verhältnis von relevanten zu gefundenen Dokumenten. Daher hat P einen Wertebereich von $[0, 1]$. Sei weiterhin D_n die Anzahl aller relevanten

[16] Bootstrapping Ontology Evolution with Multimedia Information Extraction
[17] Identifications
[18] Das BOEMIE Projekt hat im Rahmen seiner Arbeit u.A. einen Semantischen Browser erstellt, mit dem man die angebotenen Daten durchsuchen kann. Dieser Browser nutzt das, in den BOEMIE Repositorien genutzte Indizierungsverfahren.
[19] Uniform Resource Locators

Dokumente im Index, dann ist der *Recall* definiert als:

$$R = \frac{D_r}{D_n} \tag{2}$$

Die *Average Precision* berücksichtigt im Gegensatz zur Precision die Reihenfolge in der die Ergebnisse zurückgegeben werden. Sei D_i die Anzahl relevanter Ergebnisse bis einschließlich Position i. Damit ist die relative Precision P_i definiert als:

$$P_i = \frac{D_i}{i} \tag{3}$$

Der relative Recall R_i ist definiert als:

$$R_i = \frac{D_i}{D_r} \tag{4}$$

Mit der relativen Precision und dem relativen Recall läst sich die *Pseudo-Precision* $\tilde{P}(x)$ bei Revall-Level x wie folgt definieren:

$$\tilde{P}(x) = max\, P_i,\ für\ x \leq \frac{D_i}{D_r}\ mit\ i = 1, 2, ..., n \tag{5}$$

Damit lässt sich die *Average Precision* wie folgt darstellen:

$$P_{av} = \frac{1}{n} \sum_{i=0}^{n-1} \tilde{P}\left(\frac{i}{n-1}\right) \tag{6}$$

Für diese Arbeit werden, analog zu [Reb09] die Precision und die Average Precision als Indikator für die Qualität der Suchergebnisse genutzt. Dabei ist zu beachten, dass die Relevanz eines Ergebnisses eine subjektive Größe ist. Es hängt vom Anwender ab, ob ein Ergebnis als relevant empfunden wird oder nicht. Daher wurde bei Erweiterung der Ergebnisse aus [Reb09] die gefundene Relevanz übernommen, um Inkonsistenzen zu vermeiden.

3.3 Einbeziehung der Individuenkonzepte

In den Experimenten in [Reb09] wurde folgendermaßen vorgegangen:

- Lade Internetseite und zugehörige Ontologie
- Extrahiere die von der Internetseite beschriebenen Individuen
- Finde alle Individuen, die mit den Gefundenen in Beziehung stehen
- Bestimme die Namen all dieser Individuen
- Füge die gewonnenen Daten zum Dokument hinzu
- Indiziere Dokument für die Suche

Die letzten beiden Schritte sind natürlich immer notwendig. Im Folgenden soll dieses Experiment nun erweitert werden. Zusätzlich zu den bereits gewonnenen Daten sollen auch die Konzepte mit einbezogen werden, von denen ein Individuum eine Instanz ist. Im Falle des Beispiels aus der Einleitung kann man so feststellen, dass Andre Agassi ein Athlet ist und dass es im Text um mehrere Wettkämpfe geht, die wiederum Teil einer Hierarchie sind. Im Anschluss sollen neue Suchterme gefunden werden, um diese neu gewonnenen Informationen nutzen zu können. Hierzu wird ein neuer Index auf Basis der Konfiguration von Index5 aus [Reb09] generiert. Dieser Index enthält semantische Informationen, nicht nur aus dem Text sondern auch aus Bildern und Videos gewonnen wurden, hat die Hierarchietiefe 2^{20} und enthält zusätzlich alle Konzepte, die von jedem einzelnen betrachteten Individuum instanziert werden. Zur Bestimmung dieser Konzepte wird der *RacerPro*-Befehl `(individual-types` *ind*`)` genutzt, der nicht nur direkte Instanzen betrachtet, sondern die gesamte Instanzenhierarchie. Die zusätzlichen Infor-

