Sprachen und Methoden des Semantic Web
bei
Prof. Dr. Fischer

Szenariobasierte Implementation eines prototypischen Software-Agents

Eduard Litau, Zeljko Carevic
FH Köln

12. März 2010

Inhaltsverzeichnis

1. Einleitung **3**

2. Projektgegenstand **3**
2.1. Ziel und Projektplan 3

3. Ein erster Ansatz **6**
3.1. Erster Prototyp auf Basis des Jena Framework 7

4. Linked Data **9**
4.1. Geonames Datenbank 10
4.2. Anpassung der Ontologie für Linked Data 11
4.3. Architektur des Prototypen 12
4.4. Umsetzung der Architektur 14
4.4.1. Triple Store 14
4.4.2. Import der Flickr Daten 14
4.4.3. Linked Data Schnittstelle 15
4.4.4. Software Agent 16
4.4.5. Frontend 16

5. Ausblick **16**

6. Fazit **18**

Literatur **18**

A. Anhang **20**
A.1. Szenarien 20
A.2. Quellcode 22
A.3. Pubby Konfigurationsdatei 22

Abbildungsverzeichnis

1. Projektplan 5
2. Vernetzung der Ontologien von DBPedia und FlickrO 6
3. Architektur DBPedia - FlickrO 7
4. Erweiterte Ontologie 11
5. Architektur des Prototypen 13
6. Frontend des Prototyps 17

1. Einleitung

Das Semantik Web entwickelt sich immer weiter von einer reinen Vision die das erste Mal von dem Erfinder des WWW Tim Berners Lee kommuniziert wurde zur Realität. Mittlerweile existiert bereits eine Vielzahl an öffentlich verfügbaren Daten die dem Standard des Semantic Web entsprechen. Beispielsweise das auf Basis der semantischen Techniken konzipierte Gegenstück zu Wikipedia, *DBPedia*. Um jedoch das volle Potential des Semantik Web auszuschöpfen muss man in der Lage sein verschiedene Ontologien miteinander zu kombinieren. Nur so kann sich ein reichhaltiges Netz an Wissensbasen spannen. Die Kombination verschiedener Ontologien war der Gegenstand der Projektarbeit. Das Ziel der Projektarbeit war es aufzuzeigen welche Schritte zur Kombination verschiedener Ontologien notwendig sind und wie eigene Strukturen integriert werden können. Zu diesem Zweck wurde auf Basis von Szenarios ein Prototyp implementiert der dem Nutzer Informationen zu einem Reiseziel liefert. Die Informationen sollten dabei sowohl aus bestehenden Ontologien wie beispielsweise DBPedia als auch aus einer selbst entwickelten Ontologie generiert werden. Die selbst entwickelte Ontologie beinhaltet Bilder zu verschiedenen Orten wobei die Bilder über die öffentlich verfügbare *Flickr API* bezogen werden.

Die vorliegenden Projektdokmentation stellt die erzielten Ergebnisse und Erkenntnisse diese Projektes dar. Im folgenden Abschnitt soll nun zunächst der Projektgegenstand näher erläutert werden. Dabei wird speziell auf die Motivation für dieses Projekt eingegangen. Die weiteren Abschnitte stellen die erzielten Ergebnisse während der Implementationsphase dar. Grundsätzlich gliedert sich das Projekt in drei Phasen, die hier erläutert werden. In der ersten Phase wurde ein auf dem *Jena* Framework basierter Ansatz gewählt. In der zweiten Phase dieses Projekts wurde ein auf dem Konzept der *Linked Data* basierter Ansatz dargestellt. Abschliessend wird in der dritten Phase eine mögliche Verbesserung der erzielten Projektergebnisse dargestellt.

Diese Dokumentation richtet sich an ein Fachpublikum aus dem Gebiet der Informatik, welches sich mit dem Semantik Web in seinen Grundzügen auskennt.

2. Projektgegenstand

2.1. Ziel und Projektplan

Im Rahmen eines Hochschulprojekts soll ein Prototyp implementiert werden, mit dem evaluiert wird, welche aktuellen Methoden und Techniken zur Kombination mehrerer unabhängiger Ontologien existieren. Die Implementierung erfolgte aus szenariobasierter Perspektive. So wurden zu Beginn des Projektes verschiedene Szenarien entwickelt die bei der Entwicklung einen Überblick über die zu realisierenden Methoden geben soll. Ein Auszug aus den entwickelten Szenarien findet sich im Anhang A. Als zentrales

