UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Navigation in DBpedia mit Hilfe maschinellen Lernens

Diplomarbeit

Leipzig, Oktober 2008

vorgelegt von

Knappe, Sebastian
geb. am: 11.12.1980

Studiengang Diplom Informatik

Betreuender Hochschullehrer:
Prof. Dr. Ing. habil. Klaus-Peter Fähnrich

Abteilung Betriebliche Informationssysteme am Institut für Informatik der Universität Leipzig

verantwortlicher Betreuer:
Jens Lehmann

Zusammenfassung

Eine Vision des Semantic Web ist es, die Vorzüge semantischer Technologien einem weitem Publikum zugänglich zu machen. Große Wissensbasen, wie zum Beispiel DBpedia, YAGO und andere sind frei verfügbar als Linked Data und SPARQL-Endpoints. In dieser Arbeit stelle ich die Anwendung *DBpedia Navigator* vor, welche es Nutzern erlaubt, in DBpedia und den damit verbundenen Datenbeständen zu navigieren, zu suchen und zu browsen. Im speziellen präsentiere ich ein neuartiges Feature: Navigationsvorschläge. Basierend auf den letzten Instanzen, die im DBpedia Navigator angezeigt wurden, werden dem Nutzer Vorschläge über damit verbundene Instanzen geliefert, an denen er interessiert sein könnte. Diese Vorschläge nutzen die Semantik der zugrunde liegenden Wissensbasis voll aus und werden von einem überwachten maschinellen Lernalgorithmus für OWL geliefert, der als Open Source im DL-Learner Projekt verfügbar ist.

Inhaltsverzeichnis

Glossar

Ajax	asynchrone Datenübertragung zwischen Server und Browser
Artikel	grafische Aufbereitung von Informationen zu einer Resource
browsen	schmökern in einem Datenbestand
DBpedia	frei verfügbare Wissensbasis, aus Wikipedia-Artikeln extrahiert
DBpedia Navigator	Programm zum Navigieren in DBpedia mittels Maschinellem Lernen
DL-Learner	Tool für überwachtes Maschinelles Lernen in OWL und DL
Extraktion	Auswahl einer für das Lernen benötigten Teilmenge der Wissensbasis
JENA	Framework für den Bau von Semantic Web Anwendungen
JSON	kompaktes Datenformat in für Mensch und Maschine lesbarer Form
Klasse	die Definition einer Gruppe gleichartiger Objekte
Konzept	eine Klasse von Objekten in Beschreibungslogiken
Konzeptbeschreibung	die Definition eines Konzepts mittels anderer Konzepte
Korpus	eine Sammlung von Texten, die Gegenstand einer Darstellung wird
Label	der Name einer Resource, im Gegensatz zu einer URI nicht eindeutig
Linked Data	Datensätze, die über RDF-Tripel verbunden sind
Literal	Zeichenfolgen, zur Darstellung der Werte von Basistypen zulässig
Maschinelles Lernen	Maschine findet logische Erklärung für gegebene Daten
Navigationsvorschläge	mittels Maschinellem Lernen erstellte Konzeptbeschreibungen zum Navigieren in DBpedia
Objekt	der Eigenschaftswert einer Resource in einem RDF-Tripel
Pagerank	Verfahren, verlinkte Dokumente anhand ihrer Struktur zu gewichten
Properties	Eigenschaften von Resourcen
Prädikat	eine Eigenschaft einer Resource in einem RDF-Tripel
Reasoner	zieht aus einer Menge von Fakten oder Axiomen logische Konsequenzen
Resource	eine Entität, die durch eine Menge von Fakten beschrieben wird
REST	Softwareachitekturstil, wobei jede Resource eigene URI hat
Semantic Web	Erweiterung des WWW, Informationen wird Bedeutung zugeordnet
SPARQL-Endpoint	Schnittstelle zur Wissensbasis für SPARQL-Anfragen
Statement	eine Aussage über ein Objekt, bestehend aus Subjekt, Prädikat, Objekt
Subjekt	eine Resource, die beschrieben wird in einem RDF-Tripel
Tripel	RDF-Statement bestehend aus Subjekt, Prädikat und Objekt
URI	ein Identifikator zur Identifizierung von Resourcen im Internet
Usability	Benutzerfreundlichkeit einer Anwendung
YAGO	frei verfügbare Wissensbasis mit Klassenhierarchie

1 Einleitung

Die Vision des *Semantic Web* ist es, semantische Repräsentationen auf der größtmöglichen Ebene zu nutzen, dem Internet. Momentan sind Semantic Web Technologien auf dem Vormarsch und große Wissensbasen wie zum Beispiel *DBpedia*[1], *YAGO*[20] und andere sind frei verfügbar. Diese Wissensbasen basieren auf semantischen Wissensrepräsentationsstandards wie *RDF* und *OWL*. Sie beschreiben Millionen von Objekten und enthalten Hunderttausende von Eigenschaften und Klassen, im Falle von DBpedia akkumulieren sich diese Eigenschaften und Klassen der Objekte zu mehr als 100 Millionen Tripeln. Diese Wissenbasen sowie einige andere [1] liegen als *Linked Data* [6; 7] oder SPARQL-Endpoints [17] vor.

Aufgrund ihrer riesigen Größe treffen Nutzer allerdings auf das Problem, dass sie kaum alle *Bezeichner* kennen können, die in den Wissensbasen genutzt werden. In den meisten Fällen können diese Nutzer also keine Anfragen in einer strukturierten Form wie SPARQL formulieren, aber sie haben oft eine recht genaue Vorstellung davon welche Resultate sie gerne erhalten würden. Ein Historiker beispielsweise, der in DBpedia nach altertümlichen griechischen Philosophen sucht, die durch Plato beeinflusst wurden, kann recht einfach passende Beispiele nennen oder aus einer Liste falsche aussortieren. Er wird aber wahrscheinlich nicht in der Lage sein, aus dem Stehgreif eine formale Anfrage zu konstruieren, welche eine korrekte Liste aus der riesigen DBpedia Wissensbasis extrahiert.

Der *DBpedia Navigator* ist eine Anwendung, die versucht, dieses Problem zu bewältigen, indem es das browsen in großen Wissensbasen erleichtert und die Suche darin ohne tief gehende Kenntnisse formaler Suchmechanismen ermöglicht. Sie wurde als eine Schnittstelle zu DBpedia und dem damit verlinkten Datenbestand geschrieben, kann aber im Prinzip so konfiguriert werden, dass sie in Verbindung mit beliebigen großen Wissensbasen genutzt werden kann, welche als SPARQL-Endpoints verfügbar sind. Die Anwendung geht diese Aufgabe an, indem sie Techniken des *Maschinellen Lernens* nutzt, um dem Nutzer *Navigationsvorschläge* anzubieten. Außerdem gibt es noch weitere Möglichkeiten die zugrunde liegende Semantik zu nutzen, um durch die Wissensbasis zu navigieren. Dabei nutzt die Anwendung Suchmechanismen, eine Klassenhierarchie oder Eigenschaften von Objekten. Die Informationen zu ausgewählten Objekten werden zu einem Artikel zusammengefasst, der in einer Wikipedia[2]-artigen Weise erscheint.

Die Diplomarbeit ist folgendermaßen strukturiert:
Kapitel 2 enthält die Grundlagen, die zum Verständnis der weiteren Arbeit notwendig sind, es beschäftigt sich mit dem Semantic Web und zugehörigen Technologien, sowie mit Wissensbasen und Maschinellem Lernen. Im Kapitel 3 wird der Aufbau des DBpedia Navigators beschrieben, das Zusammenspiel zwischen dem Navigator Client, einer Datenbank und dem *DL-Learner*. Im folgende

[1] `http://esw.w3.org/topic/SparqlEndpoints` und `http://linkeddata.org`
[2] `http://www.wikipedia.org`

Kapitel 4 wird der DBpedia Navigator Client genauer beschrieben, alle Funktionen inklusive deren GUI und Funktionsweise. Es schließt sich das Kapitel 5 an, in dem die Generierung der Navigationsvorschläge genau beschrieben ist. Eine Evaluation der Laufzeit der Navigationsvorschlagsgenerierung sowie der Qualität der Vorschläge befindet sich dann in Kapitel 6. Das letzte Kapitel 7 widmet sich der Zusammenfassung der Arbeit. Es enthält außerdem eine Zusammenstellung von möglichen Weiterentwicklungen für die vorliegende Anwendung.

2 Grundlagen

2.1 Semantic Web: Vision

Das Internet wurde geschaffen, um nicht nur den Informationsaustausch zwischen Menschen, sondern auch Maschinen den Zugriff auf und die Verarbeitung von Informationen zu ermöglichen. Das Haupthindernis dafür ist, dass die meisten Informationen für Menschen aufbereitet sind und nicht in einer maschinenlesbaren Form vorliegen. Das Semantic Web ist die Vision eines Netzes, in dem Informationen eine Bedeutung und Struktur zugeordnet wird und sie so auch für Maschinen verständlich werden.

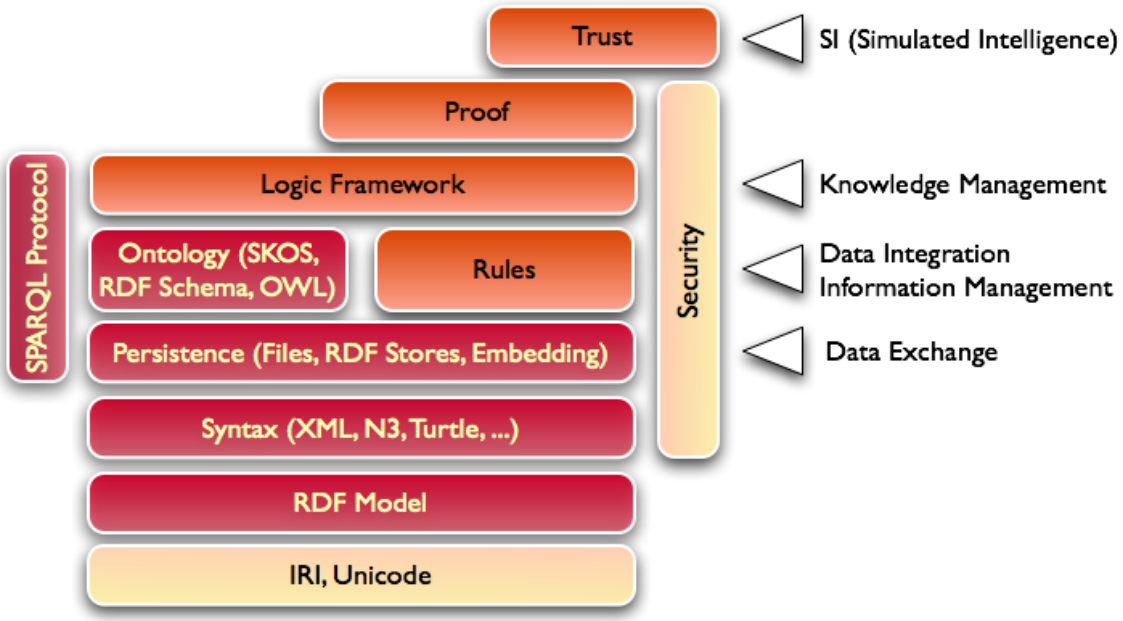

Abbildung 1: Schichtenmodell des Semantic Web und Anwendungsmöglichkeiten

Im heutigen Netz können Rechner das Netz durchsuchen, indem sie Webseiten nach Wörtern parsen, die mit der gesuchten Information in Verbindung stehen. Sucht man z.B. nach dem Geburtsdatum von Albert Einstein, empfiehlt es sich, `Albert Einstein` und `Geburt` oder `Biografie` als Stichwörter zu benutzen. Dadurch findet man Webseiten, auf denen es um die Biografie Albert Einsteins geht und man findet mit großer Wahrscheinlichkeit auch sein Geburtsdatum. Aber die benutzte Suchmaschine hat keine Ahnung von der Semantik des Wortes Geburtsdatum. Sie findet die Information nicht auf Grund der Bedeutung der Worte, sondern durch reinen Zeichenkettenvergleich. Werden komplexere Zusammenhänge gesucht, wie z.B. alle Wissenschaftler, die im selben Jahr wie Einstein

geboren wurden, versagt diese Technik. Das Semantic Web gibt dem Inhalt von Webseiten Struktur und Bedeutung. Es wird möglich das Geburtsjahr Einsteins zu finden, alle Personen, die in diesem Jahr geboren sind und von diesen wiederum alle Wissenschaftler. Aber die Bedeutung des Semantic Web geht über die Verbesserung von Suchmaschinen hinaus. Es wird eine Umgebung geschaffen, in der Software Agenten Webseiten durchsuchen und komplexe Aufgaben durchführen können. Man könnte sich einen solchen Agenten im Handy oder Handheld vorstellen, der seinen Standort über GPS Koordinaten bestimmt, Information über Sehenswürdigkeiten auf der Internetseite der Stadt abruft, ein Restaurant in der Nähe findet und dort einen Tisch bucht und gleichzeitig einen lokalen Blumendienst ausfindig macht, der einen Strauss Blumen in das Restaurant liefert, mit einer persönlichen Glückwunschkarte zum Hochzeitstag. Er könnte all dies bewerkstelligen, aufgrund der Tatsache, dass er die Bedeutung der Worte Adresse, Restaurant oder Blumen `versteht` und einen Zusammenhang herstellen kann. Ein solches Semantic Web würde also nicht nur das Internet selbst verändern, sondern unser Leben als ganzes, indem Maschinen viele Aufgaben automatisieren, die heute von Hand erledigt werden müssen, oder sogar ganz neue Möglichkeiten bieten ([5] und [18]).

2.2 Beschreibungslogiken (Description Logics)

Beschreibungslogiken sind eine Klasse von Logik-basierten Wissensrepräsentationsformalismen. Das Ziel dieser Logiken ist es, Aussagen über die Welt auf einem hohen Level zu treffen, welche von intelligenten Applikationen effektiv genutzt werden können. Dazu existiert eine wohldefinierte Syntax und eine formal eindeutige Semantik, die auf der Übersetzbarkeit zur First-Order-Prädikatenlogik basiert. Wichtige Elemente der Anwendungsdomäne wie Klassen, Beziehungen und Objekte werden definiert, sowie Einschränkungen in der Art und Weise wie diese Elemente interpretiert werden können. Klassen, auch Konzepte genannt, können Dinge sein wie `Person`, `Lehrer` oder `Student`, Beziehungen, auch Rollen genannt, sind z.B. `lehrt`, `besucht`, `mag`. Objekte, auch Individuen genannt, sind z.B. `Franz`, `Werner` usw., eine Einschränkung wäre `Kurse werden nur von Lehrern gelehrt`. Beschreibungslogiken sind ein Eckpfeiler des Semantic Web wegen dem Nutzen beim Erstellen von Ontologien. OWL-DL und OWL-Lite basieren auf Beschreibungslogiken[3].

2.2.1 Syntax

Die Syntax von Beschreibungslogiken besteht aus

- einer Menge von einstelligen Prädikatensymbolen zur Bezeichnung von Konzeptnamen
- einer Menge von zweistelligen Prädikatensymbolen zur Bezeichnung von Rollennamen

- einer rekursiven Definition um Konzeptbeschreibungen aus Konzeptnamen und Rollennamen aufzubauen

Bei Beschreibungslogiken werden Konzeptnamen als atomare Konzepte bezeichnet und Rollennamen als atomare Rollen. Im allgemeinen bezeichnet ein Konzept die Menge an Individuen, die zu diesem Konzept gehören und eine Rolle bezeichnet eine Beziehung zwischen Konzepten. Die rekursive Definition beinhaltet logische Konstruktoren wie Konjunktion, Disjunktion oder Negation. Allerdings werden bei Beschreibungslogiken für Konjunktion und Disjunktion die Symbole $\sqcap$ und $\sqcup$ benutzt. Das folgende ist ein Beispiel einer Definition der Syntax der Beschreibungslogik ALC.

Sei N_C eine Menge von Konzeptnamen und N_R eine Menge von Rollennamen. ALC Konzeptbeschreibungen sind dann wie folgt induktiv definiert:

- Wenn $A \in N_C$, dann ist A eine ALC Konzeptbeschreibung
- Wenn C, D ALC Konzeptbeschreibungen sowie r eine Rolle sind und $r \in N_R$, dann sind die folgenden auch ALC Konzeptbeschreibungen:
  - $\texttt{C} \sqcap \texttt{D}$ (Konjunktion)
  - $\texttt{C} \sqcup \texttt{D}$ (Disjunktion)
  - $\neg\texttt{C}$ (Negation)
  - $\forall\texttt{r}.\texttt{C}$ (Einschränkung)
  - $\exists\texttt{r}.\texttt{C}$ (Einschränkung)
- Für viele Beschreibungslogiken erlaubt man außerdem noch Äquivalenz- und Subklassenaxiome der Form:
  - $\texttt{C} \equiv \texttt{D}$ (Äquivalenz)
  - $\texttt{C} \sqsubseteq \texttt{D}$ (Subklasse)

Damit sind Beispiele für Konzeptbeschreibungen:

$\texttt{Person} \sqcap \texttt{Weiblich}$
$\texttt{Person} \sqcap \exists\texttt{besucht.Kurs}$
$\texttt{Person} \sqcap \forall\texttt{besucht.}(\texttt{Kurs} \sqcap \neg\texttt{Einfach})$
$\texttt{Person} \sqcap \exists\texttt{lehrt.}(\texttt{Kurs} \sqcap \forall\texttt{topic.DL})$
$\texttt{Person} \sqcap \forall\texttt{lehrt.}(\texttt{Kurs} \sqcap \exists\texttt{topic.}(\texttt{DL} \sqcup \texttt{NMR})$

2.2.2 Semantik

Die Semantik von Beschreibungslogiken wird definiert, indem man Konzepte als Mengen von Individuen und Rollen als Mengen von Paaren von Individuen auffasst. Diese Individuen gehören einer bestimmten Domäne an. Die Semantik zusammengesetzter, nicht atomarer Konzepte und Rollen ergibt sich dann rekursiv aus der der atomaren. Für die ALC Beschreibungslogik ist die Semantik folgendermaßen definiert:

Eine Interpretation $I = (\Delta^I, .^I)$ besteht aus einer nicht leeren Domäne Δ^I und einer Interpretationsfunktion $.^I$:

- $A^I \subseteq \Delta^I$ für alle $A \in N_C$,
- $r^I \in \Delta^I \times \Delta^I$ für alle $r \in N_R$.

Die Interpretationsfunktion wird für ALC Konzeptbeschreibungen erweitert zu:

- $(C \sqcap D)^I := C^I \cap D^I$
- $(C \sqcup D)^I := C^I \cup D^I$
- $(\neg C)^I := \Delta^I \backslash C^I$
- $(\forall r.C)^I := \left\{ d \in \Delta^I \mid \forall e \in \Delta^I : (d, e) \in r^I \; impliziert \; e \in C^I \right\}$
- $(\exists r.C)^I := \left\{ d \in \Delta^I \mid \exists e \in \Delta^I : (d, e) \in r^I \; und \; e \in C^I \right\}$

2.2.3 Modellieren mit Beschreibungslogiken

Eine Wissensbasis in der Beschreibungslogik besteht aus einer so genannten TBox, die die Terminologie der Anwendungsdomäne definiert und der ABox, die Fakten über eine bestimmte Welt enthält. In der ALC Beschreibungslogik ist eine Konzeptbeschreibung von der Form $A \equiv C$ wobei

- A ist ein Konzeptname
- C ist eine Konzeptbeschreibung

Eine TBox ist dann eine endliche Menge von Konzeptbeschreibungen, die keine mehrfachen und keine zyklischen Definitionen enthält. Die TBox enthält Konzepte und Rollen wie `Lehrer` oder `lehrt`, bzw. Konzeptbeschreibungen, wie z.B.: `ein Lehrer ist eine Person, die lehrt`.

Zuweisungen sind von der Form $C(a)$ (Konzeptzuweisung) oder $r(a, b)$ (Rollenzuweisung), wobei `C` eine Konzeptbeschreibung, `r` eine Rolle und a, b Individuen aus einer Menge N_I solcher Individuen sind. Eine ABox ist eine endliche Menge von Zuweisungen. Die ABox enthält dann Fakten wie `Franz ist Lehrer`.

Übersetzt man Beschreibungslogiken zu First-Order-Logik, gibt es keinen signifikanten Unterschied zwischen Sätzen aus der TBox und der ABox. Diese Unterscheidung wird trotzdem getroffen, weil z.B. *Reasoner* (Software, die aus einer Menge von Fakten oder Axiomen logische Konsequenzen zieht) unterschiedliche Funktionen auf diesen Teilen der Wissensbasis ausführen. Außerdem ist es für jemanden, der Wissen modelliert, durchaus plausibel zwischen der Definition von Konzepten in der Welt und ihren konkreten Manifestationen zu unterscheiden.