Index	Sem. Inf	Nur Text	Tiefe	Größe in MB	Konzeptinstanzen
4	√		1	2,8	
5	√		2	2,8	
6	√		2	3,4	√

Tabelle 1: Der neue Index Nr. 6 gleicht Index 5 bis auf die Addition der Konzeptinstanzen.

mationen, die mit diesem Verfahren gewonnen wurden werden im Folgenden aufgelistet. Ein Index besteht aus 606 Dokumenten, die jeweils mehrere Individuen abbilden. Für jedes dieser Individuen wurde `(individual-types` *ind*`)` aufgerufen, jedes Konzept aber pro Dokument nur einmal abgespeichert, wodurch die maximale Häufigkeit ebenfalls 606 beträgt.

Konzept	Häufigkeit
MLC	604
AEO_Thing	604
HLC	604
SportsTrial	602
Ranking	599
Person	595
PersonName	594
Name	594
Performance	590
SportsName	589
SportsCompetition	589
Date	583
GeographicObject	577
GIO_Thing	577
GeopoliticalArea	576

[20] Im in [Reb09] genutzten Breitensuchalgorithmus

Konzept	Häufigkeit
SportsEvent	574
SportsEventName	574
Country	573
Athlete	549
City	496
RunningCompetition	472
Gender	469
Female	409
Male	369
JumpingCompetition	311
Age	304
PersonBody	300
AnatomicalPartOfPerson	300
PersonFace	300
OrganismPart	300
Running100mName	298
Running100mCompetition	286
SprintCompetition	283
SportsRound	240
AthleticsTrial	233
SportsRoundName	231
ThrowingCompetition	223
Object	212
LongJumpName	211
LongJumpCompetition	211
HighJumpCompetition	208
HighJumpName	208
PoleVaultName	200
Jumping	197
PoleVaultCompetition	191
Stadium	187
Point	187
POI	187
SportPOI	187
GeometryObject	187
Jumper	183
RoadRunningCompetition	172
MarathonCompetition	172
LongDistanceRunningCompetition	172
MarathonName	172
HorizontalBar	167
Hurdling110mName	142
Hurdling110mCompetition	134

Konzept	Häufigkeit
JavelinThrowName	130
JavelinThrowCompetition	125
PoleVault	112
PoleVaulter	109
DiscusThrowName	108
DiscusThrowCompetition	105
HighJump	102
HammerThrowName	100
HighJumper	97
WalkingCompetition	97
HammerThrowCompetition	97
RunningRound	79
Walking20kmCompetition	76
Walking20kmName	76
Running100mRound	73
SprintRound	73
JumpingRound	54
Running	51
Walking10kmName	51
Walking10kmCompetition	51
Sprint	48
Running100m	48
Pillar	47
Walking50kmCompetition	40
Walking50kmName	40
Javelin	39
ThrowingRound	34
Pole	33
LongJumpRound	27
Throwing	24
Runner	16
LongJump	16
Sprinter	15
PoleVaultRound	15
HighJumpRound	15
Runner100m	15
JavelinThrowRound	14
JavelinThrow	12
HammerThrowRound	11
DiscusThrow	10
TripleJumpName	10
DiscusThrowRound	10
Thrower	10

Konzept	Häufigkeit
Discus	8
MaleHurdling110mCompetition	8
HurdlingCompetition	8
Hammer	7
LongJumper	7
JavelinThrower	6
MarathonRound	4
RoadRunningRound	4
LongDistanceRunningRound	4
TripleJump	3
HalfMarathonName	3
TripleJumpRound	3
HammerThrow	3
TripleJumpCompetition	3
DiscusThrower	3
LongDistanceRunning	2
Marathon	2
RoadRunning	2
HammerThrower	2
Walking	1
Walking50kmRound	1
MaleHurdling110mRound	1
Hurdling60mCompetition	1
WalkingRound	1
LongDistanceRunner	1
Hurdling60mName	1
HurdlingRound	1
Walking50km	1
RoadRunner	1

Tabelle 2: Neu gewonnene Konzepte in Index 6

Diese Ergebnisse werden allerdings in der Anwendung durch das Stemming etwas reduziert. So liefert z.B. eine Suche nach `HighJumper` nicht die trivialen 97 Ergebnisse, sondern 102, da *HighJumper* auf seinen Stamm *HighJump* reduziert wird. Alle Begriffe, die durch stemming auf einen anderen aus der Liste reduziert werden können, gehen also verloren.