Szenario hat sich eine Urlaubsplanung durchgesetzt bei der zu einem gegebenen Ort möglichst viele hilfreiche Informationen dargestellt werden sollen. Um eine möglichst reichhaltige Aufbereitung von Urlaubsinformationen zu ermöglichen werden dem Nutzer Bilder zu dem gewünschten Ort angezeigt. Das vollständige Szenario ist in Listing 2.1 dargestellt.

```
Urlaubsplanung:
Die Familie Kunz plant einen Urlaub in nächster Zeit
    durchzuführen. Hierzu möchten die Kunzes einen groben
    Eindruck von dem Zielland in Bild und Schrift bekommen. Sie
    starten die Software und geben in die Suchmaske den Namen
    eines Landes, einer Stadt, Gegend ein. Daraufhin analysiert
    der Software-Agent die Anfrage und liefert dazu aus den ihm
    bekannten Quellen verschiedene Informationen wie Texte und
    Bilder. Gleichzeitig bietet die Software weitere
    Einstellungsmöglichkeiten an, um z.B, den Zeitraum für den
    Urlaub einzuschränken oder den Umkreis der angezeigten
    Informationen zu dem gefunden Ort zu begrenzen.
```

Projektplan

Der in Abbildung 1 dargestellte Ausschnitt des Projektplans zeigt auf, welche Aufgaben von uns identifiziert wurden und welchen zeitlichen Aufwand wir den jeweiligen Aktivitäten zugewiesen haben. Der vollständige Projektplan befindet im Projektverzeichnis. Es ist zu erkennen, dass wir einen verhältnismäßig grossen Zeitraum für die Recherche gewählt haben, wobei hier der Fokus auf die Techniken und Werkzeuge des Semantik Web gelegt wurde. Dies ist auf die Motivation zu diesem Projekt, eine praktische Anwendung für das Semantik Web zu entwickeln, zurückzuführen. Unser Ziel war es Konzepte, die uns bereits in theoretischer Form in bisherigen Veranstaltungen vermittelt wurden, in einer praktischen Arbeit anzuwenden. Trotzdem sollten einer Hochschule entsprechende theoretische Aspekte nicht ausser Acht gelassen werden. Aus diesem Grund kann das Projekt als zweigeteilt betrachtet werden. Zum Einen die Recherche und zum Anderen die tatsächliche Umsetzung bestehender theoretischer Konzepte. Diese Projektdokumentation fokussiert somit auf die praktischen Ergebnisse, die bei der Implementierung erzielt wurden. Einhergehend damit werden die evaluierten Techniken und Werkzeuge betrachtet.

Als Quelle für die Bilder kommt die Community Flickr zum Einsatz, die eine öffentlich verfügbare API bereitstellt. Eine Schwierigkeit die sich bei der Verwendung der Flickr API ergibt, ist dass die Ergebnisse die Flickr liefert in diesem Kontext nur bedingt nutzbar sind. So liefert die Flickr API zwar ein auf XML basiertes Datenformat, jedoch bietet Flickr keine Möglichkeit die Informationen in Form von RDF abzufragen. Die Daten besitzen keine semantische, maschinenlesbare Annotation. Aus diesem Grund soll eine eigene Ontologie entwickelt werden, die aus den Informationen die über die