2.2.4 Die Ausdruckskraft von Beschreibungslogiken

Es gibt einige Variationen von Beschreibungslogiken und dafür eine informalle Namenskonvention, die ungefähr die Operatoren beschreiben, die darin erlaubt sind. Die Ausdruckskraft ist in den Bezeichnungen für eine Logik kodiert, wobei jeder Buchstabe eine Menge an erlaubten Operatoren bezeichnet.

$\mathcal{SHOIN}^{(\mathcal{D})}$

Abbildung 2: OWL DL basiert auf der Beschreibungslogik mit dieser Bezeichnung

$\mathcal{SHIF}^{(\mathcal{D})}$

Abbildung 3: OWL Lite basiert auf der Beschreibungslogik mit dieser Bezeichnung

2.3 RDF - Ressource Description Framework

Um Informationen Bedeutung zuzuordnen bedarf es eines Weges diese Bedeutung auszudrücken. Das Resource Description Framework (RDF)[3] ist eine Sprache zur Repräsentation von Informationen über Resourcen im Web, speziell Metadaten wie zum Beispiel Titel, Autor und Modifikationsdatum einer Webseite. Das das Konzept einer Webresource allerdings sehr verallgemeinert wird, ist es auch möglich, Informationen über Dinge zu repräsentieren, die im Web identifiziert aber nicht abgefragt

[3] `http://www.w3.org/TR/rdf-primer/`

werden können. RDF ist für Situationen gedacht, in denen Informationen von Anwendungen bearbeitet werden müssen und nicht nur für Menschen angezeigt werden. RDF bietet dabei ein allgemeines Framework, um die Informationen auszudrücken, die zwischen Applikationen ausgetauscht werden können. Dabei basiert RDF auf der Idee Dinge über so genannte *Uniform Resource Identifier* (oder URI's) zu identifizieren und die Resourcen mit einfachen Eigenschaften und Eigenschaftswerten zu beschreiben. Diese Beschreibungen erfolgen in der Form von Statements, wie z.B.:

```
Eric Miller besitzt den Titel Dr.
```

Dabei ist `Eric Miller` die beschriebene Resource, `besitzt den Titel` die Eigenschaft, `Dr.` der Wert der Eigenschaft. Resourcen wie auch Eigenschaften werden durch URIs identifiziert, Werte der Eigenschaften können wieder Resourcen sein, die durch URIs dargestellt würden oder einfache Werte wie Zahlen oder Zeichenketten sein, so genannte Literale. Damit würde das Beispiel dann lauten:

```
http://www.w3.org/People/EM/contact#me
    http://www.w3.org/2000/10/swap/pim/contact/#personalTitle
        "Dr."
```

RDF nutzt dabei eine spezielle Terminologie, um über die verschiedenen Teile eines Statements zu reden, der Teil der die Sache beschreibt, über die das Statement abgegeben wird, wird *Subjekt* genannt, die Eigenschaft wird *Prädikat* genannt und der Wert der Eigenschaft *Objekt*. Mittels dieser Statements lassen sich komplexe Informationen über Resourcen darstellen.[10] Es gibt mehrere Arten diese Informationen darzustellen, z.B. als einen gerichteten Graph mit Knoten und Pfeilen, wobei die Knoten, von denen die Pfeile losgehen Subjekte darstellen, die Pfeile Prädikate und die Knoten, auf die die Pfeile zeigen die zugehörigen Objekte.

RDF bietet außerdem eine XML-basierte Syntax (RDF/XML genannt), mit der diese Daten auch in maschinenlesbarer Art ausgetauscht werden können. Das korrespondierende XML zu dem obigen Graph ist damit:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

Abbildung 4: Ein RDF-Graph der Eric Miller beschreibt

Ein weiterer alternativer Weg Statements aufzuschreiben, ist mit so genannten *Tripeln*. In der Tripel-Notation wird jedes Statement als Tripel aus Subjekt, Prädikat und Objekt geschrieben, in dieser Reihenfolge. Damit würde das obige Beispiel als Menge von Tripeln geschrieben folgendermaßen lauten:

```
<http://www.w3.org/People/EM/contact#me>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://www.w3.org/2000/10/swap/pim/contact#Person> .
<http://www.w3.org/People/EM/contact#me>
    <http://www.w3.org/2000/10/swap/pim/contact#fullName>
        "Eric Miller" .
<http://www.w3.org/People/EM/contact#me>
    <http://www.w3.org/2000/10/swap/pim/contact#mailbox>
        <mailto:em@w3.org> .
<http://www.w3.org/People/EM/contact#me>
    <http://www.w3.org/2000/10/swap/pim/contact#personalTitle>
        "Dr." .
```

2.4 OWL - Web Ontology Language

OWL[4] wird von Applikationen genutzt, die die Bedeutung von Informationen verarbeiten und diese Informationen nicht nur Menschen-lesbar darbieten. Es ist somit eine Art von Erweiterung von

[4] http://www.w3.org/TR/owl-features/

RDF und RDF-Schema, indem es ein erweitertes Vokabular und eine formale Semantik bietet, z.B. Beziehungen zwischen Klassen (Disjointness), Kardinalität, Eigenschaften von Properties (vorher als Eigenschaften bezeichnet) wie zum Beispiel Symmetrie, usw. Es kann genutzt werden, um die Bedeutung von Begriffen und ihre Beziehung zueinander, das heisst eine Ontologie, zu beschreiben. Dies ermöglicht Maschinen sinnvolle Reasoning Aufgaben zu erledigen. OWL bietet drei immer ausdrucksstärker werdende Subsprachen, die für bestimmte Zwecke und Nutzer unterschiedlich geeignet sind[4].

2.4.1 OWL Lite

OWL Lite ist hauptsächlich dafür da, eine simple Klassenhierarchie mit einfachen Restriktionen darzustellen. Es unterstützt z.B. Kardinalitätsbeschränkungen, aber nur mit den Werten 0 und 1 und Klassen können nur mittels benannter Superklassen definiert werden. Es ist weniger komplex als OWL DL und es ist leichter, Tools zur Verarbeitung von OWL Lite zu entwickeln, als für die komplexeren Sprachen OWL DL und OWL Full. Es basiert auf der Beschreibungslogik SHIF(D).

2.4.2 OWL DL

OWL DL bietet maximale Ausdruckskraft, bleibt aber trotzdem noch entscheidbar (alle Berechnungen enden in endlicher Zeit) und berechenbar. Es enthält alle Sprachkonstrukte von OWL, im Gegensatz zu OWL Full können sie aber nur mit einigen Einschränkungen genutzt werden. Klassen können selber keine Instanzen von anderen Klassen sein, allgemeiner gesagt müssen Klassen, Instanzen und Properties disjunkte Mengen sein. Außerdem müssen Properties entweder `ObjectProperties` (Beziehungen zwischen Instanzen zweier Klassen) oder `DatatypeProperties` (Beziehungen zwischen Instanzen einer Klasse und RDF Literalen bzw. XML Schema Datentypen) sein. Im Vergleich zu OWL Lite bietet OWL DL Kardinalitätsbeschränkungen mit beliebigen nicht negativen ganzen Zahlen oder auch die Bool'sche Kombination von Klassen mittels `unionOf` oder `complementOf`. Der Name dieser Subsprache ist so gewählt auf Grund der Korrespondenz mit der Beschreibungslogik SHOIN(D), welche die formale Grundlage von OWL darstellt.

2.4.3 OWL Full

OWL Full ist für Nutzer gedacht, die maximale Ausdruckskraft und syntaktische Freiheit wollen, ohne dass die Berechenbarkeit garantiert werden kann. Im Unterschied zu OWL DL kann z.B. eine Klasse sowohl eine Kollektion von Instanzen sein als auch selber eine Instanz. Auf Grund dieser Ausdrucksstärke ist es kaum möglich, dass ein Reasoner für alle Elemente von OWL Full Ergebnisse berechnen kann.

Jede dieser Subsprachen ist eine Erweiterung des simpleren Vorgängers, sowohl darin, was korrekt ausgedrückt als auch darin, was daraus geschlossen werden kann. So ist jede korrekte OWL Lite Ontologie eine korrekte OWL DL Ontologie und jede korrekte OWL DL eine korrekte OWL Full Ontologie. Genauso verhält es sich mit den Schlussfolgerungen. OWL Full kann als eine Erweiterung von RDF gesehen werden, während OWL DL und OWL Lite Erweiterungen einer eingeschränkten Sicht von RDF sind. Damit ist jedes OWL Dokument (Lite, DL, Full) ein RDF Dokument und jedes RDF Dokument ein OWL Full Dokument, aber nur manche RDF Dokumente sind korrekte OWL Lite bzw. OWL DL Dokumente[14].

2.5 SPARQL - Anfragesprache für RDF

SPARQL[5] ist eine Anfragesprache für RDF. Der Name ist ein Akronym und steht für Simple Protocol and RDF Query Language. Man kann sie nutzen, um Anfragen an unterschiedliche Datenquellen zu formulieren, egal ob die Daten direkt als RDF gespeichert oder als RDF mittels einer Middleware betrachtet werden. SPARQL bietet die Möglichkeit, benötigte oder auch optionale *graph pattern*, das heisst Teile eines RDF-Graphs anzufragen bzw. auch ihre Konjunktionen und Disjunktionen. Das Ergebnis von Sparql-Anfragen sind Ergebnismengen oder wieder RDF-Graphen.
In den meisten Fällen enthalten SPARQL-Anfragen Mengen von Tripelmustern, so genannte *basic graph pattern*. Diese Tripelmuster sind wie RDF Tripel, nur dass sowohl Subjekt, Prädikat als auch Objekt Variablen sein können. Diese Muster entsprechen Subgraphen des vorliegenden RDF-Graphen dann, wenn RDF-Terme dieses Subgraphen für die Variablen eingesetzt werden können und die Subgraphen äquivalent sind.[16]
Ein Beispiel solch einer SPARQL-Anfrage ist folgendes, welches im DBpedia Navigator genutzt wird:

```
SELECT ?label
FROM <http://dbpedia.org>
WHERE {
<http://dbpedia.org/class/yago/CitiesOnTheRhine>
<http://www.w3.org/2000/01/rdf-schema#label> ?label
}
```

Diese Anfrage würde eine Ergebnismenge produzieren, die das Label der Klasse `CitiesOnTheRine` enthält. Hinter dem SELECT Stichwort stehen immer die Ergebnis-Variablen,

[5] `http://www.w3.org/TR/rdf-sparql-query/`

also die Variablen, deren Ergebnisse in der Ergebnismenge vorliegen. Nach dem FROM Stichwort ist der Name des RDF-Graphen aufgeführt, in dem gesucht wird. Die WHERE Klausel enthält die Bedingungen oder auch Muster, die ein Subgraph enthalten muss. Das vorliegende beispiel enthält ein Tripelmuster mit dem Subjekt `http://dbpedia.org/class/yago/CitiesOnTheRhine`, dem Prädikat `http://www.w3.org/2000/01/rdf-schema#label` und dem Objekt `?label`, welches in diesem Fall eine Variable ist, für die RDF-Terme eingesetzt werden können.
Ein SPARQL-Endpoint ist ein SPARQL Protokoll Service. Er versetzt Nutzer in die Lage Wissensbasen mittels der SPARQL-Anfragesprache anzufragen. Die Resultate werden meist in verschiedenen maschinenlesbaren Formaten zurückgegeben. Es handelt sich also um eine Maschinen-freundliche Schnittstelle zu einer Wissensbasis. Normalerweise bezeichnet man damit einen Server, der über das Internet angefragt werden kann.

2.6 DBpedia und YAGO

DBpedia[6] ist eine Community-Initiative, die strukturierte Daten aus Wikipedia[7] extrahiert. Diese Daten werden im Web zugänglich gemacht, mit anderen Datenkollektionen verbunden und es ist über entsprechende Schnittstellen möglich, anspruchsvolle Anfragen an die extrahierten Daten zu stellen.
Eines der Hauptprobleme beim Erschaffen strukturierter Daten im Web ist die schiere Menge an Informationen sowie ihre sich ständig verändernde Größe. Der *top-down-Ansatz* erst ein Schema für Daten zu erschaffen und dann die Daten mittels dieses Schemas zu erstellen, funktioniert dafür offensichtlich nicht. DBpedia geht deshalb so vor, eine große Menge an interessanten und bereits vorhandenen Daten zu nehmen (Wikipedia als eine der meistbesuchten Seiten im Netz) und daraus strukturierte Informationen zu extrahieren. Dabei entsteht ein Korpus an Daten mit über 100 Millionen RDF-Tripeln, wenn man diese mit anderen Datenkollektionen verbindet sogar ein Netz von Daten mit an die 2 Milliarden Tripeln.[1]
Um diese Daten zu extrahieren nutzt DBpedia die Struktur von Wikipedia Artikeln. Der größte Teil dieser Artikel besteht aus einfachem Text, aber es gibt auch strukturierte Informationen wie Infoboxen, Kategorisierungsinformationen, Bilder, Geo-Koordinaten, usw. Diese Strukturinformationen sind auch in den relationalen Datenbanken vorhanden, in denen Wikipedia einige Informationen cached und von denen regelmäßig Dumps veröffentlicht werden. Die Beziehungen zwischen den Daten, die in der Struktur der Datenbanktabellen abgebildet sind, sowie Informationen aus den Artikeln und den Infoboxen werden in RDF umgewandelt. Weitere Verarbeitungsschritte garantieren die Qualität der gewonnenen Daten. Die DBpedia Datenkollektion umfasst am Ende unter anderem Personen, Plätze, Musikalben, Filme, Links zu anderen Webseiten und Wikipedia sowie YAGO Kategorien. Jede der Resourcen wird durch eine eindeutige URI der Form: **http://dbpedia.org/resource/*Name***

[6] `http://dbpedia.org`
[7] `http://wikipedia.org`

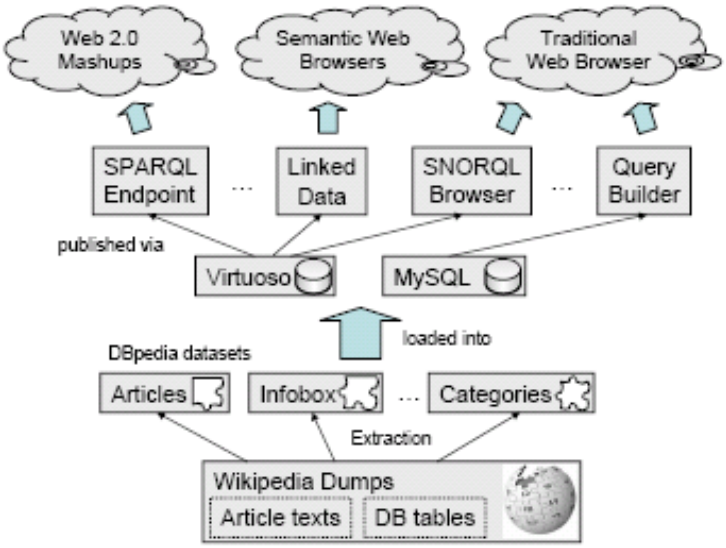

Abbildung 5: Ein Überblick über die DBpedia-Komponenten

beschrieben, wobei *Name* von der URL des als Quelle dienenden Wikipedia-Artikels stammt, die die Form **http://en.wikipedia.org/wiki/*Name*** hat.
Diese Daten sind über verschiedene Mechanismen zugänglich. Ein Mechanismus ist so genannte `Linked Data`. Dabei werden die RDF-Resourcen über das HTTP-Protokoll über eine URL erreichbar, die ihrer URI entspricht. Um also Informationen über die Resource **http://dbpedia.org/resource/Angela_Merkel** zu sehen, kann man genau diese URI in einen traditionellen Webbrowser oder aber einen Semantic Webbrowser eingeben. Je nach benutztem Betrachtungsmedium erscheinen Eigenschaften dieser Resource und Beziehungen zu anderen Resourcen, die über Links aufgerufen werden können.
Einen weiteren Mechanismus bietet das SPARQL Protokoll. DBpedia bietet einen öffentlichen SPARQL-Endpoint an, über den die Informationen mittels des SPARQL Protokolls abgerufen werden können. Dabei wird der Virtuoso Universal Server genutzt. Um den Dienst vor Überlastung zu schützen ist er nur mit einigen Einschränkungen zugänglich, Anfragen, die die gesamte Datenmenge anfordern, werden abgewiesen, es werden immer nur gleichzeitig 1000 Ergebnisse angezeigt.[2]
Ein dritter Weg sind RDF-Dumps von Datenkollektionen im N-Triple[8] Format. Diese können auf der DBpedia Webseite heruntergeladen werden.

DBpedia ist mit anderen Datensätzen verlinkt, um dem Nutzer weitere Informationen über Resourcen zugänglich zu machen. RDF-Links machen es möglich, von einer Datenkollekti-

[8] `http://www.w3.org/TR/rdf-testcases/#ntriples`

on zur nächsten zu navigieren. Dies ist sowohl für menschliche Nutzer als auch für Maschinen möglich. Ein paar Beispiele für Datensätze, mit denen DBpedia verlinkt ist, sind: US Census[9], Project Gutenberg[10], Geonames[11] uvm. Diese Links können z.B. erstellt werden mit Tripeln wie diesem `<http://dbpedia.org/resource/Busan> owl:sameAs <http://sws.geonames.org/1838524>`.
Ein weiterer verlinkter Datensatz ist YAGO[12], eine riesige semantische Wissensbasis. Es beinhaltet eine Klassenhierarchie, welche auf WordNet[13] aufbaut. Daher besitzt die YAGO Hierarchie eine Oberklasse `Entity`, von der alle weiteren Klassen abgeleitet sind, die dann wiederum einen gerichteten azyklischen Graphen aufspannen. Es gibt dabei über 150,000 verschiedene Klassen. Annähernd jeder Resource in DBpedia sind eine oder mehrere YAGO-Klassen zugeordnet. Diese Klassen werden über eine URI identifiziert und besitzen außerdem ein Menschen-lesbares Label. Da sich das Klassensystem von WordNet ableitet, gibt es viele sehr abstrakte Klassen, für die es keine passenden Resourcen gibt, es handelt sich um eine sehr tiefe Hierarchie([20]und [19]).

2.7 Das Lernproblem in OWL

In diesem Abschnitt beschreibe ich kurz das Lernproblem in OWL. Der Prozess des Lernens in der Logik, das heisst eine logische Erklärung für gegebene Daten zu finden, wird auch *induktives Schlussfolgern* genannt. Das Hintergrundwissen ist eine Wissensbasis $\mathcal{K}$. Das Ziel ist es, eine Definition für eine Klasse zu finden, die wir `Target` nennen. Daher sind Beispiele von der Form $\texttt{Target}(a)$, wobei a ein Individuum ist. Wir suchen also eine Klassendefinition der Form $\texttt{Target} \equiv C$ so, dass wir die Wissensbasis um diese Definition erweitern können. Sei $K' = \mathcal{K} \cup \{\texttt{Target} \equiv C\}$ diese erweiterte Wissensbasis. Wir wollen, dass die positiven Beispiele daraus folgen, das heisst $\mathcal{K}' \models E^+$, und die negativen nicht daraus folgen, das heisst $\mathcal{K}' \not\models E^-$.

Beim induktiven Lernen ist ein Generierungs- und Test-Ansatz üblich. Verschiedene Klassenbeschreibungen[14] werden in einem Lernprozess getestet, wobei jede mittels eines OWL Reasoners bewertet wird. Elegante Algorithmen berücksichtigen die Ergebnisse dieser Bewertungen, um weitere aussichtsreiche Beschreibungen vorzuschlagen. Auf diese Art und Weise kann Lernen als ein Suchprozess im Raum der Beschreibungen angesehen werden. Eine natürliche Idee ist es, eine Ordnung in diesem Suchraum (der Menge aller Klassendefinitionen) einzuführen und Operatoren zu nutzen, um ihn zu durchlaufen. Diese Strategie ist bekannt beim ILP (Inductive Logic Programming), wo Refi-

[9] `http://www.census.gov/`
[10] `http://www.gutenberg.org/`
[11] `http://www.geonames.org/`
[12] `http://www.mpi-inf.mpg.de/~suchanek/downloads/yago/`
[13] `http://wordnet.princeton.edu/`
[14] `http://www.w3.org/TR/owl-ref/#ClassDescription`

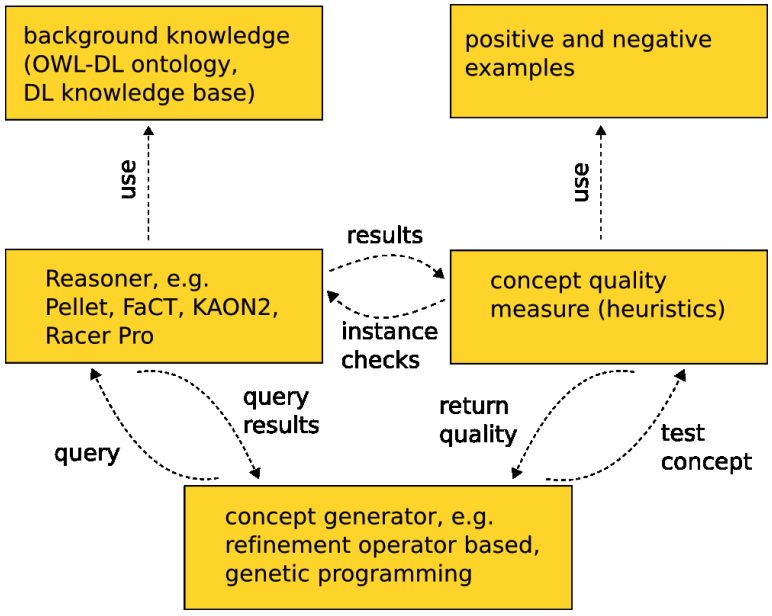

Abbildung 6: Generierung- und Test-Ansatz beim induktiven Lernen

nement Operatoren weitverbreitet sind, um Hypothesen zu finden.