3.4 Hypothese

Nach Betrachtung der Ergebnisse aus [Reb09] ist anzunehmen, dass sich die Anzahl der Suchergebnisse durch weitere Anreicherung steigern lässt. Höchstwahrscheinlich wird es insgesamt mehr neue als neue relevante Dokumente geben, was zu einer Verkleinerung der

Precision führen wird. Durch den größeren Tokenraum sollte sich auch eine Veränderung im Score der einzelnen Dokumente beobachten lassen. Hier kann es durch die Gewichtung einzelner Tokens durchaus eine Verbesserung geben. In den meisten Fällen sollte er jedoch leicht sinken. Ebenfalls, wie auch in [Reb09] angemerkt, kann sich der Nachrichtencharakter der zugrundeliegenden Dokumente negativ auswirken, da die wichtigen Informationen bereits im Text enthalten und auch Bilder mit einem Mehrwert an Information gut kommentiert sind. Trotzdem sollte sich bei geschickter Wahl der Anfragen eine Veränderung in den Ergebnissen erzielen lassen.

3.5 Naiver Ansatz und camel-case Filter

Auf Basis dieser Informationen können nun neue Suchterme generiert werden. Aufgrund der Allgemeinheit der Konzepte müssen diese weniger spezifisch ausfallen, als die in [Reb09] genutzten. Zuerst wird der naive Ansatz betrachtet und nach einigen Konzepten direkt gesucht. Außerdem betrachten wir einige natürliche Anfragen. Die Bewertung der Ergebnisse spielt hier zu Beginn keine Rolle, da nur beobachtet werden soll, wie sich die Ergebniszahlen verändern (Tabelle 3).

Term	Index 5	Index 6
HammerThrowCompetition	0	97
Stadium	201	241
Performance	388	590
Pole	225	233
SportsTrial	0	602
triplejump men +“olympic games“	154	154
hurdling +“olympic games“ +2004 +Athens +“world record“	69	69
+“dwain chambers“ +“world rankings“	1	1

Tabelle 3: Naiver Ansatz für die Suche in den neu gewonnenen Informationen

Anhand dieser Ergebnisse kann man zwar sehen, dass die Hinzunahme der Konzeptinformationen bei gezielter Suche neue Ergebnisse liefert, jedoch sind diese Beispiele stark akademisch und entsprechen kaum einem Suchterm in realer Anwendung. Insbesondere die camel-case[21] Schreibweise der Mehrwortterme entspricht nicht der natürlichen Art, diese Begriffe zu schreiben[22]. So kann man bei an den Ergebnissen der drei natürlichen Anfragen sehen, dass die neuen Informationen für diese Art der Suche nicht automatisch neue Ergebnisse liefern. Darüberhinaus verändert sich der Score[23] der einzelnen Ergebnisse durch die größere Tokenmenge in Index 6. So unterscheiden sich die Scores des

[21] Camel-case wird eine Schreibweise genannt, in der mehrere Wörter aneinandergereiht geschrieben und jedes neue mit einem Großbuchstaben begonnen wird. z.B. *HammerThrowCompetition* statt *Hammer Throw Competition*.

[22] Natürlich achten Suchmaschinen nicht auf Groß- und Kleinschreibung, hier geht es um die zusammengesetzte Schreibweise.

[23] Die Bewertung des Ergebnisses im Vektorraum-Retrieval

Ergebnisses der Anfrage `+"dwain chambers" +"world rankings"` bereits in der zweiten Nachkommastelle. In Abbildung 3 ist dieser Umstand für eine Anfrage dargestellt.