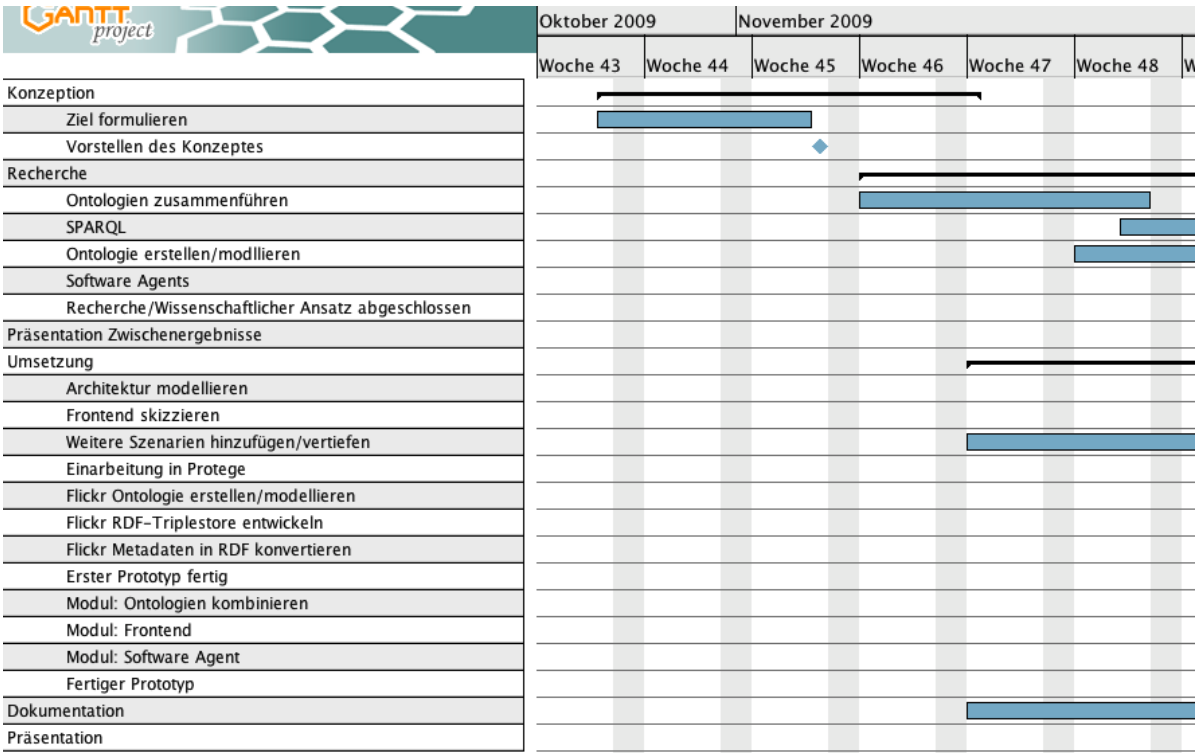

Abbildung 1: Projektplan

Flickr API bezogen werden, gebildet wird. Dabei sind die folgenden Punkte für uns von Bedeutung.

- Tags
- Titel
- Metadaten
  - Aufnahmeort
  - Autor
  - Aufnahmezeit
  - Kommentare
  - Personen auf dem Bild
  - Rechte

3. Ein erster Ansatz

In einem ersten Ansatz zur Realisierung der Kombination mehrerer Ontologien sollte versucht werden eine SPARQL Anfrage an zwei voneinander unabhängige und verteilte Ontologien zu richten. Als bereits bestehende Ontologie kommt DBPedia zum Einsatz. Die Ontologie und die entsprechenden RDF Triples sind unter http://wiki.dbpedia.org/Ontology frei verfügbar.

Als zweite Ontologie kommt die von uns entwickelte *FlickrO* zum Einsatz. Ein Auszug der FlickrO ist in Listing 3 dargestellt. Für dieses Beispiel sei angenommen, dass ein Anwender Informationen zu dem italienischen Ort Corleone haben möchte.

```
1 <owl:Thing rdf:about="#CorleonePlace">
2         <rdf:type rdf:resource="#Place"/>
3         <hasName>Corleone</hasName>
4     </owl:Thing>
5 </rdf:RDF>
```

Eine Meta Ontologie soll nun diese beiden getrennten Ontologien über OWL Klassenbeziehungen, wie sie in Abschnitt 3.1 näher erläutert werden, verbinden. Die Abbildung 2 zeigt Ausschnitte aus der DBPedia und der FlickO Ontologie sowie die Meta Ontologie.