Formal, wenn S die Menge von OWL Klassendefinitionen ist, können wir den quasi-geordneten Raum $(S, \sqsubseteq)$ ($\sqsubseteq$ bezeichnet Subsumption) betrachten. Ein *downward (upward) Refinement Operator* ρ ist eine Abbildung von S zu 2^S, so dass für irgendein $C \in S$ gilt $C' \in \rho(C)$ impliziert $C' \sqsubseteq C$ ($C \sqsubseteq C'$). C' wird *Spezialisation* (*Generalisation*) von C genannt.

Diese Idee wurde genutzt, um einen top-down Refinement Operator basierten Algorithmus zu erstellen. Das heißt, die erste Beschreibung, die getestet wird, ist die allgemeinste Beschreibung (`owl:Thing`, $\top$ bezeichnet in DL Syntax), welche dann abgebildet wird auf eine Menge spezieller Beschreibungen mittels eines downward Refinement Operators. Naturgemäß kann der Refinement Operator wieder auf die gewonnenen Beschreibungen angewandt werden, wodurch ein Suchbaum aufgespannt wird. Der Suchbaum kann beschnitten werden, wenn man eine unvollständige Beschreibung erreicht, das heisst eine Beschreibung, die nicht alle positiven Beispiele abdeckt. Dies kann getan werden, da der downward Refinement Operator garantiert, dass alle Verfeinerungen dieser Beschreibung auch nicht alle positiven Beispiele abdecken und daher keine Lösungen des Lernproblems darstellen können. Ein Beispiel für einen Pfad in einem Suchbaum, der von einem downward Refinement Operator aufgespannt wurde:

$$\top \rightsquigarrow \texttt{Person} \rightsquigarrow \texttt{Person} \sqcap \texttt{nimmtTeilAn.Veranstaltung}$$
$$\rightsquigarrow \texttt{Person} \sqcap \texttt{nimmtTeilAn.Konferenz}$$

Das Herz einer solchen Lernstrategie ist es, einen passenden Refinement Operator und eine passende Suchheuristik, die entscheidet welche Knoten im Suchbaum expandiert werden sollten, zu definieren. Der Refinement Operator für den vorliegenden Algorithmus kann in [13] gefunden werden und baut auf solide theoretische Grundlagen [12]. Es wurde gezeigt, dass es der beste erreichbare Operator ist, wenn man eine Menge an Eigenschaften (die hier nicht näher betrachtet werden) betrachtet, die die Qualität von Refinement Operatoren beurteilen.

2.8 DL-Learner

Der DL-Learner[15] ist ein Tool, um Konzepte in der Beschreibungslogik anhand von positiven bzw. negativen Beispielen zu lernen. Genauso können Klassen in OWL anhand vorgegebener Objekte gelernt werden. Das Ziel ist es, Nutzer dabei zu unterstützen, weiteres Wissen aus der vorhandenen Datenbasis zu ziehen und dieses implizit vorhandene Wissen zu nutzen. Damit bietet der DL-Learner also Lösungen für das Lernproblem an, das heisst bei gegebenen positiven und negativen Beispielen für ein noch nicht näher definiertes Konzept sucht er eine Konzeptbeschreibung, aus der zusammen mit der Datenbasis alle positiven und keines der negativen Beispiele folgt.
Dabei geht der DL-Learner so vor, dass neue Konzeptbeschreibungen generiert werden. Deren Qualität wird anschließend gemessen. Dabei wird geschaut, wie viele positive Beispiele aus der Kombination der Wissensbasis und dieser Konzeptbeschreibung folgen und wie viele negative. Je mehr positive und je weniger negative, desto höher ist die Qualität. Anhand der Qualität wird entschieden, welche neuen Konzeptbeschreibungen generiert werden. Diese neuen Definitionen werden nun wieder bewertet. Das geht so lange, bis eine Konzeptbeschreibung alle positiven und keines der negativen Beispiele abdeckt, denn dann ist eine Lösung gefunden. Der Algorithmus kann im Zweifelsfall ewig laufen wenn es keine Lösung gibt.[11]
Um sehr flexibel bei der Lösung des Problems sein zu können, verwendet der DL-Learner ein kom-

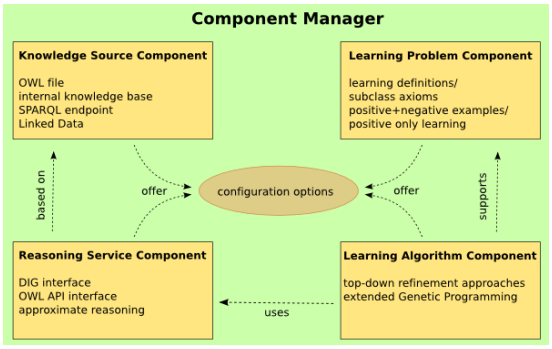

Abbildung 7: DL-Learner Komponenten-Modell

[15] http://dl-learner.org/Projects/DLLearner

ponentenbasiertes Modell. Es gibt vier verschiedene Typen von Komponenten. Die Wissensbasis, der Reasoning Service, der zur Beurteilung der Qualität einer Konzeptbeschreibung gebraucht wird, das Lernproblem und der Lernalgorithmus. Für jeden Typ gibt es mehrere Komponenten und es ist möglich angepasst an das zu lösende Problem verschiedene Kombinationen dieser Komponenten zu benutzen.
Außerdem kann jede Komponente ihre eigenen Konfigurationsoptionen haben. Um eine eigene Komponente zu definieren und zu benutzen, muss man sie nur von der richtigen Klasse ableiten, den Namen der Klasse dieser Komponente in der `components.ini` bekannt machen und die benötigten Konfigurationsoptionen definieren. Um den restlichen Code nicht anpassen zu müssen und eine neue Komponente sofort nutzen zu können, müssen diese Konfigurationsoptionen einen bestimmten Typ haben:

- boolean
- String (ein Menge an erlaubten Werten kann definiert werden)
- int (ein minimaler und maximaler Wert kann definiert werden)
- double (ein minimaler und maximaler Wert kann definiert werden)
- eine Menge von Strings
- eine Liste von String Tupeln

Außer diesen Kernfunktionalitäten gibt es noch verschiedene Interfaces, um diese Funktionen aufzurufen, einige Tools, die die DL-Learner Algorithmen nutzen, sowie ein paar Skripte, die die Arbeit mit dem DL-Learner vereinfachen.

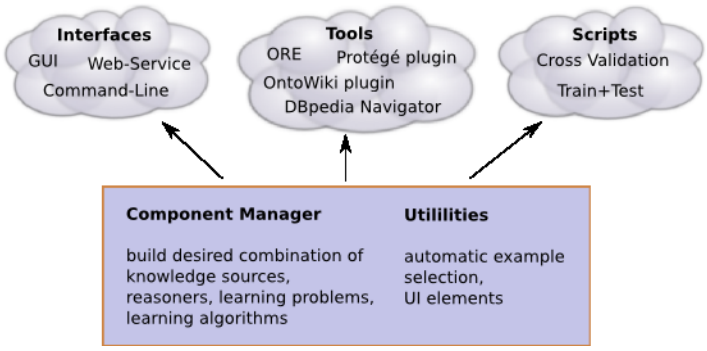

Abbildung 8: DL-Learner Feature

3 Aufbau des DBpedia Navigators

3.1 Architektur

Der DBpedia Navigator zeigt verschiedene Sichten auf die DBpedia Daten und stellt verschiedene Wege zur Verfügung, durch diese Daten zu browsen. Er ist aufgebaut aus dem eigentlichen DBpedia Navigator Client, mit einer Anbindung zu einer Datenbank, zu einer Instanz des DL-Learners mit einem OWL-Reasoner sowie einem DBpedia-SPARQL-Endpoint. Alle diese 4 Komponenten können räumlich getrennt auf verschiedenen Servern laufen oder zusammen auf einem. Dies ermöglicht maximale Flexibilität bezüglich der Wahl der Serverstandorte. Der DBpedia Navigator Client ist in PHP geschrieben. Die Inhalte werden dynamisch erzeugt, dazu wird jede Funktion über Ajax[16]-Anfragen aufgerufen, wodurch immer die selbe Webseite angezeigt wird, deren Inhalte dynamisch nachgeladen werden. Es ist allerdings auch möglich die verschiedenen Funktionen nicht über die GUI aufzurufen, sondern über ein REST[17]-Interface. Jeder Zustand der Anwendung kann in eine URL abgebildet werden, bei deren Aufruf die letzte Aktion, die zu dem Zustand geführt hat, wiederholt und die Session entsprechend angepasst wird. Näheres dazu in Kapitel 4.12 Diese Funktionen kommunizieren nun mit den einzelnen Komponenten, um die benötigten Daten abzurufen. Die Datenbank enthält eine ausgewählte Teilmenge der DBpedia Daten, die für alle Suchanfragen relevant ist. Die Anwendung kommuniziert über eine geeignete PHP-Schnittstelle mit der Datenbank, es schickt die Suchanfragen als SQL-Anfragen und bereitet die Antwort für den Nutzer auf.
Um Artikel anzuzeigen, müssen die Daten vom SPARQL-Endpoint geladen werden. Mit dem DBpedia-SPARQL-Endpoint findet allerdings keine direkte Kommunikation statt. Stattdessen werden SPARQL-Anfragen, die die erforderlichen Daten abrufen sollen, über eine Webservice-Schnittstelle an den DL-Learner geschickt. Dazu wird eine lokale WSDL[18]-Datei verwendet, die den Webservice und dessen Schnittstelle beschreibt. Die SPARQL-Anfragen werden dann mittels des JENA[19]-Frameworks an den Endpoint gesendet oder falls die Resultate im Cache liegen sollten, daher geholt. Das JENA-Framework bietet außerdem Methoden an, die Ergebnisse in verschiedenen Formaten zurückzugeben. Da es sich um ein besonders kompaktes Format handelt, werden die Daten in JSON[20]-Notation zurückgegeben. Lernprobleme werden vom Navigator ebenfalls an den DL-Learner geschickt. Dieser holt die für die Lösung dieser Probleme benötigten Daten wieder von dem DBpedia-SPARQL-Endpoint oder aus dem Cache. Für das Lernen wird sowohl die YAGO-Klassenhierarchie als auch weitere Eigenschaften von Resourcen genutzt. Zum Lernen wird ein OWL-Reasoner benötigt, der über eine DIG/OWL API angesprochen wird. Es werden Lösungen in Form von Konzepten

[16] http://de.wikipedia.org/wiki/Ajax_(Programmierung)
[17] http://de.wikipedia.org/wiki/Representational_State_Transfer
[18] http://www.w3.org/TR/wsdl
[19] http://jena.sourceforge.net/
[20] www.json.org/

für das Problem berechnet und an den Navigator zurückgeschickt.

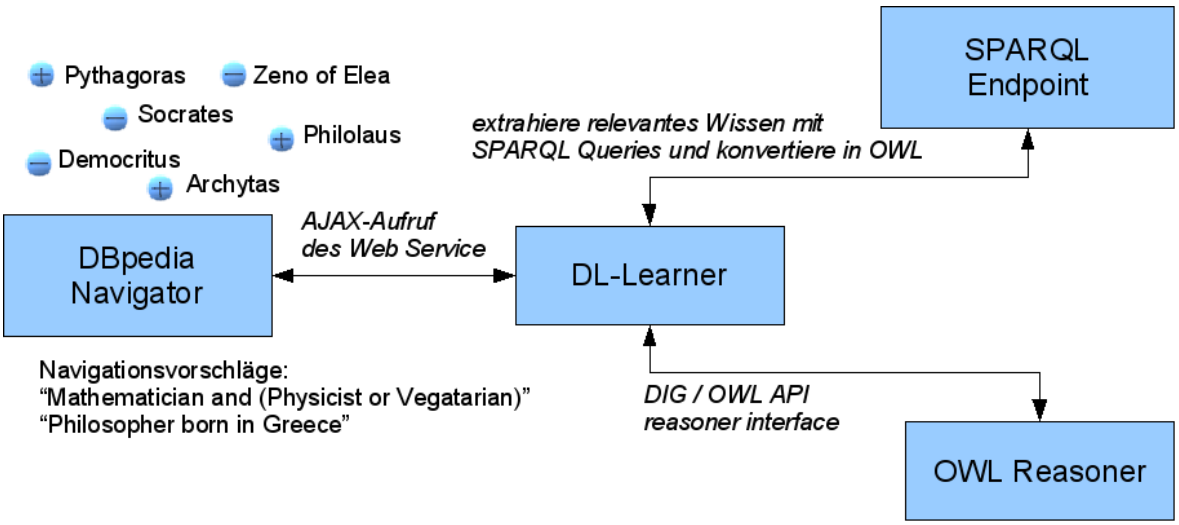

Abbildung 9: Architektur des DBpedia Navigators

3.2 Erweiterungen des DL-Learners

Der DL-Learner wird sowohl genutzt, um Wissen aus dem DBpedia Datensatz zu extrahieren, als auch, um auf der Basis dieses Wissens mittels ausgewählter Beispiele Lernprobleme zu lösen und so Navigationsvorschläge zu generieren. Dazu war es nötig, den DL-Learner dahingehend zu erweitern, dass ein SPARQL-Endpoint als Wissensbasis verwendet werden kann. Außerdem waren Erweiterungen nötig, um die vom Endpoint erhaltenen Daten zu manipulieren und in geeigneter Weise an den DBpedia Navigator weiterzugeben. Für die Weitergabe der Daten war es nötig, die Webservice-Schnittstelle zu erweitern. Es mussten Methoden zum Zugriff auf die SPARQL-Wissenskomponente hinzugefügt sowie bestehende Methoden für den Lernvorgang angepasst werden. Erweiterungen, die die Generierung von Navigationsvorschlägen betreffen, werden im Kapitel 5 besprochen.
Der DL-Learner ist aus verschiedenen Komponenten und Komponententypen aufgebaut. Der `Knowledge Source`-Komponententyp repräsentiert eine Wissensquelle, aus der Wissen gewonnen werden kann, um dann mit Hilfe dieses Wissens Lernprobleme zu lösen. Die Wissensquelle für den DBpedia-Navigator ist die DBpedia Wissensbasis, auf die mittels eines SPARQL-Endpoints zugegriffen wird. Die Klasse `SparqlKnowledgeSource` repräsentiert solch eine Wissensquelle, auf die mittels eines SPARQL-Endpoint zugegriffen werden kann. Die Komponentenstruktur des DL-Learners garantiert auch dieser `Knowledge Source` zahlreiche Konfigurationsoptionen, über die einstellbar ist, wie auf das Wissen in dem repräsentierten Endpoint zugegriffen wird. Sie wird sowohl genutzt, um das für ein gestelltes Lernproblem benötigte Wissen abzurufen, als auch, um beispiels-

weise das Wissen für einen gesuchten Artikel im DBpedia-Navigator zu erlangen. Dies geschieht, indem eine SPARQL-Anfrage an den Endpoint geschickt wird.
Die Anfrage wird durch die Klasse `SparqlQuery` repräsentiert. Sie enthält die eigentliche Anfrage als auch den Endpoint, der durch ein Objekt der Klasse `SparqlEndpoint` repräsentiert wird. Das Senden der SPARQL-Anfrage sowie die Verarbeitung des Ergebnisses erfolgt mit Hilfe des JENA-Frameworks. Die Ergebnisse können im JENA spezifischen ResultSet Format, als XML String oder als JSON String abgerufen werden.
Die Klasse `SparqlEndpoint` repräsentiert eine Sparql-Endpoint-Konfiguration. Dabei werden die URL des Endpoints, sowie die `default Graph URIs` und die `named Graph URIs` festgelegt. Einige vorkonfigurierte Endpoints inklusive des DBpedia Endpoints sind auswählbar.
Um nicht die gleichen SPARQL-Anfragen mehrmals an einen Endpoint stellen zu müssen, kann die `Knowledge Source`-Komponente so konfiguriert werden, dass ein Cache benutzt wird. Die Klasse `Cache` repräsentiert diesen Cache. In einem konfigurierbaren Verzeichnis wird jede SPARQL-Anfrage und ihr Ergebnis in eine Datei gespeichert. Der Name der Datei ist ein Hash der Anfrage, das Ergebnis der Anfrage wird mittels JSON-Serialisation in die Datei geschrieben. Außerdem wird ein `Timestamp` hinzugefügt. Nach einer konfigurierbaren Zeit wird der Cache Eintrag ungültig und wird nicht mehr benutzt.
Außerdem gibt es verschiedene Klassen, die bestimmte öfter wieder kehrende Aufgaben erleichtern. Mit der Klasse `SparqlQueryMaker` ist es möglich auf einer abstrakteren Ebene SPARQL-Anfragen zu erstellen, ohne sie selbst formulieren zu müssen. Durch Angabe weniger Optionen ist es so möglich, verschiedene Anfragen mit Filtern zu erstellen. Die Klasse `SPARQLTasks` bietet weitere Methoden bestimmte Aufgaben, die das Abfragen des SPARQL Endpoints erfordern, zu erledigen, wie z.B. Super- oder Subklassen von angegebenen Klassen zu finden.
Weiterhin existieren einige Klassen, die die vom Lernalgorithmus berechneten Konzepte in verschiedener Weise verarbeiten. Dabei werden die Konzepte rekursiv durchlaufen, um die Struktur der Konzepte ganz erfassen und die Struktur der resultierenden Konstrukte darauf abstimmen zu können. Beispielsweise wandelt die Klasse `NaturalLanguageDescriptionConvertVisitor` das Konzept, welches in Beschreibungslogik vorliegt, in eine natürlichsprachliche Form um. Diese kann dann selbst von einem Endnutzer des Navigators, der in der Regel keine Beschreibungslogiken kennt, verstanden und von ihm als Navigationsvorschlag genutzt werden (siehe Kapitel 5.4). Die Klasse `SparqlQueryDescriptionConvertVisitor` dagegen setzt aus einem Konzept eine SPARQL-Anfrage zusammen. Diese SPARQL-Anfrage wird dann an den SPARQL-Endpoint geschickt, der die Instanzen dieses Konzepts liefert. Dies wird benötigt, um mittels der Navigationsvorschläge Artikel zu finden(siehe Kapitel 5.5).
Außer der SPARQL-Komponente wurden verschiedene Webservice-Methoden zum DL-Learner hinzugefügt, die auf die Komponente zugreifen und verschiedene Aufgaben erfüllen. Es gibt zwei Metho-

den, um eine SPARQL-Anfrage abzusetzen, eine in einem eigenen *Thread*, die andere ohne eigenen Thread. Es ist möglich, die in einem eigenen Thread startende Methode abzufragen, ob sie noch läuft sowie sie zu unterbrechen. Das ist vor allem für kostenintensive Anfragen nützlich, denn ein Nutzer will nicht immer die volle Zeit warten und in der Lage sein, solche langen Anfragen abzubrechen. Die Ergebnisse der Anfragen können in verschiedenen Formaten abgerufen werden, als XML- oder JSON-String. Beide Formate eignen sich gut als Austauschformate, der JSON-String ist allerdings kürzer und kann sehr leicht in ein assoziatives Array in PHP umgewandelt werden, mit dem man sehr einfach arbeiten kann.
Außer diesen Methoden wurden weitere zur Verarbeitung der Konzepte hinzugefügt, die von dem Lernalgorithmus als Navigationsvorschläge produziert werden. Es ist möglich verschiedene Parameter der Konzepte abzurufen, wie z.B. Konzepttiefe oder Stelligkeit. Außerdem kann man die schon weiter oben beschriebenen Klassen `NaturalLanguageDescriptionConvertVisitor` und `SparqlQueryDescriptionConvertVisitor` dazu benutzen, eine natürlichsprachliche Form der Konzepte zu erhalten oder eine SPARQL-Anfrage, die Instanzen der Konzepte liefert. Als letztes gibt es eine Methode, um negative Beispiele aus positiven zu erzeugen, dies wird aber in einem späteren Kapitel näher erläutert.