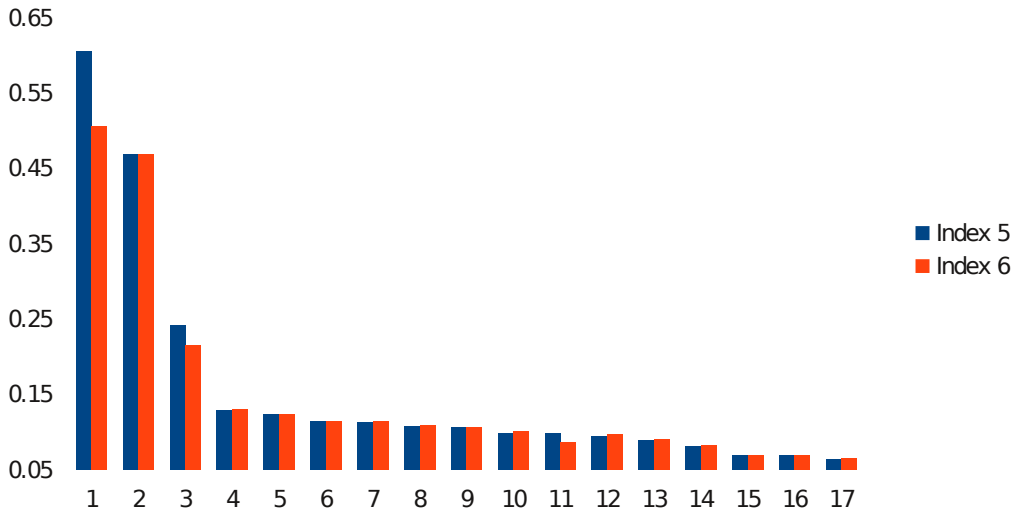

Abbildung 3: Scores der Ergebnisse der Anfrage `+fukuoka +marathon "paul biwott"` in Index 5 und 6. Die Ergebnisdokumente sind jeweils dieselben, jedoch fällt der Score pro Dokument in Index 6 um bis zu 16,5% kleiner aus. Die Reihenfolge der Ergebnisse für Index 6 wurde an die von Index5 angepasst.

Mit dem Ziel, den entstandenen Nachteil der camel-case Schreibweise auszugleichen, wurde für Index 7 das Verfahren angepasst. Für jedes gefundene Konzept wurde ein Algorithmus durchlaufen und sowohl die camel-case- als auch die gemischte Buchstaben-Zahlen-Schreibweise in die Wortbestandteile aufgeteilt und beide Schreibweisen hinzugefügt (Listing 2 auf der nächsten Seite). Von diesem Schritt wurde sich eine Vergrößerung der Ergebnismenge bei natürlichen Suchen versprochen, da die einzelnen Bestandteile zusammengesetzter Wörter auch als solche suchbar werden. Hierzu wurde ein weiterer Index generiert (Tabelle 4 auf Seite 23). Die Optimierung innerhalb von Apache Lucene jedoch führte bei diesem Index zu teilweise stark abweichenden Ergebnissen, da die Engine[24] in der Lage ist, mehrfach auftretende Tokens zu erkennen und zu löschen. Hierdurch wurde schnell deutlich, dass dieser Ansatz nicht zum gewünschten Ergebnis führt.

3.6 Proof Of Concept: Adressextraktion

Die bisherigen Experimente haben gezeigt, dass sich Suchergebnisse durch gezielte Hinzunahme semantischer Informationen verändern lassen und es möglich wird, Texte einem

[24]Der Kern von Lucene zur Indizierung von Dokumenten.

Abbildung 4: Beispielcode zum Aufteilen von Wörtern in *camel-case* Notation.