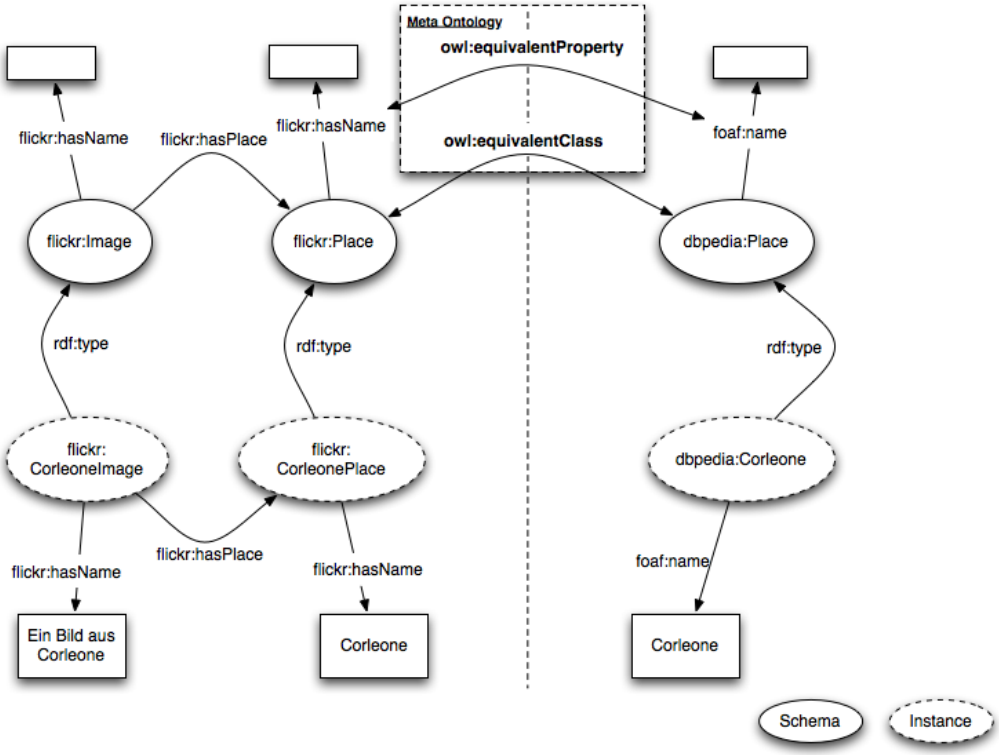

Abbildung 2: Vernetzung der Ontologien von DBPedia und FlickrO

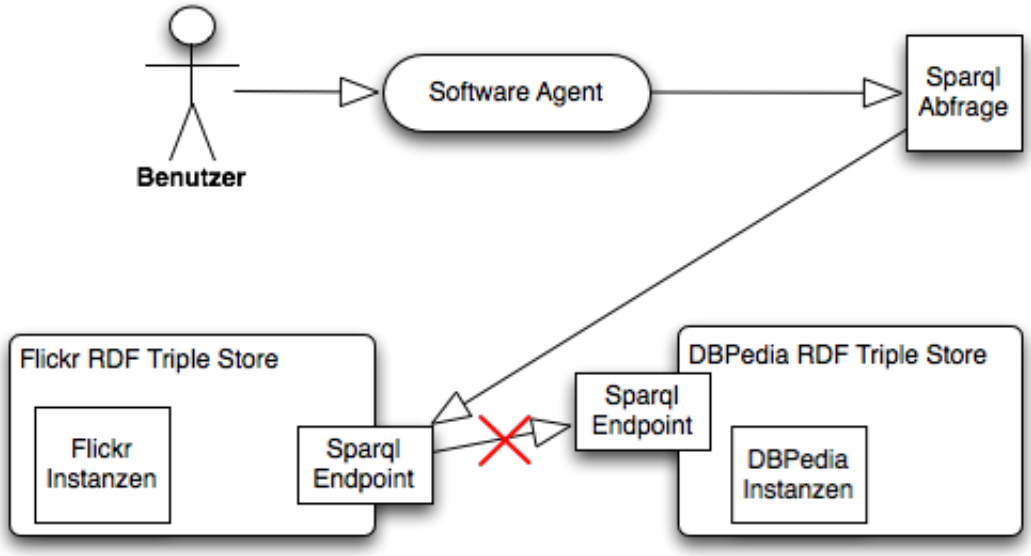

Abbildung 3: Architektur DBPedia - FlickrO

In Listing 3 ist ein RDF Triple zu dem Ort Corleone dargestellt. In dem dargestellten RDF Triple wird der Ressource *Corleone Place* mit dem Prädikat *hasName* der Name Corleone zugewiesen. Innerhalb von DBPedia ist Corleone als Ressource unter http://dbpedia.org/ressource/Corleone verfügbar. In einem ersten Schritt soll nun versucht werden die Ressource Corleone Place aus der FlickrO mit der Ressource Corleone von DBPedia zu kombinieren. Um dies zu erreichen soll versucht werden, eine Anfrage an DBPedia zu richten, und nach einer Ressource vom Typ *Place* mit dem Namen Corleone zu suchen. Dies geschieht in Form von Sparql Anfragen die an den Sparql Endpoint von DBPedia gerichtet werden. Die Architektur ist in Abbildung 3 dargestellt.

Es ist zu erkennen das sowohl die Anwendung, die die Anfrage nach dem zu suchenden Ort als auch die FlickO sich auf dem selben System befinden. Die Anfrage zum DBPedia Sparql Endpoint ist entsprechend verteilt.