3.3 Aufbau der Datenbank

Wenn man in einer großen Datenmenge etwas sucht, möchte man im Allgemeinen nicht einfach alle Suchergebnisse ungeordnet dargestellt bekommen, sondern gerade bei mehreren hundert Ergebnissen eine geordnete Teilmenge der Resultate. Um aber eine geordnete Artikelmenge (ein Artikel ist hierbei eine visuelle Umsetzung der zu einer Resource existierenden Daten, siehe Kapitel 4.5) anzeigen zu können, braucht es eine Ordnung auf diesen Artikeln. Im ursprünglichen DBpedia-Datensatz existiert eine solche Ordnung nicht, aber es existieren Pagelinks, also Links von einem Artikel zu einem anderen. Diese können verwendet werden, um einen Pagerank zu berechnen, man kann also jedem Artikel bezüglich seiner Verlinkungsstruktur ein Gewicht zuordnen.
Ein Ansatz wäre nun, alle Artikel, die einem bestimmten Suchkriterium entsprechen, über den SPARQL-Endpoint abzurufen und anschließend mittels des Pageranks zu ordnen. Dieser Ansatz hat allerdings den Nachteil, dass man zwar eine geordnete Ergebnismenge vorliegen hätte, man aber trotzdem alle Ergebnisse abrufen müsste. Aus diesem Grund wählte ich einen Ansatz, in dem ich die Artikel (bzw. ihre URIs) mit dem Pagerank verband, um so nur eine geordnete Teilmenge der Suchergebnisse abrufen zu können. Eine relationale Datenbank ist dabei ein überaus geeignetes Speichermedium für diese Informationen, da sie mit SQL über eine sehr flexible Anfragesprache verfügt, die das Ordnen und Limitieren von Suchergebnissen erlaubt, sowie über eingebaute Mechanismen für z.B. eine Volltextindizierung verfügt.
Für die Entwicklung wurde eine MySQL-Datenbank genutzt, es kann theoretisch aber jede Daten-

bank mit einer PHP-Schnittstelle verwendet werden. Die Datenbank enthält 4 Tabellen. Die Tabelle `rank` enthält alle Artikel mit ihrem Pagerank und einem Label. Der Pagerank berechnet sich als die Summe aller eingehenden Links für diesen Artikel. Es ist natürlich möglich einen genaueren Pagerank zu berechnen, der nicht nur die Anzahl der eingehenden Links addiert, sondern diese eingehenden Links auch noch mit dem Pagerank des Artikels, von der dieser Link kam, gewichtet. Auf Grund der großen Datenmenge und meiner beschränkten Rechenkapazität verzichtete ich allerdings auf diese größere Genauigkeit. Für das Label der Artikel ist ein Volltextindex eingerichtet. So ist eine schnelle Suche von Artikeln auf Basis des Labels möglich.
Die Tabelle `articlecategories` enthält die Zuordnung aller Artikel zu YAGO-Klassen. Dabei kann ein Artikel zu mehreren Klassen gehören. Um die Suche nach Artikeln, die einer bestimmten Klasse angehören zu vereinfachen, ist auch in dieser Tabelle für jeden Artikel die Anzahl der eingehenden Links gespeichert. Es handelt sich hierbei um redundante Informationen, aber es beschleunigt die Suche. Die Tabelle `categories` enthält alle Klassen mit ihren Labels. Dies ermöglicht eine natürlichsprachliche Anzeige von Klassen. Die vierte Tabelle `classhierarchy` enthält die Beziehungen der Yago-Klassen untereinander, also die Ober-Unter-Klassen-Beziehungen. Mit diesen vier Tabellen ist es möglich eine Label-bezogene und eine Klassen-bezogene Suche nach Artikeln sowie ein Browsen in der YAGO-Klassenhierarchie zu implementieren, ohne auf den SPARQL-Endpoint zugreifen zu müssen. Dies führt zu einer sehr guten Laufzeit dieser Funktionen, welche ohne die Datenbank in einem nicht akzeptablen Bereich lag.
Das folgende ER-Diagramm modelliert die Datenbank, dabei wurden leichter verständliche Namen für Entitäten, Relationen und Attribute gewählt, die entsprechende Bezeichnung in der Datenbank steht in Klammern dahinter.

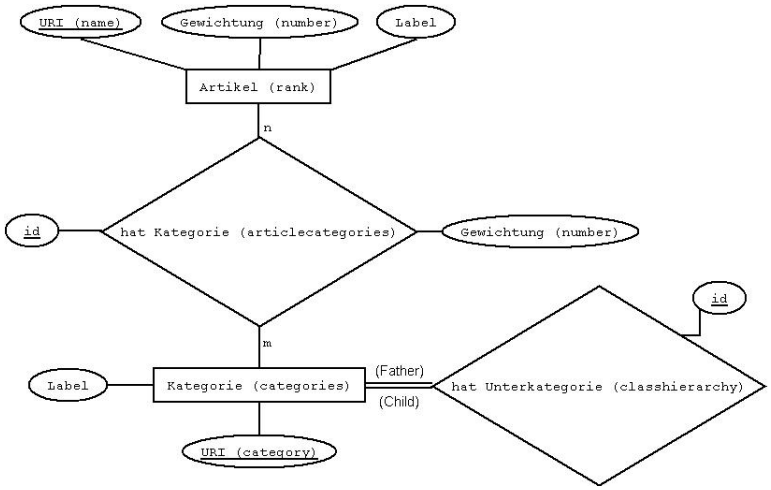

Abbildung 10: ER-Diagramm des Datenbankschemas

4 Der DBpedia Navigator Client

4.1 Aufbau des Navigators

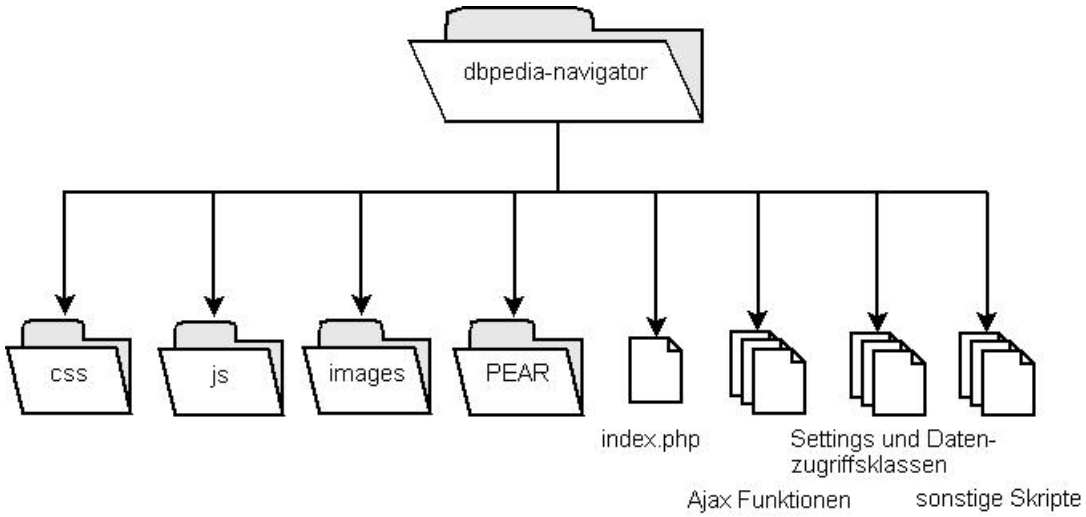

Abbildung 11: Schematische Dateistruktur

Der DBpedia Navigator Client ist eine in PHP geschriebene Webanwendung. Sie besteht aus einigen Skripten, JavaScript- und CSS-Dateien, Bildern und Icons, sowie PEAR-Komponenten. Die `index.php` bildet dabei das Grundgerüst der Webseite. Sie enthält die Eingabemasken und Steuerelemente, über die der User die Funktionen der Anwendung aufrufen kann, sowie die einzelnen Container, in welche durch Ajax-Funktionsaufrufe die Inhalte eingefügt werden. Der Aufruf der Ajax-Funktionen geschieht über JavaScript-Funktionen, die in der `ajax.js` gesammelt sind. Diese JavaScript-Funktionen rufen wiederum PHP-Skripte auf, die nach dem Schema `ajax_<Funktionsname>.php` benannt sind. So enthält das Skript `ajax_search.php` beispielsweise die Suchfunktion.
Der Zugriff auf die Daten erfolgt mittels zweier Klassen in gleichnamigen Skripten, die diesen Zugriff auf die Datenquellen abstrahieren. Die `DatabaseConnection`-Klasse repräsentiert die Verbindung zu der Datenbank. Dabei ist die Klasse so geschrieben, dass von der konkret benutzten Datenbank abstrahiert wird, um größtmögliche Flexibilität zu ermöglichen. Die `DLLearnerConnection`-Klasse repräsentiert die Verbindung zum DL-Learner über einen Webservice. Sie kapselt alle Funktionen, die eine Kommunikation mit dem Webservice benötigen. Die Kommunikation erfolgt mit Hilfe der nativen SOAP-Erweiterung von PHP.
Die aktuellen Einstellungen des Navigators sind in der `Settings.php` niedergelegt und können so einfach an die individuelle Anwendungsumgebung angepasst werden. Dabei ist es möglich folgende

Einstellungen zu verändern:

- die URI des DLLearner Webservice
- die URI des DBpedia-Endpoint (lokale oder entfernte Endpoints)
- die maximale Bearbeitungszeit einer SPARQL-Anfrage, die über den Webservice abgesetzt wird
- die maximale Bearbeitungszeit eines Lernvorgangs
- die verwendete Sprache (zur Zeit nur Englisch)
- Nutzung eines Caches
- Datenbankeinstellungen (Datenbanktyp, Serveradresse, Nutzer, Passwort, Datenbankname)
- beim Lernen zu ignorierende Konzepte und Rollen

Sonstige Hilfsfunktionen, die von verschiedenen Skripten aufgerufen werden können, sind in der `helper_functions.php` enthalten. Dies sind Funktionen wie der Aufbau einer Tagcloud, die Umwandlung von Zeichen in Unicode-Kodierung zu HTML-Entities, die Entwicklung von URIs aus Labeln oder die Zusammenstellung von verschiedenen Resultatsdarstellungen. Einige JavaScript-Funktionen, mit denen im wesentlichen die Darstellung auf dem Client des Users manipuliert wird, sind in der `navigator.js` enthalten. Dies sind z.B. eine Funktion zum Darstellen eines Ortes auf einer Google Map[21] oder auch eine Funktion zum Navigieren durch eine Ergebnismenge mittels `durchblättern` mehrerer Seiten.
Außerdem existieren noch einige weitere Skripte, die vor allem für die Installation bzw. die Entwicklung von Bedeutung sind, zum resetten der Session und Neuladen der WSDL-Datei vom Webservice (`rebuild.php`), zur Erschaffung des Datenbankschemas (`database.sql`), zur Füllung der Datenbank mit Daten und der Berechnung des Pageranks aus vorliegenden DBpedia Datensätzen (`CalculatePageRank.java`).
Als letztes existiert noch eine .htaccess-Datei, die erstens alle Aufrufe nicht vorhandener Dateien auf die `index.php` lenkt und vor allem ein REST-Interface realisiert.

4.2 Funktionen

Das Ziel des DBpedia Navigators ist es, durch die DBpedia Wissensbasis auf strukturierte und intelligente Weise zu navigieren. Um dies zu erreichen, bietet der Navigator zahlreiche Funktionen, die

[21] `http://code.google.com/apis/maps/`

ich im Folgenden beschreiben werde. Doch zuerst ein Überblick über die gesamte GUI des DBpedia Navigator Clients.

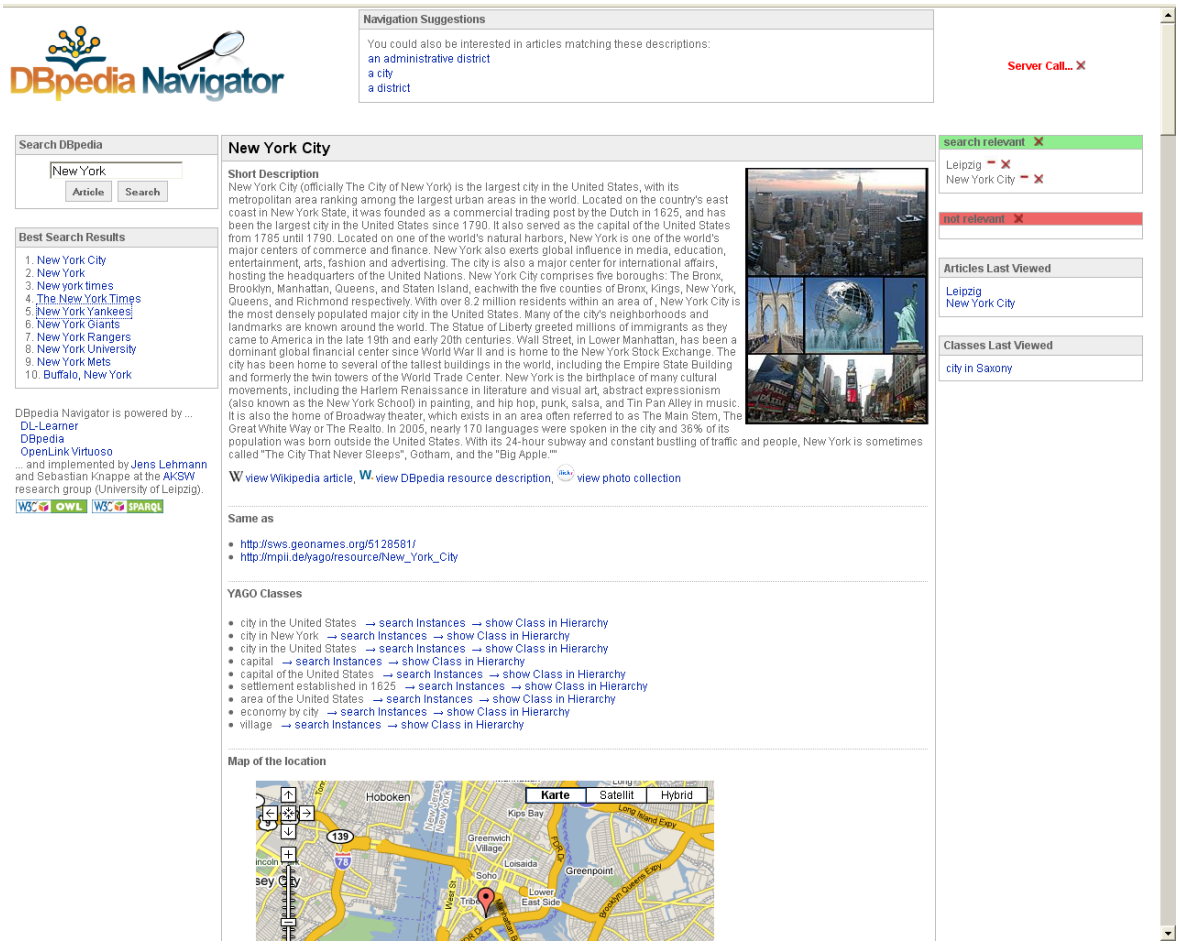

Abbildung 12: Die GUI des DBpedia Navigators

Der obere Teil zeigt das Logo sowie Navigationsvorschläge, die auf Basis der suchrelevanten und nicht relevanten Artikel generiert wurden. Die linke Sidebar enthält die Suchbox, eine Box mit den zehn letzten Suchergebnissen, sowie ein Impressum. Der zentrale Bereich enthält die Box, die die eigentlichen Inhalte zeigt, Artikel, Klassen, Suchergebnisse. Die rechte Sidebar enthält die Boxen mit den suchrelevanten Artikeln, nicht relevanten Artikeln und die zuletzt angeschauten Artikel und Klassen. Im untere Bereich wird bei jeder Aktion eine URL generiert, die den aktuellen Zustand der Anwendung abbildet und mit der es möglich ist, diesen Zustand wiederherzustellen. Zusammengefasst bietet die Anwendung folgende Funktionen:

- eine Artikelsuche

- eine Artikelansicht
- eine Suchresultatsansicht
- eine Klassenansicht
- eine Liste der zehn letzten Suchresultate
- suchrelevante und nicht relevante Artikel
- eine Liste der zuletzt angeschauten Artikel
- eine Liste der zuletzt angeschauten Klassen
- eine Funktionsunterbrechungsfunktion für zeitintensive Anfragen
- vom DL-Learner generierte Navigationsvorschläge
- ein REST Konzept mit der Möglichkeit, für Zustände der Anwendung eine entsprechende URL zu generieren

Im Folgenden werde ich die einzelnen Funktionen und die damit zusammenhängenden Teile der GUI näher beschreiben.

4.3 Die Artikelsuche

Beschreibung Um nach einem Artikel zu suchen gibt man in das Textfeld in der Box mit dem Titel `Search DBpedia` auf der linken Seite einen Suchbegriff ein. Es gibt dabei zwei Ausprägungen der Artikelsuche. Einerseits ist es möglich genau einen Artikel zu suchen, der den Titel trägt, der vom Benutzer eingegeben wurde, andererseits kann man nach Artikeln suchen, die das eingegebene Stichwort enthalten. Für jede Art der Suche gibt es einen Button, `Article` für die Suche nach einem Artikel, `Search` für die Suche nach mehreren.
Sucht man nach genau einem Artikel wird aus der eingegebenen Wortgruppe eine URI gebildet, in der Form, dass vor die Wortgruppe `http://dbpedia.org/resource/` gesetzt, Leerzeichen durch Unterstriche ersetzt und das ganze URL-kodiert wird. So wird aus dem Suchterm `Angela Merkel` beispielsweise die URI `http://dbpedia.org/resource/Angela_Merkel`. Nun wird eine SPARQL-Anfrage geformt, die alle mit der durch diese URI repräsentierten Resource zusammenhängenden Informationen abruft. Dabei werden auch Weiterleitungen beachtet wie die Struktur der Anfrage zeigt, so dass die Suche nach `Deutschland` automatisch zu `Germany` weitergeleitet würde ($uri steht für die oben gebildete URI):

```
SELECT ?pred ?obj
WHERE {
{<$uri> ?pred ?obj}
UNION
{<$uri> <http://dbpedia.org/property/redirect> ?Conc.
?Conc ?pred ?obj}
}
```

Die Informationen, die so über eine Resource wie beispielweise `Angela Merkel` gewonnen wurden, werden dann zu einer Artikelansicht zusammengefasst. Werden zu einem eingegebenen Begriff keine Informationen gefunden, wird automatisch die Artikelsuche für mehrere Artikel gestartet.
Die Suche nach mehreren Artikeln läuft über die angeschlossene Datenbank. Es existiert ein Volltextindex für die Label aller Artikel. Dieser Index wird nach der eingegebenen Suchphrase durchsucht und nach den Pageranks der gefundenen Seiten geordnet. Die Suche über das Eingabefeld ist auf zehn Ergebnisse beschränkt, in der Suchresultatsansicht (siehe Kapitel 4.6) kann dies allerdings ausgeweitet werden. Die folgende SQL-Anfrage zeigt die Suche beispielhaft für eine MySql-Datenbank ($label ist hierbei der eingegebene Suchbegriff):

```
SELECT name, label
FROM rank
WHERE MATCH (label) AGAINST ('$label')
ORDER BY number DESC
LIMIT 10
```

Typische Anwendungsfälle Die Suche nach einem Artikel ist vor allem dann sinnvoll, wenn man genau weiss, wonach man sucht, bzw. wofür man sich interessiert. Es ist vor allem ein guter Einstiegspunkt für das Navigieren, da man über Links in einem Artikel schnell zu anderen Artikeln oder Klassen von Artikeln kommt und durch deren Auswahl weitere Navigationsvorschläge generiert werden.
Die Suche nach Aritkeln über ihr Label eignet sich für Situationen, in denen man sich z.B. über ein größeres Themengebiet informieren will oder wenn man nicht so genau weiss, wonach man sucht, sondern nur eine Ahnung hat, welche Terme mit der Thematik zu tun haben. Natürlich kann man so auch einfach ein wenig herumstöbern oder schauen, welche Artikel besonders relevant sind.

Abbildung 13: Das Artikelsuchfeld und die Box mit den letzten Suchergebnissen

4.4 Die Liste der zehn letzten Suchresultate

Beschreibung Ebenfalls auf der linken Seite befindet sich eine Liste der letzten Suchresultate, direkt unter dem Artikelsuchfeld (siehe Kapitel 13). Hier werden immer Links zu den zehn relevantesten Artikeln in einer Box angezeigt. Beim Klicken auf die Links werden die entsprechenden Artikel in der zentralen Box angezeigt. Dabei wird die gleiche Funktion wie bei der Suche nach einem Artikel benutzt, nur dass die URI nun nicht mehr zusammengesetzt werden muss, da diese bei der Suche schon vollständig abgerufen wurde.