Listing 2: Algorithmus zur Erkennung von camel-case

```
1 private Set<String> splitCamelCase(String string)
2 {
3     HashSet<String> splits = new HashSet<String>();
4     String current = "";
5     char last = '\0';
6
7     for (char ch : string.toCharArray()) {
8         if (current.isEmpty() || Character.isLowerCase(ch)) {
9             current += ch;
10            last = ch;
11            continue;
12        }
13        if (Character.isUpperCase(ch)) {
14            if (Character.isUpperCase(last)) {
15                current += ch;
16            } else {
17                splits.add(current);
18                current = "" + ch;
19                last = ch;
20            }
21            continue;
22        }
23        if (Character.isDigit(ch)) {
24            if (Character.isDigit(last)) {
25                current += ch;
26            } else {
27                splits.add(current);
28                current = "" + ch;
29                last = ch;
30            }
31        }
32    }
33    splits.add(current);
34    return splits;
35 }
```

Index	Sem. Inf	Tiefe	Größe in MB	Konzeptinstanzen	CC-Split	Addressen
4	√	1	2,8			
5	√	2	2,8			
6	√	2	3,4	√		
7	√	2	3,2	√	√	
8	√	2	3,4	√	√	√

Tabelle 4: Index 7 (mit camel-case filter) ist kleiner als index 6. Dieser Umstand ist der Optimierung innerhalb von Apache Lucene geschuldet.

Suchbegriff zuzuordnen, die keinen direkten textuellen Bezug haben. Durch die Nutzung dieser Informationen eröffnen sich jedoch noch weitere Möglichkeiten. In diesem Teilexperiment soll betrachtet werden, ob sich durch die Nutzung von Daten aus den KBs Adressinformationen gewinnen lassen. Die Gewinnung von Adressinformationen wird als *Georeferenzierung* oder *Geocoding* bezeichnet. Typischerweise werden Adressen innerhalb von Texten durch syntaktische Auswertung erkannt und in Längen- und Breitenkoordinaten umgewandelt. Diese Informationen werden im Internet oft genutzt, um eine Karte darzustellen. Daneben gibt es noch weitere Anwendungsfälle wie z.B. das Einpassen von Geodaten in ein Computermodell mit Hilfe von GPS[25]. Für das Geocoding gibt es verschiedenen Anbieter, die eine Schnittstelle anbieten, mit der man Adressen in GPS Daten umwandeln kann oder umgekehrt. Für dieses Experiment kam die Geocoding Engine von Google[26] zum Einsatz.

Die Übersetzung von Adressen im Text in Geodaten findet seit Jahren breite Anwendung und birgt in sich keinen Mehrwert für dieses Experiment. Durch die Hinzunahme von semantischen Informationen jedoch, kommt eine Dimension hinzu, die bisher wenig Anwendung fand. Durch die Informationen, die aus den KBs gewonnen werden, können auch Adressen von Orten bestimmt werden, die zwar im Kontext eines Dokuments relevant, jedoch nicht direkt erwähnt oder als solche gekennzeichnet sind. Für dieses Experiment wird sich auf die Suche nach Adressen von Stadien beschränkt und dabei eine strikte Vorgehensweise eingehalten.

1. Finde alle Instanzen von aeo:SportsEvent.
2. Für jedes dieser Individuen:
   a) Finde alle Individuen, die über die *aeo:hasPart* Rolle mit diesem verwandt sind.
   b) Für jedes dieser Individuen:
      i. Finde alle Individuen, die über die *aeo:takesplaceInCity* Rolle mit diesem verwandt sind.
      ii. Finde die Namen dieser Individuen (Städte).

[25] Global Positioning System
[26] http://code.google.com/apis/maps/documentation/geocoding

iii. Finde alle Individuen, die über die *aeo:takesplaceInCountry* Rolle mit diesem verwandt sind.

iv. Finde die Namen dieser Individuen (Länder).

c) Finde alle Individuen, die über die *aeo:takesPlaceInSportsPOI* Rolle mit den *SportsEvent* Individuen verwandt sind.

d) Rufe den Inhalte der *gio:hasStadiumNameValue* Eigenschaft ab.

Wenn all diese Anfragen ein Ergebnis liefern, liegt ein Datensatz bestehend aus dem Namen eines Stadions, der Stadt und dem Land in dem dieses Stadion steht vor. Diese Daten werden dann der Geocoding-Engine übergeben und bei erfolgreicher Anfrage die genaue Adresse und die globale Position des Stadions gewonnen. Diese Daten werden dem Suchindex als Feld hinzugefügt, sodass sie als abrufbares Ergebnis zur Verfügung stehen. Zur Demonstration der Ergebnisse wurde das GUI von *SemTexSearch* um eine Spalte erweitert, in der gefundene Adressen dargestellt werden. Obwohl in den meisten Fällen

119	newsId=34416.h IAAF I Associ	Pino Dordoni stadium	0.074
120	newsId=38997.h title:		0.074
121	index.html title: Picker victory	Crystal Palace stadium	0.074
122	newsId=39894.h IAAF In Associa		0.074
123	newsId=31403.h IAAF In Associa		0.074
124	view. aspx@DUID=US html	Olympic stadium	0.074

Abbildung 5: Gefundenen Adressen werden in einer neuen Spalte dargestellt. Die gesammelten Daten können auch zum Einblenden einer Karte genutzt werden.

die Stadien in den KBs auch direkt im Text genannt wurden, lassen sich einige Beispiele finden, in denen diese Informationen nicht auf direkte Weise zugänglich waren, durch semantische Anreicherung aber trotzdem gewonnen werden konnten. Viele der Daten aus dem Boemie Repository sind jedoch mittlerweile zu alt, um zu genauen Ergebnissen zu führen. Dadurch, dass Stadien umbenannt werden (z.B. durch Sponsorwechsel), lag die Erkennungsrate auszugsweise zwar bei fast 100%, es ließen sich jedoch nicht alle gefundenen Stadien geocoden. Trotzdem kann dieser Versuch als Erfolg gewertet werden, da sich die Erkennung von Stadien ausschließlich auf die Informationen in den KBs beschränkt und keinerlei reine Texterkennung stattgefunden hat. Auch wenn die Abhängigkeit von

der Qualität der KB Informationen nicht zu verkennen ist. Der Nachrichtencharakter der vorliegenden Dokumente wirkt sich auf dieses Experiment klar negativ aus, da viele wichtige Informationen bereits im Text enthalten sind. Mit einer anderen, vor allem aktuelleren, Datenbasis könnten sich die Ergebnisse vermutlich noch verbessern lassen.

4 Ergebnisse und Bewertung

4.1 Anfragen

41. `+Hammerthrow +event`
42. `"Yipsi Moreno" +hammerthrow`
43. `women +Longjump +2003 +championship +final`
44. `+hammerthrow +"olympic games"`
45. `+"paula radcliffe" +ranking +2005`
46. `sprint +"veronica campbell" +100m +"world athletics final" +2004`
47. `+"Dwight Phillips" +2005 +result`
48. `+hurdling +2006 +"Liu Xiang" +110m +times`
49. `+women +20km +walking +"prize money"`
50. `+"steven hooker" pole~vault +rank`

4.2 Ergebnisstabelle

	Index 5				Index 6			
Query Nr.	Erg.	Rel.	Pre	Avg. Pre	Erg.	Rel.	Pre	Avg. Pre
10	17	10	0.59	1	19	10	0.53	1
15	23	1	0.04	0.01	24	1	0.04	0.01
41	0	0			3	2	0.67	0.6
42	0	0			3	2	0.67	0.6
43	1	1	1	0.1	7	2	0.29	0.6
44	0	0			1	1	1	0.1
45	4	4	1	0.8	16	6	0.38	0.9
46	3	1	0.3	0.1	4	2	0.5	0.6
47	9	5	0.56	0.45	9	5	0.56	0.45
48	19	6	0.32	0.9	21	6	0.29	0.9
49	5	3	0.6	0.7	5	3	0.6	0.7
50	3	3	1	0.7	6	5	0.83	0.9

4.3 Analyse

Von den 40 Anfragen in [Reb09] ließ sich bei naiver Durchführung einer Batch-Suche nur bei zwei Anfragen (10 und 15) eine Veränderung in den Ergebniszahlen beobachten. Dies entspricht weitgehend der Hypothese, dass sich die Ergebnisse nur leicht verändern lassen. Diese wird durch die Tatsache, dass es keinen Ansteig an relevanten Ergebnissen gibt, gestützt. An den eigens für dieses Experiment generierten Anfragen lässt sich jedoch gut erkennen, dass die Anreicherung mit weiteren Daten eine deutliche Veränderung der Resultate auslösen kann. Die Qualität dieser Ergebnisse hängt aber stark von dem gewählten Suchterm ab. Die hier genutzten Terme wurden alle generiert indem die Liste der neuen Tokens durchlaufen und in den Suchergebnissen nach weiteren Kriterien gesucht wurde. Trotzdem lässt sich nicht in allen Fällen ein Unterschied zwischen den beiden Indexen erkennen.