Typische Anwendungsfälle Aufgrund der Verwendung von Ajax-Funktionsaufrufen funktioniert der Zurück-Button des Browsers nicht. Somit ist es über diesen Button nicht möglich, nachdem man einen Artikel in der Suchresultatsansicht angeklickt hat, zu dem Suchergebnis zurückzukehren. Diese Funktion ermöglicht nun das schnelle Wechseln zwischen den Artikeln, die als letztes gefunden worden sind, ohne diese Suche noch einmal durchführen zu müssen. Sie gleicht das Handicap aus und ermöglicht sogar ein schnelleres Wechseln der Artikel als mit dem Zurück-Button.

4.5 Die Artikelansicht

Beschreibung Die Artikelansicht ist eine Zusammenfassung bzw. Zusammenstellung von Informationen zu einer bestimmten Resource wie z.B. `Leipzig`, die dann einen Wikipedia[22]-artigen Artikel formen. Ein Artikel ist unterteilt in mehrere Abschnitte, die durch gepunktete Linien getrennt sind. Über dem eigentlichen Artikel steht dessen Titel, das Label der URI dieser Resource. Darunter

[22] `http://www.wikipedia.org`

beginnt der erste Abschnitt. Er wird dominiert von einer kurzen Beschreibung der Resource und einem Bild, sollte eines existieren. Dabei werden die verschiedenen möglichen Webadressen eines Bildes danach überprüft, ob es wirklich vorliegt oder ob es sich um einen toten Link handelt. Das Bild besitzt außerdem einen Tooltip, sollte dieser in den DBpedia Daten vorhanden sein. Sollte der Artikel weitergeleitet worden sein, steht über der Kurzbeschreibung die URI des weiterleitenden Artikels. Unter der Kurzbeschreibung sind Links zum korrespondieren Wikipedia-Artikel, zur Beschreibung der Resource auf der DBpedia-Seite und zu einer Fotokollektion. Beim Klick auf einen der Links öffnet sich ein kleines Fenster, so dass man den Artikel im DBpedia Navigator und die angeklickte Seite parallel betrachten kann.

Abbildung 14: Der obere Teil einer Artikelansicht mit der Kurzbeschreibung und Bild, äquivalenten Resourcen auf anderen Seiten und den YAGO-Klassen

Im zweiten Abschnitt finden sich Links zu der gleichen Resource auf anderen Seiten, also Resourcen, die über das Prädikat `owl:sameAs` verbunden sind. Dort wird die Resource in anderen Kontexten dargestellt, beispielweise als Region auf einer Karte mit entsprechenden zugehörigen Informationen[23].

[23] `http://www.geonames.org/`

Im nächsten Abschnitt finden sich die YAGO Klassen, zu denen die entsprechende Resource gehört, also eine `rdfs:subClassOf` Beziehung besitzt. Angezeigt wird dabei das Label dieser Klassen bzw. ein Teil der URI wenn dieses mehr Information birgt (nähere Erklärungen dazu siehe Kapitel 5.4), um eine bessere Lesbarkeit zu erreichen. Hinter jeder Klasse befinden sich zwei Links, die es ermöglichen, nach weiteren Instanzen dieser Klasse zu suchen, also nach weiteren Artikeln oder eine Ansicht dieser Klasse aufzurufen, welche weiter unten beschrieben wird in Kapitel 4.7. Bei der Suche nach Instanzen einer Klasse wird die entsprechende Datenbanktabelle durchsucht und die relevantesten Artikel (dem Pagerank zufolge) ausgewählt.

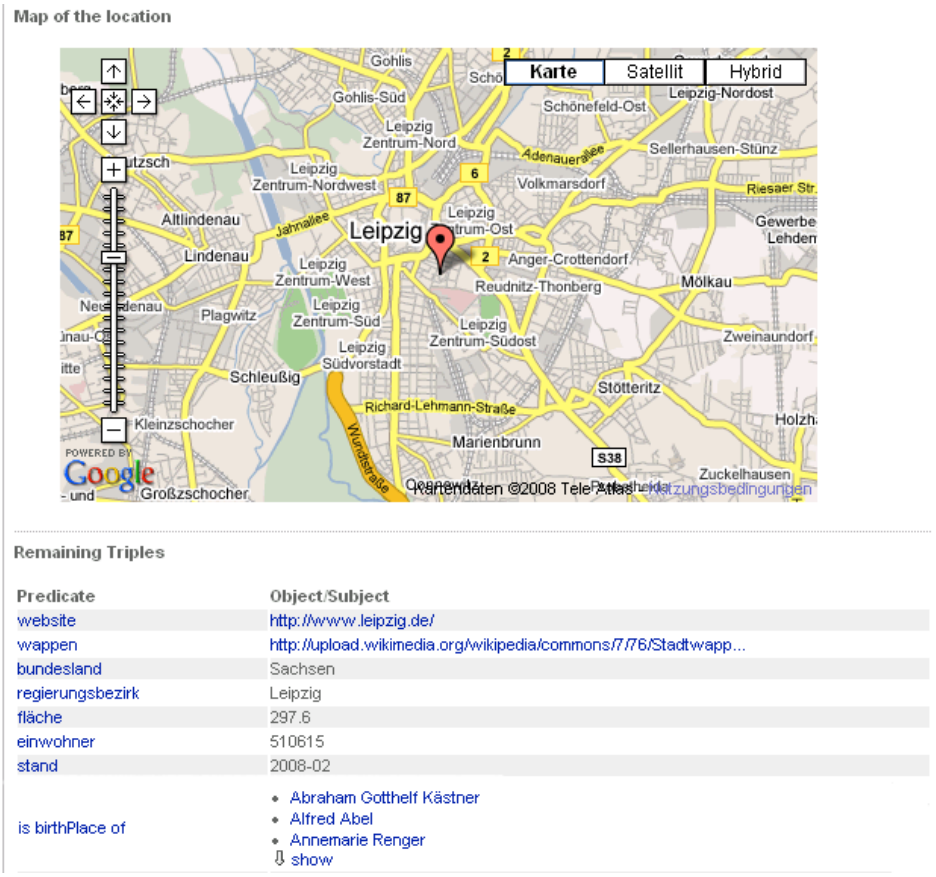

Abbildung 15: Der untere Teil einer Artikelansicht mit dem optionalen Abschnitt (in diesem Fall einer Karte) und den restlichen Tripeln

Der folgende Abschnitt ist ein optionaler Abschnitt, abhängig von der Art des Artikels. Handelt es sich bei der dargestellten Resource um einen Ort oder ein Ereignis, den/das man auf einer Karte lokalisieren kann, wird sie auf einer Google Map angezeigt. Dabei werden die Koordinaten, die

in den Tripeln mit den Prädikaten `http://www.w3.org/2003/01/geo/wgs84_pos#long` und `http://www.w3.org/2003/01/geo/wgs84_pos#lat` vorliegen, als Punkt mit einem Marker auf einer Google Map dargestellt, die auf ein mittleres Maß gezoomt ist, um sowohl größere Regionen als auch kleine Plätze vernünftig anzeigen zu können. Handelt es sich bei der Resource um eine Person wird ein kurzer Steckbrief mit Daten wie dem Geburtstag, Sterbetag, Geburtsort, Sterbeort, Geburtsnamen, alternativen Namen und Ehepartnern gezeigt. Für einige dieser Daten gibt es mehrere alternative Prädikate, unter denen sie zu finden sind. Es wird dafür gesorgt, dass immer nur eines der Prädikate verwendet wird, alle anderen werden ignoriert. Am Beispiel des Geburtsdatums sind das z.B.:

- http://dbpedia.org/property/dateOfBirth
- http://dbpedia.org/property/birth
- http://dbpedia.org/property/birthdate
- http://dbpedia.org/property/birthDate

Weiterhin gibt es einen Abschnitt, in dem Links zu korrespondierenden Wikipedia-Artikeln in anderen Sprachen aufgeführt sind. Die Sprache des angezeigten Artikels ist derzeit nur Englisch, da die Auswahl der Artikel für die englische Sprache am größten ist und es sich um die verbreitetste Sprache handelt. Die Links selber sind in der Sprache verfasst, in denen die korrespondierenden Artikel geschrieben sind.
Der letzte Abschnitt enthält eine Liste der restlichen Informationen zu der Resource, gefiltert um die vorher schon in anderer Form angezeigten Informationen sowie einige nicht relevante Daten wie die SKOS-Kategorien (mit YAGO liegt bereits ein anderes Klassensystem zugrunde) und andere Eigenschaften, die keinen Nutzwert haben. Die Liste zeigt die Informationen in Form von RDF-Tripeln, oder besser Tupeln. Angezeigt sind Paare von Prädikaten und Objekten, das Subjekt ist die Resource, um die sich der Artikel dreht. Außerdem sind auch Paare von Prädikaten und Subjekten angezeigt, wobei dann die Artikelresource das Objekt ist. Dies ist dadurch gekennzeichnet, dass nicht nur das Prädikat selbst dasteht, sondern ein Konstrukt der Form: `is <Prädikatname> of`. Die Prädikate, Eigenschaften der Resource, sind dabei selber Links, die im besten Fall eine Erklärung der Prädikate liefern, sollte der Name nicht für sich selbst sprechen. Die Objekte, Ausprägungen dieser Eigenschaften, können Literale oder URIs, also andere Resourcen, sein. Handelt es sich um Literale wie Zahlen oder Namen sind sie einfach nur als solche ausgeschrieben, sind es URIs werden sie als Links dargestellt. Dabei verlinken URIs, die andere DBpedia Resourcen bezeichnen, zur Ansicht der Artikel dieser Resource im DBpedia Navigator, während Links zu anderen Webseiten auch zu diesen führen. Man kann erkennen, ob es sich um eine anzeigbare DBpedia Resource handelt, wenn der Link nicht mit `http://` startet, das `http://dbpedia.org/resource/` wurde aus Gründen

der Lesbarkeit abgeschnitten, Links zu anderen Webseiten werden immer als vollständige URLs angezeigt. Sollten für eine Eigenschaft mehr als drei Ausprägungen existieren, werden standardmäßig nur die ersten drei angezeigt zusammen mit einem Link `show`. Bei Klick auf eben diesen, werden alle Ausprägungen angezeigt mit der Option sie auch wieder zu verstecken.

Typische Anwendungsfälle Die Artikelansicht ist die zentrale Ansicht der Anwendung. Sie gibt einem Nutzer eine Übersicht über eine Resource, sowie Links zu verknüpften Resourcen. So ist es einerseits möglich, sich umfassend über eine Resource zu informieren, sei es über die Artikelansicht oder den Wikipedia-Artikel oder auch eine Fotokollektion. Andererseits kann zu verknüpften Resourcen über gleiche bzw. ähnliche Klassenzugehörigkeit oder auch über die Ausprägungen verschiedener Eigenschaften navigiert werden. Mit klugem Navigieren können sogar Fragen geklärt werden wie folgende: `Welche Preise haben Personen gewonnen, die an der selben Universität wie Albert Einstein studiert haben?`.

4.6 Die Suchresultatsansicht

Abbildung 16: Die Suchresultatsansicht mit Tag-Cloud und der Liste der Suchresultate

Beschreibung Die Suchresultatsansicht wird immer dann gezeigt, wenn nach Artikeln gesucht wird. Dies kann geschehen, indem im Artikelsuchfeld (siehe Kapitel 4.3) ein Suchbegriff eingegeben wird, der mit den Labels der Artikel verglichen wird, oder über die Zugehörigkeit zu einer YAGO-Klasse, wenn man in der Artikelansicht nach Instanzen einer Klasse sucht (siehe Kapitel 4.5) oder wenn man nach Instanzen von Konzeptbeschreibungen sucht, indem man auf Navigationsvorschläge klickt (siehe Kapitel 5.5). Bei der Suche mittels des Labels eines Artikels wird eine Tag-Cloud angezeigt, die alle Klassen bzw. die Label der Klassen (zur Anzeige der Label siehe Kapitel 5.4) zeigt, zu denen die angezeigten Artikel gehören. Dabei werden die Klassen in unterschiedlicher Schriftgröße dargestellt, je nachdem, wie viele Artikel zu ihnen gehören. Beim Klick auf sie werden nur solche Artikel angezeigt, die Instanzen der jeweiligen Klasse sind, es können dabei auch alle Artikel oder Artikel mit keiner Klasse ausgewählt werden. Darunter folgt eine Liste der gefundenen Artikel, geordnet nach dem Pagerank, den jeder Artikel besitzt. Diese Liste zeigt maximal 25 Ergebnisse gleichzeitig an, sind es mehr so werden mehrere Seiten gebildet, durch die man sich durchklicken kann. Dabei wird mit dem relevantestem Artikel angefangen, also dem, auf den am meisten verlinkt wird. Standardmäßig werden zehn Suchergebnisse gezeigt, auf Wunsch können aber auch mehr Ergebnisse (25,50,75,100) angezeigt werden, indem auf die entsprechende Anzahl geklickt wird.

Typische Anwendungsfälle Diese Ansicht wird aufgerufen, wann immer man nach Artikeln sucht. Man kann sowohl herausfinden, welche Relevanz Artikel haben, sowie durch die Ergebnisliste klicken. Die Suche ist immer ein guter Ausgangspunkt für ein Browsen eines bestimmten Themengebietes. Im Falle der Suche nach Instanzen einer Konzeptbeschreibung ist es auch möglich bestimmte Fragen zu klären, z.B. welche altertümlichen griechischen Philosophen durch Plato beeinflusst wurden. Eingeschränkt auf einzelne Klassen gilt dies auch für die Suche nach Instanzen von Klassen.

4.7 Die Klassenansicht

Beschreibung Die Klassenansicht zeigt den Kontext einer Klasse innerhalb der YAGO-Klassenhierarchie. Das heisst, es werden die Superklassen (Vaterklassen) als auch die Subklassen (Kinderklassen), der ausgewählten Klasse gezeigt. Diese werden aus der Datenbank gewonnen, in der die gesamte Klassenhierarchie gespeichert ist. Die Super- und Subklassen werden in Select-Boxen gezeigt, dabei wird sowohl das Label oder ein Teil der URI angezeigt, je nachdem was aussagekräftiger ist. Es ist möglich, jede diese Super- oder Subklassen ebenfalls in einer Klassenansicht anzuzeigen (indem man auf den `Class`-Button unter der entsprechenden Klasse drückt) oder nach Instanzen der angezeigten Klasse bzw. ihrer Super- und Subklassen zu suchen (indem man auf den `Instances`-Button unter der entsprechenden Klasse drückt). Dies ermöglicht ein durchdachtes und komplexes Navigieren durch die DBpedia-Datenbasis entlang der YAGO-Klassenhierarchie.

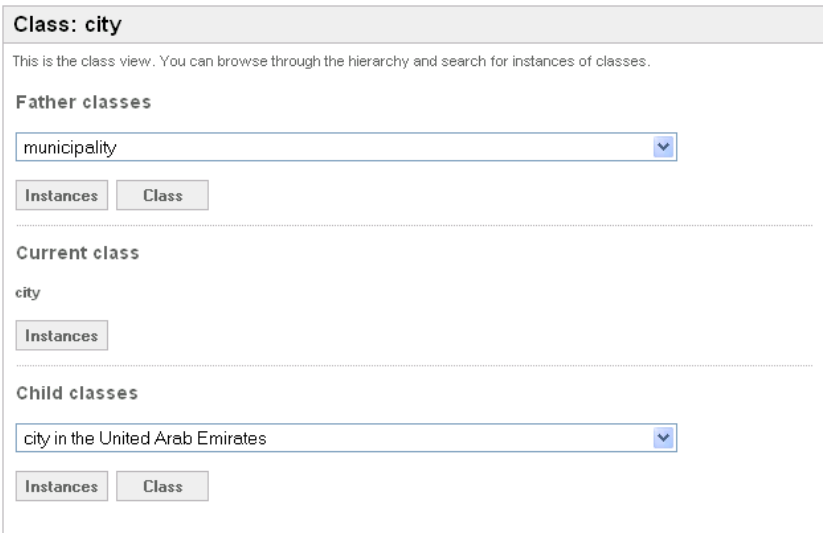

Abbildung 17: Die Klassenansicht mit der aktuellen Klasse, den Super- und Subklassen

Typische Anwendungsfälle Die Klassenansicht wird aufgerufen, wenn man etwas näheres über eine bestimmte Klasse in Erfahrung bringen oder den YAGO-Klassenbaum browsen will. Weiterhin ist die Ansicht nützlich, um zu interessanten `parallelen` Klassen zu navigieren. Mit `parallelen` Klassen meine ich solche, die eine gleiche Superklasse haben, wenn man sich also gerade einen Artikel zu einem altertümlichen griechischen Philosophen wie Plato anschaut, könnte man daran interessiert sein auch andere Philosophen wie z.B. alte Chinesische Philosophen zu betrachten. Dies wird möglich, indem man über die Klassenansicht zur Superklasse Philosoph navigiert und sich von da aus die Instanzen einer Subklasse Chinesischer Philosophen anzeigen lässt.

4.8 Die suchrelevanten und nicht relevanten Artikel

Beschreibung Auf der oberen rechten Seite befinden sich zwei Boxen mit Listen von suchrelevanten und nicht relevanten Artikeln. Neu angezeigte Artikel werden automatisch zu den suchrelevanten hinzugenommen. Durch Klicken von Plus oder Minus hinter den Artikelnamen können diese von relevant zu nicht relevant bzw. umgedreht verschoben werden, mit dem Kreuz werden sie aus den Listen entfernt, das Kreuz im Kopf der Boxen löscht die gesamte Liste. Diese Listen sind in der Session gespeichert, so dass es auch möglich ist zwischendurch zu anderen Webseiten zu wechseln, ohne dass diese Listen verlorengehen. Außerdem sind sie so für jede Funktion schnell verfügbar. Die Listen sind wichtig beim Erzeugen von Navigationsvorschlägen, da sie mit positiven und negativen Beispielen von Lernproblemen korrespondieren, deren Lösungen dann als Navigationsvorschläge interpretiert werden. Dabei sollen die resultierenden Vorschläge alle positiven Beispiele beinhalten aber

keins der negativen (Weiterführendes siehe Kapitel 5). Es werden immer nur höchstens zehn positive Artikel in die Liste aufgenommen, um ein Lernen auch bei langer Benutzungsdauer zu ermöglichen, außerdem werden so die zuletzt angeschauten Artikel gegenüber den früher angeschauten bei der Berechnung der Navigationsvorschläge bevorzugt. Die Liste arbeitet nach dem FIFO-Prinzip, wenn der elfte Artikel hinzukommt, wird der zuerst hinzugefügte gelöscht.

Typische Anwendungsfälle Die Listen der suchrelevanten und nicht relevanten Artikel haben maßgeblichen Einfluss auf das Erzeugen von Navigationsvorschlägen. Ist man also daran interessiert ein klar umrandetes Themengebiet zu durchsuchen, empfiehlt es sich, die Artikel entsprechend zu wählen. Interessiert man sich beispielsweise für altertümliche griechische Philosophen, die durch Plato beeinflusst wurden, empfiehlt es sich, alle Artikel aus der Liste zu löschen, die nichts mit dem Thema zu tun haben. Weiterhin sollte man dann einige Beispiele für solche Philosophen angeben, oder auch nur eines. Sollten die angebotenen Navigationsvorschläge zu allgemein sein (z.B. nur altertümliche griechische Philosophen), sollte man ein abgrenzendes nicht relevantes Beispiel geben, also einen altertümlichen griechischen Philosophen, der nicht durch Plato beeinflusst wurde (was möglich wäre, indem man nach Instanzen der Klasse der altertümlichen griechischen Philosophen sucht und dann solche auswählt, die nicht durch Plato beeinflusst wurden). Sollte man keine solch zielgerichtete Suche planen, kann man die Fenster auch ignorieren, da die Zuordnung zu relevanten Artikeln automatisch passiert.

Abbildung 18: Die rechte Seite der Anwendung mit den relevanten und nicht relevanten Artikeln sowie den zuletzt angeschauten Artikeln und Klassen

4.9 Die zuletzt angeschauten Artikel und Klassen

Beschreibung Unter den Boxen mit den suchrelevanten und nicht relevanten Artikeln gibt es noch zwei Boxen mit Links zu den fünf zuletzt angeschauten Artikeln sowie den fünf zuletzt angeschauten Klassen. Diese sind in der Session gespeichert, um den SPARQL-Endpoint und die Datenbank zu entlasten und ein schnelles Wiederaufrufen zu ermöglichen. Das ermöglicht außerdem den Aufruf des letzten Artikels, sollte man den Reload-Button des Browsers benutzten oder kurzzeitig auf andere Webseiten wechseln (in diesen Fällen wird immer der letzte Artikel geladen, der in der Session gespeichert wurde).

Typische Anwendungsfälle Diese Funktion ersetzt teilweise den Zurück-Button des Browsers, da dieser aufgrund der Verwendung von Ajax-Anfragen nicht funktioniert. Gerade beim Navigieren durch ein großes Themengebiet kommt es immer wieder vor, dass man an den Ausgangspunkt seiner Suche oder ein bestimmtes Zwischenergebnis zurückkehren will, dies ist so möglich.

4.10 Navigationsvorschläge

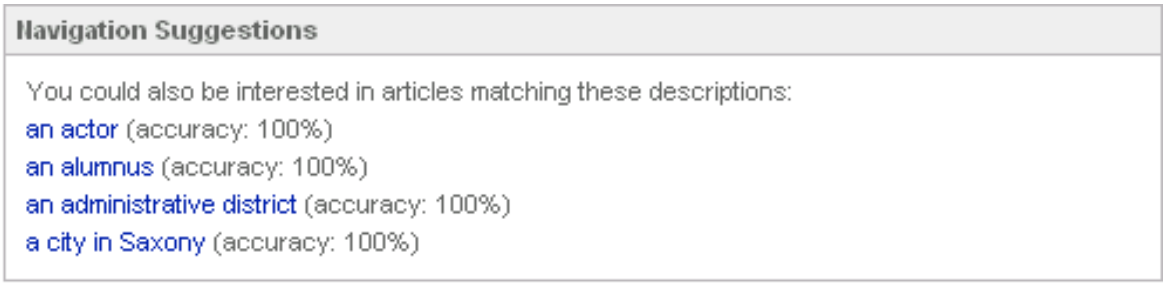

Abbildung 19: Die Box mit den Navigationsvorschlägen

Beschreibung Die Navigationsvorschläge werden über den Artikeln angezeigt. Diese Vorschläge sind Konzeptbeschreibungen, Lösungen von Lernproblemen, die möglichst alle relevanten und keine der nicht relevanten Artikel beinhalten. Diese Konzeptbeschreibungen werden allerdings nicht in beschreibungslogischer Notation angezeigt, sondern in einer natürlichsprachlichen Form (näheres siehe Kapitel 5.4). Klickt man auf die Navigationsvorschläge, wird eine Suche nach Instanzen der gelernten Konzeptbeschreibungen gestartet und die Ergebnisse in einer Suchresultatsansicht gezeigt (siehe Kapitel 5.5).

Typische Anwendungsfälle Die Navigationsvorschläge sind sowohl für zielgerichtete Suche geeignet als auch für einfaches Browsen. Der zielgerichtete Sucher kann Beispiele für eine bestimmte Klasse an Objekten angeben, um dafür eine Erklärung in Form eines Navigationsvorschlags zu erhalten, mit der er weitere Beispiele dafür erhält. Für das einfache Browsen sind die Navigationsvorschläge eine zusätzliche Möglichkeit (zusätzlich zur Suche, den Klassen und den Eigenschaften) in der DBpedia Wissensbasis zu navigieren.

4.11 Die Load-Anzeige mit Unterbrechungsfunktion

Abbildung 20: Die Server-Call Anzeige mit Abbruch-Funktion

Abbildung 21: Die Database-Call Anzeige

Beschreibung Auf der rechten oberen Seite befindet sich ein Hinweis, immer wenn die Datenbank oder der SPARQL-Endpoint angefragt werden. Es ist möglich Anfragen an den SPARQL-Endpoint abzubrechen, da diese mitunter lange dauern können, dazu wird beim Klick auf das rote Kreuz hinter der Server-Call Anzeige ein Hinweis in eine Datei geschrieben, dass die jeweilige Funktion abgebrochen werden soll. Der Inhalt der Datei wird in einer Schleife, die auf das Ergebnis des Funktionaufrufs wartet, abgefragt. Ist der Hinweis zum abbrechen gegeben, wird versucht den Thread im DL-Learner, der den SPARQL-Endpoint anfragt, kontrolliert zu beenden und die Funktion abzubrechen. Der Hinweis Server-Call verschwindet dann und man kann einfach weiterarbeiten. Die Hinweise sind in roter Signalfarbe gehalten, um den Nutzer darauf hinzuweisen, da seine Konzentration eher auf den linken oder zentralen Teil der Anwendung konzentriert ist, wenn eine solche Anfrage stattfindet.

Typische Anwendungsfälle Die Datenbank wird bei jeder Suche angefragt, der Hinweis gibt dem Nutzer das Gefühl, dass im Hintergrund etwas passiert und das Ergebnis der Anfrage bald zu erwarten ist. Der SPARQL-Endpoint wird angefragt, um den Inhalt von Artikeln abzurufen sowie für das Lernen und erstellen von Navigationsvorschlägen. Gerade das Lernen und die dazu nötige Extraktion von Wissen (siehe Kapitel 5.2) kann mitunter länger dauern, weshalb es manchmal von Vorteil sein kann, diesen Vorgang abzubrechen, besonders wenn man viele Artikel schnell hintereinander aufrufen will und nur die Navigationsvorschläge für die Gesamtheit der Artikel von Interesse ist.

4.12 Das REST Interface

Beschreibung Die Anwendung besitzt ein REST Interface, welches erlaubt, den aktuellen Zustand der Applikation auf eine URL abzubilden. Diese URL wird unter dem zentralen Bereich angezeigt und bei jeder Aktion upgedated. Dazu schreibt die jeweils letzte Aktion den entsprechenden Teil der URL in die Session. Nachdem die Aktion fertig gelaufen ist, wird per Ajax-Aufruf eine Funktion gestartet, die diese letzte Aktion ausliest, sowie die Liste der suchrelevanten und nicht relevanten Artikel und daraus eine URL kreiert. Wird diese URL im Browser eingegeben, wird die letzte Aktion wieder ausgeführt und die Listen der suchrelevanten und nicht relevanten Artikel entsprechend gefüllt. Dazu wird die URL intern durch eine `.htaccess` Datei umgeschrieben und auf die `index.php` geleitet, wobei die einzelnen Teile der URL zu *Querystrings* überführt werden. So wird beispielsweise:

```
http://navigator.dbpedia.org/search/Leipzig?
positives=[http://dbpedia.org/resource/Leipzig]
```

zu

```
http://navigator.dbpedia.org/index.php?search=Leipzig&
positives=[http://dbpedia.org/resource/Leipzig]
```

Das sind dann nichts anderes als GET-Variablen, die in der `index.php` mittels entsprechender Methoden ausgelesen werden können. Das Interface hat die folgende Syntax:

```
URL        ::= 'http://' host '/' function '/'
label ['?' parameters]
host       ::= (der Host, auf dem der Navigator installiert ist)
function   ::= 'showArticle' | 'search' | 'showClass' |
'searchInstances' | 'searchConceptInstances'
label      ::= (das Label eines Artikels oder einer Klasse)
parameters ::= parameter ['&' parameter]
parameter  ::= ('positives=' article+) | ('negatives=' article+) |
('concept=' concept)
article    ::= '[' uri ']'
uri        ::= (die URI des Artikels)
concept    ::= (eine Konzeptbeschreibung)
```

Typische Anwendungsfälle Die Benutzung des Interface ermöglicht es, Informationen mit entfernten Freunden zu teilen, einfach indem man ihm die URL gibt. Weiterhin ist es möglich bestimmte

interessante Zustände der Anwendung aufzurufen, indem man einem Link folgt. Das ermöglicht es, Demonstrationen der Anwendung auf einer Webseite zu präsentieren, auf der man einer Liste von Links folgen kann, die jede Funktion präsentieren. Erfahrene Nutzer können außerdem die Nutzung der grafischen Eingabeelemente umgehen, und direkt über eine intuitive URL Funktionen aufrufen.

http://localhost/dbpedia-navigator/search/Leipzig?positives=[http://dbpedia.org/resource/Leipzig]

Abbildung 22: Eine automatisch erzeugte URL, nachdem nach Leipzig gesucht wurde

5 Generierung der Navigationsvorschläge

Die Navigationsvorschläge werden mit Hilfe der in Kapitel 2.8 vorgestellten maschinellen Lernalgorithmen erstellt. Die Generierung der Navigationsvorschläge läuft in verschiedenen Stufen ab. In einem ersten Schritt werden die Beispiele zu Gruppen angeordnet, dann wird ein Fragment der Wissensbasis ausgewählt, basierend auf den positiven und negativen Beispielen bzw. den relevanten und nicht relevanten Artikeln. Nun wird der Lernprozess für dieses Fragment und die gegebenen Beispiele gestartet, welcher dann in beschreibungslogischen Konzeptbeschreibungen mündet. In einem letzten Schritt werden für diese Beschreibungen natürlichsprachliche Ausgaben generiert.

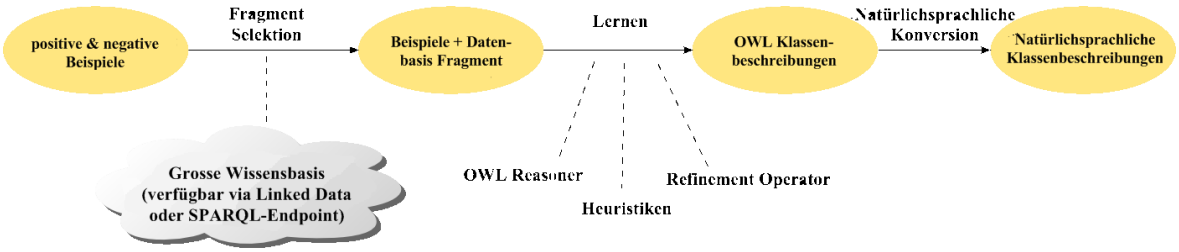

Abbildung 23: Visualisierung des Generierungsprozesses

5.1 Bildung der Lernprobleme

Navigationsvorschläge werden jedes mal dann berechnet, wenn sich die Listen der relevanten und nicht relevanten Artikel ändern. Ist dies der Fall, werden die Artikel bzw. die Objekte, die dadurch beschrieben werden, zu Gruppen mit ähnlichem Themengebiet geordnet. Dies wird getan, um zu verhindern, dass völlig unterschiedliche Objekte wie eine Stadt und ein Tier zu einem Lernproblem zusammengefasst werden. Wäre dies der Fall, würden nur sehr generelle Navigationsvorschläge mit wenig Aussagekraft entstehen, da die Konzeptbeschreibung alle unterschiedlichen Beispiele abdecken müsste. Im genannten Beispiel würde das also in eine Konzeptbeschreibung resultieren, die irgendeine physische Entität darstellen würde. Da das sicher auf einen Großteil der Objekte zutrifft, wäre dieser Navigationsvorschlag wohl relativ nutzlos.
Um nun die Beispiele in Gruppen zu ordnen, wird als erstes festgestellt, welche YAGO-Klassen ihnen zugeordnet sind. Als nächstes werden alle Superklassen dieser Klassen gesucht und eventuell wieder deren Superklassen und jedem Beispiel so eine Menge an Klassen K zugeordnet. Für die Menge und Feinkörnigkeit der entstehenden Gruppen ist entscheidend, wie oft diese Superklassen gebildet werden. Ich habe mich dafür entschieden nur einmal diese Superklassen zu bilden, da Tests ergeben haben, dass die Gruppen sonst zu unspezifisch werden, dies kann sich bei Veränderungen des YAGO-

Klassensystems natürlich ändern. Zwei Beispiele $A\ und\ B$ gehören nun zu einer Gruppe G_{AB} wenn $K_A \cap K_B \neq \emptyset$ mit $K_{G_{AB}} = K_A \cup K_B$. Diese Gruppe kann dann wieder mit anderen Beispielen oder Gruppen verglichen werden. Dies geht so lange, bis nur noch Gruppen existieren deren Klassenmengen disjunkt sind.
Nun da die Beispiele in Gruppen angeordnet sind, wird für jede Gruppe ein eigenes Lernproblem erstellt und der Lernalgorithmus darauf angewandt. Es kann natürlich vorkommen, dass eine Gruppe nicht sowohl positive als auch negative Beispiele besitzt. In diesem Fall wird das jeweils fehlende generiert. Dazu werden entweder parallele Instanzen gesucht, das heißt Instanzen der selben Klassen, oder mit dem Beispiel über eine Eigenschaft verbundene Instanzen oder Instanzen von Superklassen. Die Generierung nah verbundener Beispiele garantiert, dass die Beispiele einer Gruppe wirklich zu einem Themengebiet gehören. Die resultierenden Lernprobleme bestehen also immer aus mindestens einem positiven und einem negativen Beispiel.
Da immer nur eine begrenzte Anzahl von Navigationsvorschlägen angezeigt werden soll, wird von jeder Gruppe nur eine bestimmte Anzahl an Lösungen angefordert, gibt es nur eine Gruppe wird versucht 3 Vorschläge zu generieren, bei 2 Gruppen jeweils 2 und bei 3 oder mehr Gruppen nur jeweils einer.

5.2 Skalierung des Lernalgorithmus für sehr große Wissensbasen

Da der Lernalgorithmus von Reasonern abhängig ist und OWL-Reasoner normalerweise nicht in der Lage sind sehr große Wissensbasen wie DBpedia zu bearbeiten, wird zuerst eine Wissensfragmentauswahl durchgeführt. Diese Extraktion wird durchgeführt, indem SPARQL-Anfragen ausgeführt werden, die mit den Beispielinstanzen zusammenhängendes Wissen abrufen. Das Fragment ist wesentlich kleiner als die ursprüngliche Wissensbasis und macht es dem Reasoner möglich, seine Arbeit zu tun. Der Extraktionsprozess selbst wird in [8] beschrieben und wird hier nicht im Detail besprochen. Allgemein gesagt, startet die Extraktionsprozedur mit den Beispielinstanzen (suchrelevanten und nicht relevanten Artikeln in diesem Fall) und findet damit verbundene Instanzen bis zu einer bestimmten Rekursionstiefe. Dann werden wichtige Teile der Schemainformationen extrahiert (siehe Abbildung 24). Die Extraktion befasst sich auch mit OWL DL Konformitätsfragen, z.B. korrektes Ausschreiben der Resourcen, um sicherzustellen, dass der Reasoner das Fragment richtig verarbeiten kann.

5.3 Anpassung des Maschinellen Lernalgorithmus

Die Qualität der resultierenden Navigationsvorschläge hängt stark davon ab, welche Wissensfragmente extrahiert werden und mit welchen Einstellungen der Lernalgorithmus gestartet wird. Deshalb

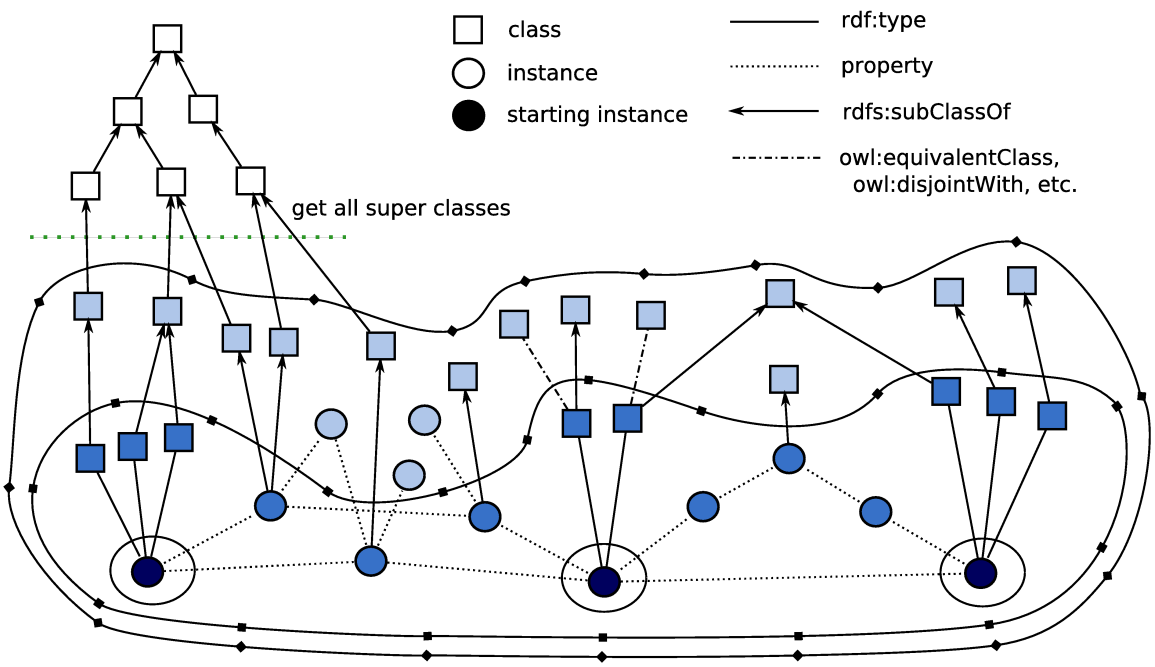

Abbildung 24: Extraktion mit drei Startinstanzen, die Kreise repräsentieren verschiedene Rekursionstiefen

werden zuerst einige Filter angewandt, um das extrahierte Wissen zusätzlich zu den oben beschriebenen Mechanismen weiter zu beschränken. Um die Beispiele zu klassifizieren und damit die Generierung von Konzeptbeschreibungen möglich zu machen, wird das YAGO-Klassensystem genutzt. Andere Klassensysteme wie SKOS oder Umbel [24] werden deshalb ausgefiltert, da erstens damit erzeugte Navigationsvorschläge nicht gut weiterverarbeitet werden können (siehe Kapitel 5.5) und die Anwendung verschiedener Systeme den Nutzer außerdem verwirren könnte.
Weitere Anpassungen werden über Optionen der Wissenskomponente sowie des Lernalgorithmus vorgenommen. Die Rekursionstiefe des Extraktionsprozesses wird auf 1 festgelegt, das bedeutet, das genügend Information extrahiert wird, um ein nichttriviales Ergebnis des Lernprozesses zu erhalten aber nicht zu viel Information, welche die Laufzeit sowohl des Extraktionsprozesses selbst als auch des Lernprozesses negativ beeinflussen würde. Da der Lernprozess und vor allem die Extraktion zeitaufwendige Operationen darstellen, ist es für ein positives Erleben einer schnell reagierenden Anwendung unabdingbar, dass die Komplexität des Prozesses eingeschränkt wird. Das Nutzen eines Caches für die extrahierten Fragmente ist dabei ein weiterer wichtiger Punkt sowie die Einstellung einer maximalen Laufzeit des Lernprozesses von drei Sekunden.
Um relevante und nichttriviale Navigationsvorschläge zu erzeugen, sind einige weitere Optionen

[24] http://www.umbel.org

wichtig. Eine Option sorgt dafür, dass der Refinement Operator (siehe Kapitel 2.7) angewendet wird, bis alle Konzepte mit gleicher Länge gefunden wurden. Das verhindert, dass sehr generelle Konzepte nicht expandiert werden, weil sie nicht aussichtsreich erscheinen. Außerdem wird sichergestellt, dass der Algorithmus solange läuft, bis mindestens drei gute Konzeptbeschreibungen (Beschreibungen, die alle positiven aber keine negativen Beispiele abdecken) gefunden wurden. Weiterhin werden Konzeptbeschreibungen ausgeschlossen, die Negation oder den All-Konstruktor verwenden. In einer so großen Wissensbasis wie DBpedia bedeutet die Negation eines Konzeptes meist ein Konzept, welches viel zu allgemein ist, um informationsbringend zu sein. Z.B. wären alle Nicht-Philosophen sowohl andere Personen, die nicht Philosophen sind, aber genauso Orte, Ereignisse, andere Lebewesen, Ideen, usw., solche Konzepte schränken zu wenig ein. Der All-Konstruktor kann wiederum nicht vernünftig angewandt werden, da große Wissensbasen wie DBpedia, die automatisch generiert werden, natürlicherweise unvollständig sind. Somit ist es annähernd unmöglich Instanzen für solche Konzepte zu finden bzw. wenn man sie findet, ist keineswegs sichergestellt, dass sie auch wirklich dem Konzept entsprechen, da nicht notwendigerweise alle Rollenbeziehungen, auf die sich der All-Konstruktor bezieht, vollständig vorhanden sind. Dies trifft natürlich auch schon für die positiven und negativen Beispiele zu, weshalb es sogar sein kann, dass das generierte Konzept mit dem All-Konstruktor selbst schon unzutreffend ist, da es vielleicht für die unvollständigen Daten zutrifft, keineswegs aber für die reellen Entitäten, die dadurch beschrieben werden. Ein Beispiel wäre ein Konzept `Schriftsteller` $\sqcap \forall$ `istInteressiertAn.Person`. Dies könnte man erhalten, wenn man als Beispiel einen Schriftsteller hat, der sich für Napoleon und Hannibal interessiert. Es kann aber sein, dass die Daten unvollständig sind und er sich auch für Physik und Flora und Fauna interessiert hat, dann wäre das Konzept nicht korrekt.