Entsprechend der Hypothese ist in den meisten Fällen die Anzahl der Gesamtergebnisse gestiegen. Die Auswirkungen auf den Score sind jedoch geringer ausgefallen als erwartet. In einigen Fällen hat die neue Gewichtung jedoch sogar dazu geführt, dass relevante Ergebnisse weiter oben in der Liste auftraten. So bei Anfrage 47, in der ein irrelevantes Dokument an dritter Stelle in den Ergebnissen von Index 5 für Index 6 mit dem an vierter Stelle getauscht hat. Gleiches ist bei Anfrage 50 zu beobachten (Ergebnis 5 wird zu Ergebnis 4). Anfrage 45 brachte sogar zwei gänzlich neue, relevante Dokumente zu Tage, was sogar eine qualitative Erhöhung des Recalls bedeutet, während die Precision von 1 auf 0.38 sinkt. In Anfrage 46 lässt sich die direkte Auswirkung der Nutzung eines der neuen Token beobachten. Das Dokument mit dem höchsten Score, das auch der beste Treffer war, taucht bei der Suche in Index 5 gar nicht in den Ergebnissen auf. Den Ausschlag gibt hier das Suchwort `+100m`, das in Index 5 nicht in den Semantischen Daten enthalten war.

Insgesamt zeigt sich, dass die Ergebnise aus [Reb09] weitestgehend nachvollziehbar sind. Die neu gewonnenen Daten führen bei geschickter Wahl der Anfragen zu den erwarteten Ergebnissen und fügen sich damit in das Fazit von Herrn Rebhan ein. Der Nachrichtencharakter der Dokumente wirkt sich auf diese Ergebnisse genauso aus wie auf die von Herrn Rebhan. Trotzdem lassen sich die Ergebnisse leicht verbessern, was der eingangs aufgestellten Hypothese entspricht.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[Reb09] Torben Rebhan. „Untersuchungen zur Kombination von klassischer und semantischer Textindizierung“. Technische Universität Hamburg Harburg, 2009.

[Web07] Ingmar Weber. „Efficient Index Structures for and Applications of the CompleteSearch Engine“. Diss. Universität des Saarlandes, 2007.

[Wik11c] Wikipedia. *Semantik*. 1. Juli 2011. URL: `http://de.wikipedia.org/wiki/Semantik`.

[Wik11a] Wikipedia. *Metadaten*. 6. Juli 2011. URL: `http://de.wikipedia.org/wiki/Metadaten`.

[BOE11] BOEMIE. *Bootstrapping Ontology Evolution with Multimedia Information Extraction*. 4. Juni 2011. URL: `http://www.boemie.org/`.

[Wik11b] Wikipedia. *Ontologie (Informatik)*. 7. Juli 2011. URL: `http://de.wikipedia.org/wiki/Ontologie_(Informatik)`.

[KG07] Racer Systems GmbH & Co. KG. *RacerPro User's Guide Version 1.9.2*. 18. Okt. 2007.

[KG11] Racer Systems GmbH & Co. KG. *RacerPro 2.0*. 6. Juli 2011. URL: `http://www.racer-systems.com`.

[Fre+04] Eric Freeman u. a. *Head First Design Patterns*. 1. Aufl. O'Reilly Media, Nov. 2004. ISBN: 978-0596007126.

[Abr10] Witold Abramowicz, Hrsg. *Knowledge-Based Information Retrieval and Filtering from the Web*. 1. Aufl. Springer, Dez. 2010. ISBN: 9781441953766.