5.4 Natürlichsprachliche Konversion der Konzeptbeschreibungen

Der Lernalgorithmus liefert nachdem er fertig gelaufen ist eine festgelegte Menge an Konzeptbeschreibungen in beschreibungslogischer Syntax. Da diese Syntax für Endbenutzer kaum lesbar ist, findet eine Konversion in natürliche Sprache statt, die Beschreibungen werden zu kurzen Wortgruppen umgewandelt. Dazu werden die Konzepte und Rollen welche mit beschreibungslogischen Konstrukten verbunden sind und die als URIs repräsentiert sind, durch ihr Labels ersetzt. Es gibt allerdings Fälle, in denen die URI aussagekräftiger als das Label ist, z.B. URI: `http://dbpedia.org/class/yago/CoastalCities` Label: `city`. In diesem Fall wird nicht das Label genommen, sondern der hintere Teil der URI, also in dem Fall `CoastalCities`, wobei dieser Teil noch weiter bearbeitet wird. Ein weiterer Grund für die Ersetzung durch einen Teil der URI ist, dass sonst mehrere scheinbar gleiche Konzeptbeschreibungen auftauchen, die sich aber in Wirklichkeit unterscheiden, da die Label zwar gleich sind, die URIs allerdings nicht. Allgemein gesagt, wird ein Teil der URI genommen, falls diese Zeichenkette um mindestens 3

Zeichen länger ist als das Label und somit anscheinend mehr Information birgt. Dies ist vor allem der Fall bei den Blättern des YAGO-Klassenbaums, also den spezifischsten Klassen, was wohl daran liegt, dass diese Klassen erst kürzlich eingefügt wurden und noch keine Zeit war auch alle Label richtig anzupassen. Es gibt allerdings auch Fälle, in denen die URI folgendermaßen lautet: `Philosopher124597201`, diese URI ist auch länger als das Label `philosopher`. Da nahezu alle Klassen in den inneren Knoten des YAGO-Klassenbaums diese YAGO-spezifische Numerierung tragen und die Label für diese Klassen meist gut gewählt sind, wird in dem Falle einer Zahl am Ende der URI immer das Label genommen.
Außerdem werden die beschreibungslogischen Konstrukte wie Konjunktion, Disjunktion, Negation oder verschiedene Restriktionen durch einfache sprachliche Konstrukte ersetzt. Die folgende Tabelle zeigt die Ersetzungen.

$A \sqcap B$	A and B
$A \sqcup B$	A or B
$\neg A$	not A
$\exists r.A$	has r which is A
$\forall r.A$	all r are A
A	a(n) A
$<= 3r.A$	at least 3 r which is A
$>= 3r.A$	at least 3 r which is A
$<= 3r.a$	at least 3 r which is A

Tabelle 1: Ersetzungen beschreibungslogischer Konstrukte

Ein weiterer besonderer Fall sind Konzeptbeschreibungen, in denen $\top$, $\bot$ oder $http : //dbpedia.org/class/yago/Entity100001740$ (wobei dies die allgemeinste YAGO-Klasse ist, von der alle anderen Klassen erben, wodurch sie fast äquivalent mit $\top$ ist, nicht ganz allerdings, da es Objekte ohne jegliche Klassenzuordnung gibt) vorkommt. Diese Konstrukte werden nicht in natürliche Sprache übersetzt und die resultierenden Beschreibungen angepasst, z.B. wird $A \sqcap http : //dbpedia.org/class/yago/Entity100001740$ einfach zu `a(n) A` oder $\exists r.\top$ zu `has r`.

5.5 Verarbeitung der Navigationsvorschläge

Nachdem die Navigationsvorschläge generiert, in natürlichsprachliche Form umgewandelt und angezeigt wurden, hat der Nutzer die Möglichkeit, sich Instanzen der Konzeptbeschreibungen, aus denen

die Navigationsvorschläge gebildet werden, anzeigen zu lassen. Diese Instanzen werden auf zwei verschiedenen Wegen gewonnen. Für Konzepte, die nur die beschreibungslogischen Kontrukte $\sqcap$ und $\sqcup$ enthalten, ist es möglich eine SQL-Anfrage zu konstruieren, die aus der Datenbank die Instanzen gewinnt. Das hat zwei Vorteile, erstens ist der Datenbankzugriff äußerst schnell und zweitens ist der Pagerank in der Datenbank gespeichert und es ist somit möglich die Instanzen nach Relevanz anzuordnen. Diese Anfrage überprüft alle Instanzen der Klassen, die in der Konzeptbeschreibung vorkommen, verbunden durch `AND` und `OR` (anstatt $\sqcap$ und $\sqcup$), für die Konzeptbeschreibung $A \sqcap (\sqcup) B$ also alle Instanzen $c\ so\ dass\ c\ rdfs:subClassOf\ A\ AND\ (OR)\ c\ rdfs:subClassOf\ B$. Außerdem werden auch noch alle Instanzen der Subklassen dieser Klassen geprüft. Für eine hundertprozentige Genauigkeit müsste man wirklich alle Subklasen in beliebiger Tiefe überprüfen, da sich der Baum jedoch sehr weit aufspaltet würde dies zu einer riesigen Anfrage führen, die noch dazu eine zu lange Laufzeit hätte. Die oben beschriebenen Anpassungen des Algorithmus führen allerdings dazu, dass die resultierenden Navigationsvorschläge recht spezifisch sind, das heißt die Klassen sind meist Blätter des Klassenbaums oder Superklassen von Blättern, dadurch ist die Rekursionstiefe 1 ausreichend. Enthält die Konzeptbeschreibung auch Rollen ist es nicht mehr möglich eine SQL-Anfrage zu erstellen, da in der Datenbanken keine Rollen, das heißt Eigenschaften der Objekte, gespeichert sind. Ist dies der Fall wird stattdessen eine SPARQL-Anfrage konstruiert, welche Instanzen der Konzeptbeschreibung liefert. Dazu wird das Konzept rekursiv über seinen Aufbau durchlaufen und beschreibungslogische Konstrukte durch entsprechende Konstrukte von SPARQL ersetzt, beispielsweise wird:

`∃students.TheoreticalPhysicist`

zu

```
SELECT ?subject
WHERE {
  ?subject <students> ?object1.
  object1 a <TheoreticalPhysicist>
}
```

Die gewonnenen Instanzen werden dann mit Hilfe des in der Datenbank gespeicherten Pageranks geordnet und angezeigt.

6 Evaluation

6.1 zeitliche Evaluation

Die DBpedia Wissensbasis und die damit verlinkten Datenbestände sind auf eine enorme Größe angewachsen. Der letzte Release bestand aus über 100 Millionen Tripeln. Die Generierung der Navigationsvorschläge extrahiert daraus zwar nur ein Fragment, welches jedoch auch umso größer ist, je komplexer die Wissensbasis ist. Hinzu kommt, dass Datenbanken bei immer größeren Datenbeständen länger benötigen, einzelne Teile daraus zu finden. Zusätzlich dazu, benötigt der Lernalgorithmus für komplexere Wissensfragmente auch mehr Zeit, um die Lösungen eines Lernproblems zu bestimmen.
Alle diese Faktoren führen dazu, dass Zeit ein wichtiger Faktor für die Erstellung der Navigationsvorschläge und damit für den DBpedia Navigator als ganzes ist. Nutzer mögen es nicht, zu warten. Langsame Antwortzeiten sind eine der meistgenannten Gründe für Beschwerden von Internetnutzern. Nachdem eine gewisse `Wartezeit-Schwelle` überschritten wird, fangen Nutzer an, nach einer schnelleren Webseite zu suchen. Wo genau diese Schwelle liegt hängt von verschiedenen Faktoren ab. Für eine sich nicht inkrementell aufbauende Seite liegt die Zeit bei unter 8,6 Sekunden, bei sich inkrementell aufbauenden bei 20-30 Sekunden mit nützlichem Inhalt innerhalb von 2 Sekunden[9]. Für den DBpedia Navigator dürfte die Zeit eher bei den inkrementell aufbauenden Webseiten zu suchen sein, da ja schon Inhalte wie der Artikel vorliegen und die Navigationsvorschläge nur zusätzlich geladen werden. Im folgenden werde ich die Zeit zur Generierung der Navigationsvorschläge für verschiedene Konfigurationen evaluieren und damit aufzeigen, welche Konfiguration für die Usability des DBpedia Navigators optimal ist, sowie, ob die Erzeugung der Navigationsvorschläge schnell genug erfolgt.

Getestet wurden immer jeweils zehn Artikel, da nicht mehr als zehn gleichzeitig in der Liste der suchrelevanten Artikel stehen können. Diese zehn Artikel wurden nacheinander aufgerufen und es wurde jeweils die Zeit gemessen, die für die Erzeugung der Navigationsvorschläge gebraucht wurde. Dies bedeutet, dass z.B. zuerst `Leipzig` als Artikel aufgerufen wurde und die Navigationsvorschläge für `Leipzig` als einziges positives Beispiel berechnet wurden. Als nächstes wurde beispielsweise `Paris` aufgerufen und anschließend die Vorschläge für `Leipzig` und `Paris`, als die zwei positiven Beispiele, berechnet. So ging es weiter bis zu zehn Beispielen. Die vorliegenden Zeiten wurde über Beispiele aus verschiedenen Themengebieten und mehrere Durchläufe gemittelt. Die erste Abbildung zeigt einen Vergleich der Dauer für die Erstellung der Navigationsvorschläge unter Nutzung eines lokalen DBpedia-Endpoints auf dem gleichen Server, auf dem auch der DBpedia Navigator installiert ist, und unter Nutzung eines entfernten DBpedia-Endpoints. Für beide Varianten wurde ein Cache verwendet.

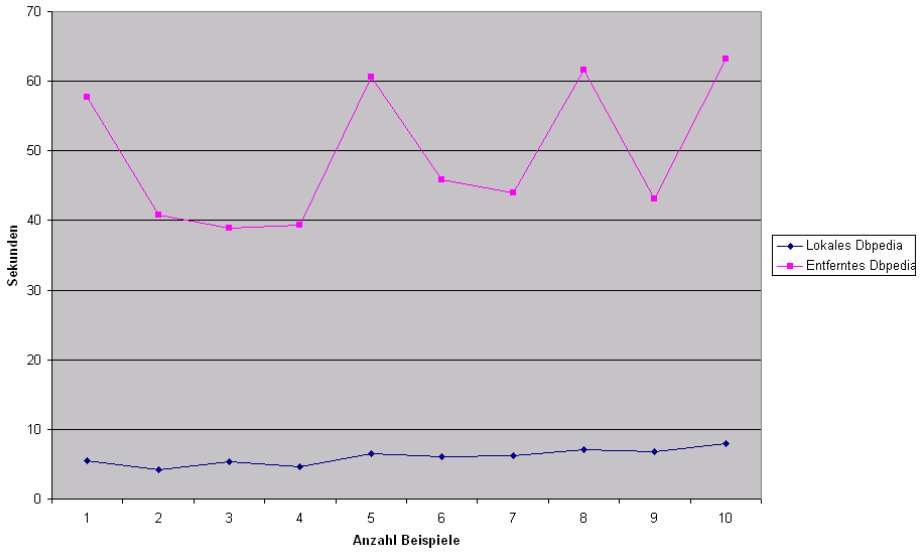

Abbildung 25: Erstellung der Navigationsvorschläge über lokales und entferntes DBpedia

Wie deutlich zu sehen ist, benötigt die Erstellung der Navigationsvorschläge für einen SPARQL-Endpoint auf einem entfernten Server wesentlich länger. Die Zeit bewegt sich im wesentlichen zwischen 40 und 60 Sekunden, während sie für einen lokalen DBpedia-Endpoint zwischen vier und acht Sekunden liegt. Des weiteren steigen beide Kurven leicht an mit einer steigenden Anzahl an Beispielen, aber sie schwanken auch recht stark.
Die Berechnung zur Gruppenbildung der Beispiele im Vorfeld der Datenextraktion, sowie des Lernalgorithmus erfolgen in jedem Fall auf dem lokalen Rechner. Aus diesem Grund ergibt sich der Unterschied bei der Erzeugung der Navigationsvorschläge bei der Extraktion von Datenfragmenten, die für das Lernen gebraucht werden. Sie ergeben sich dadurch, dass die Daten von einem entfernten DBpedia-Endpoint über das Internet abgerufen werden müssen, sowie in einem geringeren Maße dadurch, dass der entfernte DBpedia-Endpoint von einer Vielfalt von Nutzern genutzt, während der lokale Endpoint fast nur für den DBpedia Navigator eingesetzt wird. Die starken Schwankungen gerade beim entfernten Endpoint erklären sich durch Schwankungen der Übertragungsgeschwindigkeit sowie unterschiedliche Auslastung des DBpedia-Endpoints. Der Anstieg der Kurven erklärt sich durch die größere Anzahl an Beispielen, das entstehende Datenfragment ist größer und wird dadurch vom Reasoner und dem Lernalgorithmus länger bearbeitet. Da ein Cache genutzt wird, entfällt allerdings das Nachladen aller Beispieldaten bei jedem neuen Artikel, weshalb der Anstieg nur sehr leicht ist.
Aus dem Diagramm ist deutlich zu sehen, dass ein lokaler Endpoint einem entfernten vorzuziehen ist. Bei der vorhandenen Übertragungsgeschwindigkeit und den vorhandenen Rechenkapazitäten kann man sogar so weit gehen, zu sagen, dass eine Nutzung des entfernten Endpoints einem Endnutzer

nicht zuzumuten ist, da die Berechnungsdauer der Navigationsvorschläge in diesem Fall bei durchschnittlich knapp 50 Sekunden und damit weit über dem Wert liegt, der für benutzerfreundliche Anwendungen egal welchen Typs anzustreben ist. Der Wert für lokale Endpunkte liegt von der kleinstmöglichen Anzahl an Beispielen bis zur größtmöglichen unter acht Sekunden. Damit ergibt sich eine nutzerfreundliche Berechnungszeit der Navigationsvorschläge, besonders da während der Berechnung andere nützliche Inhalte wie der Artikel zur Verfügung stehen.
Ein weiteres Kriterium bei der Berechnung der Navigationsvorschläge ist die Nutzung eines Caches. Hierbei werden alle bei der Extraktion vom DBpedia-Endpoint abgerufenen Daten lokal gespeichert. Dabei entsteht für jede SPARQL-Anfrage eine Datei, mit dem Ergebnis der Anfrage in JSON-Syntax (für eine speichereffiziente Speicherung) als Dateiinhalt sowie einem Hash der Anfrage als Dateiname. Dadurch müssen für die Berechnung der Navigationsvorschläge für mehrere Beispiele immer nur die Daten des neuesten Beispiels abgerufen werden, da die für die alten bereits im Cache vorliegen. Die nächste Abbildung zeigt die Berechnungsdauer für die Navigationsvorschläge mit und ohne Cache von einem lokalen Endpoint. Ansonsten ist die Versuchsdurchführung identisch mit der oberen.

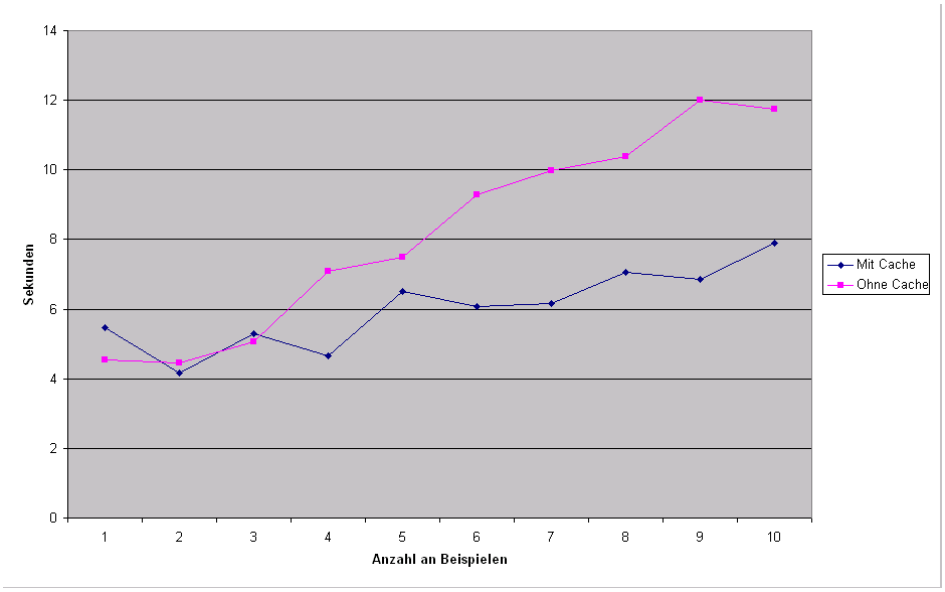

Abbildung 26: Erstellung von Navigationsvorschlägen mit und ohne Cache

Man sieht, dass beide Kurven ansteigen, dabei ist der Anstieg der Berechnungsdauer ohne Cache allerdings deutlich stärker als der mit Cache. Mit Cache bewegt sich die Dauer zwischen vier und acht Sekunden während sie ohne Cache zwischen 4,5 und 12 Sekunden liegt, es ergibt sich also ein fast doppelt so großer Anstieg. Die leichten Schwankungen in den Kurven wurden bereits oben erklärt. Der größere Anstieg der Berechnungsdauer ohne Cache ergibt sich dadurch, dass für jede

Berechnung die Daten für alle Beispiele vom DBpedia-Endpoint abgerufen werden müssen, während im Falle eines Caches immer nur die für das zuletzt gewählte Beispiel abgerufen werden.
Die Nutzung eines Caches rentiert sich damit, da durch die Art und Weise wie Lernbeispiele entstehen, nämlich nach und nach, oft die selben Daten gebraucht werden. Da der Cache außerdem nicht nutzerspezifisch ist, kommt der Cache natürlich auch Nutzern bei neu gewählten Beispielen zu Gute, da dieses Beispiel bereits im Cache sein kann.

Abschließend ist festzuhalten, dass die Nutzung eines Caches sowie eines lokalen DBpedia-Endpoints zu empfehlen ist. Wird dies getan, ist es möglich Navigationsvorschläge für jede mögliche Menge an Beispielen in unter 8 Sekunden zu berechnen. Diese Zeit bewegt sich in einem Rahmen, der als benutzerfreundlich angesehen werden kann. Weitere Verbesserungen der Laufzeit könnten durch größere Rechenkapazitäten erreicht werden, durch eine Verkleinerung der für die Berechnung der Navigationsvorschläge benötigten Datenmenge (Nutzung anderer Klassensysteme und Eigenschaften) sowie durch einen effizienteren Lernalgorithmus, der speziell auf die Bedürfnisse von Navigationsvorschlägen zugeschnitten ist.

6.2 Evaluation der Qualität

Navigationsvorschläge sollen dem Nutzer bei der Navigation helfen oder sogar ein Mittel sein, Fragestellungen zu beantworten. Deshalb ist die Qualität der Navigationsvorschläge ein weiterer wichtiger Faktor, um zu entscheiden, wie weit die Anwendung dabei ist, genau diese Aufgaben zu erfüllen und was dabei verbessert werden kann. Die Qualität wird dabei daran gemessen, inwieweit die Navigationsvorschläge dabei hilfreich sind, bestimmte Fragestellungen zu beantworten, ob sie überhaupt sinnvoll sind oder inwieweit sie intuitiven Erwartungen entsprechen. Wie der Text schon andeutet, gibt es verschiedene Arten, die Navigationsvorschläge zu nutzen, im wesentlichen zwei, weshalb diese auch verschieden getestet werden.
Einerseits gibt es die gezielte Suche. Das heißt ein Fachmann auf einem Gebiet oder zu mindestens jemand, der in der Lage ist, Beispiele für ein Themengebiet zu nennen, beispielsweise antike griechische Philosophen, die durch Plato beeinflusst wurden, nennt Beispiele für dieses Gebiet sowie eventuell noch Gegenbeispiele, um es einzugrenzen. Er will damit Navigationsvorschläge erzeugen, die in eine bestimmte von ihm erdachte Richtung gehen, um entweder ein Konzept zu finden, dass diese Beispiele beinhaltet, oder um eine Liste von Instanzen dieses Konzepts zu erhalten. Um die Qualität der Navigationsvorschläge für diese Art der Suche zu beurteilen, ist es nötig, in der gleichen Art und Weise vorzugehen. Deshalb wird diese Methode anhand einiger ausgewählter Lernprobleme getestet. Diese wurden exemplarisch ausgesucht, um einige häufige Probleme bei der Erstellung von

Navigationsvorschlägen aufzuzeigen. Dazu gehe ich so vor, dass ich die Probleme in ihrer Entwicklung zeige, das heißt für ein Lernproblem mit zwei positiven und einem negativen Beispiel zeige ich die Lösungen für ein positives, dann zwei Positive, dann ein positives und ein negatives (das eine Beispiel wurde von den suchrelevanten zu den nicht relevanten verschoben) und am Ende zwei positive und ein negatives Beispiel. Anschließend an jedes Beispiel bespreche ich Abweichungen vom Erwartungswert sowie Gründe dafür sowie die Qualität der Beispiele.

Beispiel 1 - Antike griechische Mathematiker

Erster Durchlauf:

- positive Beispiele: Pythagoras
- negative Beispiele: -
- Erwartungswert: antiker griechischer Mathematiker und Philosoph
- Navigationsvorschläge: an abstainer, an AncientGreekMathematicians, an AncientGreekPhilosophers

Zweiter Durchlauf:

- positive Beispiele: Pythagoras, Zeno of Elea
- negative Beispiele: -
- Erwartungswert: antiker griechischer Philosoph
- Navigationsvorschläge: a philosopher and has date of birth, a philosopher and has date of death, an entity and a philosopher and has date of birth

Dritter Durchlauf:

- positive Beispiele: Pythagoras
- negative Beispiele: Zeno of Elea
- Erwartungswert: antiker griechischer Mathematiker und Philosoph
- Navigationsvorschläge: an abstainer, an AncientGreekMathematicians, an AncientGreekPhilosophers

Vierter Durchlauf:

- positive Beispiele: Pythagoras, Euclid
- negative Beispiele: Zeno of Elea
- Erwartungswert: antiker griechischer Mathematiker
- Navigationsvorschläge: an AncientGreekMathematicians, a geometer, a geometer

Im zweiten Schritt erscheinen Vorschläge, die zwar nützlich sind, allerdings teilweise redundante Informationen enthalten. Jeder Philosoph hat ein Geburtsdatum und jeder Philosoph ist eine Entität. Diese Lösungen treten einerseits deshalb auf, weil die Daten unvollständig sind und somit keine Geburtdaten angegeben sind, andererseits ergeben sie sich auch aus dem genutzten Lernverfahren, das den Suchraum der Konzepte in einer für die Lösung nicht idealen Weise durchläuft. Im letzten Abschnitt treten anscheinend zweimal die selben Vorschläge auf, dies ist allerdings nicht der Fall, da es sich einmal um die Klasse `Geometer110128016` und beim anderen um `Geometers` handelt. Das Problem tritt also aufgrund der Uneindeutigkeit des Yago-Klassensystems sowie der unglücklichen Benennungen der Klassen auf. Ansonsten entsprechen die Navigationsvorschläge insgesamt den Erwartungen und sind nützlich, damit ist ihre Qualität positiv zu bewerten.

Beispiel 2 - europäische Hauptstädte

Erster Durchlauf:

- positive Beispiele: Rome
- negative Beispiele: -
- Erwartungswert: Hauptstadt
- Navigationsvorschläge: an administrative district, an area, a capital

Zweiter Durchlauf:

- positive Beispiele: Rome, Paris
- negative Beispiele: -
- Erwartungswert: europäische Hauptstadt
- Navigationsvorschläge: an area, a capital, a capital in Europe

Dritter Durchlauf:

- positive Beispiele: Rome, Paris, Tokyo
- negative Beispiele: -
- Erwartungswert: Hauptstadt
- Navigationsvorschläge: an area, a capital, a center

Vierter Durchlauf:

- positive Beispiele: Rome, Paris
- negative Beispiele: Tokyo
- Erwartungswert: europäische Hauptstadt
- Navigationsvorschläge: a capital in Europe, a land site, a parcel

Dieses Beispiel zeigt so gut wie keine Abweichung vom Erwartungswert. Durch die Hinzunahme von `Paris` als Beispiel werden die Navigationsvorschläge sogar noch spezifischer von allgemein Hauptstädten auf Hauptstädte in Europa eingegrenzt, das selbe geschieht, wenn `Tokyo` (als nicht-europäische Hauptstadt) als Negativbeispiel angeführt wird. Die Navigationsvorschläge entsprechen dem Erwartungswert und sind nützlich, damit ist die Qualität als sehr gut zu bewerten.

Beispiel 3 - abstrakte Erwachsene

Erster Durchlauf:

- positive Beispiele: Mother
- negative Beispiele: -
- Erwartungswert: physiche Entität und weiblich
- Navigationsvorschläge: has disambiguates, has owl#sameAs, has depiction

Zweiter Durchlauf:

- positive Beispiele: Mother, Father

- negative Beispiele: -
- Erwartungswert: physische Entität
- Navigationsvorschläge: has disambiguates, has hasPhotoCollection, has owl#sameAs

Dritter Durchlauf:

- positive Beispiele: Mother, Father, Child
- negative Beispiele: -
- Erwartungswert: physische Entität
- Navigationsvorschläge: has disambiguatesan entity, has disambiguates, has owl#sameAs

Vierter Durchlauf:

- positive Beispiele: Mother, Father
- negative Beispiele: Child
- Erwartungswert: Erwachsene
- Navigationsvorschläge: keine anzeigbaren Vorschläge

Die Navigationsvorschläge für dieses Beispiel entsprechen im Gegensatz zum vorherigen keineswegs den Erwartungen. Es werden keine Klassen von Objekten angegeben. Dies liegt daran, dass den Artikeln `Mother`, `Father` und `Child` keine YAGO-Klassen zugeordnet sind, weshalb der Lernalgorithmus nur Eigenschaften nutzen kann, um die Navigationsvorschläge zu erstellen. Im letzten Schritt wird `Child` als negatives Beispiel angeführt, da es allerdings nahezu die selben Eigenschaften wie `Mother` und `Father` besitzt, werden nur Lösungen generiert, die entweder ausgefiltert werden oder die nicht in natürlicher Sprache angezeigt werden können. Die Vorschläge entsprechen also nicht den Erwartungen und sind auch nicht nützlich, die Qualität ist daher schlecht.

Eine weitere Art die Navigationsvorschläge zu nutzen ist weniger gezielt, sondern tritt auf, wenn einfach aus Interesse an den Artikeln durch die Wissensbasis gebrowst wird. In diesem Fall kann es natürlich keine Erwartungswerte für die Navigationsvorschläge geben, sondern man kann sie nur anhand ihrer Nützlichkeit und Sinnhaftigkeit beurteilen. Um dieses Verhalten zu simulieren, werden die 20 häufigsten im Jahre 2008 bei Wikipedia eingegebenen Suchbegriffe getestet[25]. Dabei wurden

[25] `http://wikistics.falsikon.de/2008/wikipedia/en/`

jeweils die ersten zehn und die zweiten zehn getestet (mehr als zehn suchrelevante Artikel sind nicht möglich). Dies ermöglicht eine annähernd realistische Einschätzung des Verhaltens eines typischen Endnutzers. Die Begriffe wurden nacheinander vom häufigsten zum zehnthäufigsten eingegeben. Es wurde keine zusätzliche Aktion ausgeführt, jeder Artikel wurde also den suchrelevanten zugeordnet. Die folgende Aufstellung zeigt die eingegebenen Begriffe sowie die Navigationsvorschläge für jeweils alle zehn, die Zwischenschritte habe ich hier nicht notiert, da sich dort sehr vieles überschneidet.

Die häufigsten zehn Suchbegriffe

- positive Beispiele: Wiki, Sarah Palin, YouTube, 2008 Summer Olympics, Large Hadron Collider, Wikipedia, Michael Phelps, John McCain, Sex
- negative Beispiele: -
- Navigationsvorschläge: a LivingPeople and has birth place, an act, an abstract entity, an artefact, a book, has disambiguates

Die zehn häufigsten Suchbegriffe ab Nummer 11

- positive Beispiele: The Dark Knight (film), Facebook, MySpace, Joe Biden, Naruto, Olympic Games, Heroes (TV series), Canine reproduction, Georgia (country), Bernie Mac
- negative Beispiele: -
- Navigationsvorschläge: a film, a possession, a LivingPeople and has depiction, a debut and, a SportsCompetitions, has depiction, an abstract entity

Bei den Beispielen handelt es sich oft um Personen, im Allgemeinen um Begriffe der jüngsten Zeitgeschichte. Das wirft natürlich das Problem auf, dass gerade die vorhandenen Daten für diese aktuellen Begriffe nicht ganz so aktuell sind bzw. sein können. Ansonsten handelt es sich natürlich um verschiedenste Themengebiete. Der Algorithmus zur Gruppierung der Artikel, teilt diese in verschiedene Themengebiete ein, das sind sechs für die ersten zehn Artikel und sieben für die zweiten zehn. Das erkennt man daran, dass für jedes Themengebiet ein einzelner Navigationsvorschlag existiert. Die Vorschläge sind in beiden Fällen recht ähnlich, sie sind relativ allgemein und es gibt sowohl brauchbare als auch kaum brauchbare Vorschläge.
Da alle Personen die auftauchen aus recht unterschiedlichen Sparten kommen, ergibt sich die recht allgemeine Konzeptbeschreibung `a LivingPeople and has x`. Die Artikel ohne Klassenzugehörigkeit erzeugen meist Navigationsvorschläge ohne Klassen. Bestimmte Konzeptbeschreibungen

führen bei der Konversion der Beschreibungen in natürliche Sprache zu Fehlern, wie bei `a debut and`. Andere Vorschläge wie `a film` oder `a SportsCompetition` sind dagegen durchaus brauchbar. Durch die Mischung der Navigationsvorschläge der einzelnen Themengebiete ergeben sich immer ein paar brauchbare. Es gibt so gut wie keine völlig unsinnigen Vorschläge. Deshalb würde ich die Qualität der vorliegenden Navigationsvorschläge als durchschnittlich bis gut beurteilen.

Die vorliegende Evaluation der Qualität der Navigationsvorschläge hat ergeben, dass diese insgesamt eine gute Qualtität haben. Sie ist jedoch sehr stark von der Qualität der Daten abhängig. Weitere Faktoren sind der Lernalgorithmus sowie die Konversion der Konzeptbeschreibungen zu natürlicher Sprache. Lösungsansätze sind die Verwendung eines anderen Klassensystems, eine bessere Filterung der verwendeten Daten sowie eine Anpassung des Lernalgorithmus und der Konversion in natürliche Sprache. Genaueres dazu ist in Kapitel 7 zu finden.

7 Zusammenfassung und weitere Arbeit

Ich habe ein Web-User-Interface präsentiert, mit dem man in DBpedia und damit verlinkten Datenbeständen browsen und suchen kann. Dies erfolgt auf Basis der zugrunde liegenden Semantik, im speziellen der Klassenhierarchie und den Eigenschaften. Vor allem habe ich *Navigationsvorschläge* vorgestellt, die die Semantik nutzen, um komplexe Konzeptbeschreibungen zu erstellen, die weiter bei dieser Aufgabe unterstützen. Diese Konzeptbeschreibungen konnten in einer für den Endnutzer lesbaren Art präsentiert und zur Auffindung von Instanzen dieser Beschreibungen genutzt werden. Das Zusammenspiel dieser Funktionen erlaubt es, sowohl eine gezielte Suche für eine bestimmte komplexe Fragestellung durchzuführen, als auch, ungezielt durch den Datenbestand zu browsen und interessante Zusammenhänge zu entdecken. Außerdem konnte evaluiert werden, wie schnell die Erzeugung der Navigationsvorschläge für verschiedene Konfigurationen ist und welche Qualität sie haben, in der richtigen Konfiguration werden Vorschläge mit guter Qualität in benutzerfreundlicher Geschwindigkeit erzeugt. Ich denke außerdem, dass dies das erste Mal ist, dass kontrolliertes symbolisches maschinelles Lernen routinemäßig in einer Anwendung genutzt wird, die an den Endnutzer gerichtet ist. Das Ziel eine Anwendung zu konstruieren, die Methoden des maschinellen Lernens benutzt, um in großen Semantic Web Wissensbasen wie DBpedia effizient und intelligent zu navigieren, wird mit dem DBpedia Navigator erreicht, lässt aber noch Platz für Verbesserungen, da es sich noch um einen Prototyp handelt. Eine aktuelle Version des DBpedia Navigators kann unter `http://navigator.dbpedia.org` besucht werden.
Die Verbesserungen betreffen sowohl die Navigationsvorschläge als auch den Rest der Anwendung. Wie gezeigt wurde, sind es im Wesentlichen zwei Felder, in denen die Navigationsvorschläge verbesserungswürdig wären. Einerseits ist es der Faktor Laufzeit. Gerade für eine große Menge an positiven und negativen Beispielen liegt die Laufzeit in einem Bereich, der noch Platz für Verbesserungen lässt. Der zweite Bereich ist die Qualität der Navigationsvorschläge. Sie sind teilweise zu allgemein und manchmal sogar völlig unbrauchbar. Folgende Weiterentwicklungen könnten diese Probleme lösen:

- Große Teile des Schemas, die für den Lernalgorithmus relevant sind (Klassen- und Eigenschaftshierarchie, Domain und Range von Eigenschaften, disjunkte Klassen), könnten im Speicher gehalten werden, während der Fragmentselektionsalgorithmus dahingehend modifiziert werden würde, den SPARQL-Endpoint nur für solche Resourcen, die nicht im Speicher sind, anzusprechen. Gegenwertig wird der relevante Teil des Schemas bei jeder Anfrage neu zusammengestellt, indem der SPARQL-Endpoint angefragt wird, was merkliche Verzögerungen verursacht bis der Nutzer die Navigationsvorschläge anschauen kann.
- Der Lernalgorithmus, welcher zur Zeit adäquat dafür ist, eine große Teilmenge der verfügbaren OWL Konstrukte zu lernen, kann auf eine einfachere Sprache zugeschnitten werden, speziell Erweiterungen von EL (Expression Language).

- Da betrachtete Artikel automatisch zu den suchrelevanten hinzugefügt werden, gibt es oft Lernprobleme mit nur positiven Beispielen, im Moment werden negative dazu generiert, für die Zukunft wäre es wünschenswert den Lernalgorithmus auch für Lernprobleme mit nur positiven Beispielen anzupassen.

- Im Moment werden teilweise noch Konzeptbeschreibungen angezeigt, die keine Aussagekraft haben oder in sonst einer Weise unpassend sind. Dafür könnte ein Filter entwickelt werden, so dass immer mehr Navigationsvorschläge generiert würden, als gebraucht werden, um dann die unpassenden auszufiltern bzw. nur die besten auszuwählen (dabei könnten Länge und Struktur der Konzeptbeschreibung entscheidend sein, sowie das Vorkommen solcher Konstrukte wie $\top$ oder $\bot$.

- Das grundlegende Klassensystem, das für die Anwendung genutzt wird, ist im Moment YAGO. Dies besitzt eine sehr tiefe Hierarchie, das heißt es gibt viele Ebenen, die teilweise sehr abstrakt sind. Dies kann dazu führen, dass die gelernten Konzeptbeschreibungen auch sehr abstrakt werden, was sie dann kaum nutzbar macht. Im Moment wird eine neue DBpedia Ontologie aufgebaut[26], die erstens sehr flach ist und kaum abstrakte Klassen enthält (nur die Wurzelklasse `Resource`, die die Superklasse aller anderen Klassen ist). Weiterhin werden auch Eigenschaften wie das Geburtsdatum für einzelne Klassen definiert. Die vielfältigen verschiedenen Bezeichnungen für die gleiche Eigenschaft würden somit wegfallen und da es sich um eine von Hand erstellte Ontologie handelt, kann man auch relativ sicher sein, dass alle von ihr definierten Eigenschaften sinnvoll und informationstragend sind. Die Benutzung dieser Eigenschaften würde sowohl die Artikelanzeige als auch gelernte Konzepte verbessern, die diese Eigenschaften enthalten.

- Eine weitere Verbesserung könnte bei der Konvertierung der Konzeptbeschreibungen zu natürlicher Sprache erfolgen. Der Konverter ist im Moment in der Lage, die meisten üblichen beschreibungslogischen Konstrukte zu konvertieren, einige werden jedoch noch nicht behandelt, wie z.B. Kardinalitätsrestriktionen. Das gleiche gilt für den SPARQL-Konverter, der für eine gegebene Konzeptbeschreibung eine SPARQL-Anfrage erstellt, die Instanzen dieser Beschreibungen abruft.

- Verlinkte Datensätze und Ontologien könnten für die Erzeugung von Navigationsvorschlägen in bestimmten Gebieten genutzt werden. Über die `owl:sameAs` Beziehung, kann z.B. für Orte eine Brücke zu korrespondierenden Geonames-Entitäten geschlagen werden, die wiederum bestimmte Eigenschaften haben, wie Nachbarländer oder ähnliches. Diese Beziehungen wiederum könnten in die Navigationsvorschläge einfließen.

[26] `http://blog.georgikobilarov.com/2008/10/`

Abgesehen von diesen Verbesserungen der Navigationsvorschläge gibt es auch für die restliche Anwendung weitere Arbeit:

- Wie schon vorher beschrieben, funktioniert der Zurück-Button des Browsers nicht. Es gibt inzwischen einige Frameworks, die dies lösen, darunter die `Yahoo! UI`[27] oder auch das jQuery-Plugin `jQuery History`[28], welches ich persönlich bevorzugen würde, da das jQuery-Framework[29] sehr gut dazu genutzt werden könnte, die auf einem sehr grundlegenden Level programmierten Ajax-Funktionen mittels eines weitverbreiteten und weitentwickelten Frameworks auf einer abstrakteren Ebene zu behandeln.

- Der Pagerank für die einzelnen Artikel wird im Moment auf sehr einfache Weise berechnet, indem einfach die eingehenden Links gezählt werden. Die großen Datenmengen und beschränkten Rechenkapazitäten lassen eine genauere Berechnung nicht zu. In der Zukunft könnte man den Pagerank in einer Google-artigen Weise berechnen[15]. Außerdem könnten in einem lokalen SPARQL-Endpoint Statements hinzugefügt werden, die jeder Resource ihren Pagerank zuordnet. Somit wären Suchergebnisse, die über den SPARQL-Endpoint abgefragt wurden, sortierbar und man könnte nur die relevanteste Teilmenge an Resultaten abfragen und somit die Laufzeit verbessern.

- Die Artikelanzeige kennt bereits das Konzept, Inhalte abhängig vom Typ der Resource in spezieller Art anzuzeigen (eine Google Map für Orte, ein Steckbrief für Personen). Dies könnte für nahezu alle Kategorien ausgebaut werden. Die Anzeige der restlichen Tripel wird schon jetzt so gefiltert, dass möglichst nur die aussagekräftigsten Eigenschaften angezeigt werden, dies kann weiter ausgebaut werden, die oben bereits erwähnte DBpedia eigene Ontologie könnte eine weitere Lösung dieses Problems sein.

- Bisher ist es nur möglich nach Artikeln mittels verschiedener Kriterien zu suchen. Da aber eine Klassenansicht existiert, wäre auch eine Suche nach Klassen sinnvoll. So könnte man diese auswählen, ohne erst ein Beispiel dafür aufrufen zu müssen.

- Die GUI kann wie bei jedem Prototyp weiter ständig verbessert werden, um die Zugänglichkeit und Benutzbarkeit weiter zu verbessern, dem Ausspruch von Steve Krug[30] folgend:

  > Don't make me think!

[27] `http://developer.yahoo.com/yui/`
[28] `http://www.stilbuero.de/jquery/history/`
[29] `http://jquery.com/`
[30] `http://www.sensible.com/about.html`

Literatur

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. DBpedia: A nucleus for a web of open data. In *ISWC/ASWC (2007)*, LNCS (4825), pages 722–735. Springer, 2007.

[2] Sören Auer and Jens Lehmann. What have innsbruck and leipzig in common? extracting semantics from wiki content. In *Proceedings of the ESWC (2007)*, LNCS (4519), pages 503–517. Springer, 2007.

[3] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press New York, 2003.

[4] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[5] T. Berners-Lee, J. Hendler, O. Lassila, et al. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[6] Tim Berners-Lee. Linked data, 2006. `http://www.w3.org/DesignIssues/LinkedData.html`.

[7] Christian Bizer, Richard Cyganiak, and Tom Heath. How to publish linked data on the web, 2007. `http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/`.

[8] Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of owl class descriptions on very large knowledge bases. In *7th International Semantic Web Conference (ISWC)*, 2008.

[9] Andrew B. King, editor. *Website Optimization*. O'Reilly Media, 2008.

[10] O. Lassila, R.R. Swick, et al. Resource Description Framework (RDF) Model and Syntax Specification. 1999.

[11] Jens Lehmann. Hybrid learning of ontology classes. In Petra Perner, editor, *MLDM*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer, 2007.

[12] Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *17th Int. Conf. on Inductive Logic Programming (ILP)*, 2007.

[13] Jens Lehmann and Pascal Hitzler. A refinement operator based learning algorithm for the ALC description logic. In *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2008.

[14] D.L. McGuinness, F. van Harmelen, et al. OWL Web Ontology Language Overview. *W3C Recommendation*, 10:2004–03, 2004.

[15] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, 1999.

[16] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF W3C Recommendation 15 January 2008, 2008.

[17] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF (Proposed Recommendation). W3c:wd, W3C, 2007.

[18] N. Shadbolt, T. Berners-Lee, and W. Hall. The Semantic Web Revisited. *IEEE INTELLIGENT SYSTEMS*, pages 96–101, 2006.

[19] F. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. Technical report, Research Report MPI-I-2007-5-003, Max-Planck-Institut fiir Informatik, Stuhlsatzenhausweg 85, 66123 Saarbriicken, Germany, 2007.

[20] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 697–706. ACM, 2007.

Abbildungsverzeichnis

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Sebastian Knappe

Ort Datum Unterschrift