3.1. Erster Prototyp auf Basis des Jena Framework

Jena ist ein auf Java basierendes Framework mit dessen Hilfe sich Semantic Web Anwendungen erstellen lassen. Dabei unterstützt Jena sämtliche gängigen Standards des Semantic Web wie beispielsweise RDF, Sparql oder auch OWL. Ref[JENA]

Wie bereits erwähnt basiert ein erster Ansatz darin eine Java Anwendung zu entwickeln die eine SPARQL Anfrage an zwei verschiedene RDF Stores sendet.

```
1  private void createModel() {
2  images = ModelFactory.createOntologyModel();
```

```
3   InputStream inFoafInstance = FileManager.get().open(
4   "Ontologies/flickrO.owl");
5   images.read(inFoafInstance, flickrNameSpace);
6   inFoafInstance.close();
7       }
8
9   private void queryInferredModel(Model model) {
10  response = runQuery("␣SELECT␣?name␣WHERE␣"
11  + "{␣flickr:CorleoneImage␣flickr:takenIn␣?place.␣?place␣flickr:
        hasName␣?name␣}␣",
12  model, true);
13  response = runQuery("␣SELECT␣?name␣WHERE␣" + "{␣" + "dbpr:" +
        response
14  + "␣dbpp:abstract␣?name␣␣}␣", model, false);
15  }
16
17  private void runQuery(String queryRequest, Model model, boolean
        local){
18  queryStr.append(queryRequest);
19  if (local) {
20  Query query = QueryFactory.create(queryStr.toString());
21  DataSource dataSource = DatasetFactory.create();
22  dataSource.setDefaultModel(model);
23  QueryExecution qexec = null;
24  qexec = QueryExecutionFactory.create(query, dataSource);
25  }
26   else
27   {
28    qexec = QueryExecutionFactory.sparqlService(
29    "http://dbpedia.org/sparql", query);
30   }
31  }
```

Um nun eine SPARQL Anfrage nach dem Ort Corleone an DBPedia und FlickrO zu richten wurde eine Java Anwendung entwickelt. Ein Auszug aus dem Quellcode ist in Listing 3.1 dargestellt. Die Java Anwendung verwendet zu diesem Zweck das Jena Framework. In Listing 3.1 ist zu erkennen das es für das Jena Framework einen Unterschied macht ob ein RDF Store sich auf einem lokalen System befindet oder auf einem entfernten Server. Befindet sich der RDF Store auf einem lokalen System muss der RDF Store zunächst in ein *Model* geladen werden. Das Erstellen des Modell Objekts ist in den Zeilen 2 bis 7 dargestellt. Die Sparql Abfragen sind in den Zeilen 10 bis 15 dargestellt. Die erste Sparql Abfrage ist an den lokalen RDF Store FlickrO gerichtet, die zweite entsprechend an den entfernten DBPedia Sparql Endpoint. Warum hier zwei Sparql Anfragen nötig sind wird im weiteren Verlauf noch näher erläutert.

In den Zeilen 17 bis 29 wird die eigentliche Sparql Abfrage ausgeführt. Interessant ist hier die IF-Abfrage die prüft ob der Parameter *local* gesetzt ist. Ist der Wert local gesetzt muss der *DataSource* das in Zeile 2 instantiierte *Model* übergeben werden. Wird eine entfernte Sparql Abfrage durchgeführt genügt die Angabe des entfernten Sparql Endpoints. Dies ist gleichzeitig der Grund warum zwei Sparql Abfragen benötigt werden.

Ein Nachteil der sich bei der Verwendung von zwei Abfragen zur Kombination von Ontologien für uns ergibt ist das dies nicht mit der von uns geplanten Architektur übereinstimmt. Unser Ziel war es Klassenbeziehungen über *owl:equivalentProperty* und *owl:equivalentClass* herzustellen, indem eine eigens entwickelte Meta Ontologie verwendet wird. Die Meta Ontologie sollte dabei der Definition von Klassenbeziehungen dienen. Zudem sollte es über die Meta Ontologie möglich sein jederzeit eine Erweiterung von Ressourcen möglich zu machen. Auf diese Weise kann die Anwendung jederzeit dynamisch um neue Datenquellen erweitert werden. Dies ist jedoch nur realisierbar wenn alle Daten in einem Triple Store sind und ein Reasoner Inferenzen aus diesen Daten bilden kann.

Die soeben gezeigten Ergebnisse stehen im krassen Gegensatz zu unserem, innerhalb der Veranstaltung gebildeten Verständnis, von der Vernetzung des Semantik Web. Daher wurde dieser erste Ansatz verworfen und nach einer optimierten Lösung gesucht. Die dabei erzielten Ergebnisse werden in den folgenden Abschnitten dargestellt.

4. Linked Data

Die Abfrage von semantischen Daten über einen SPARQL Endpoint ist wie oben beschrieben nicht adäquat für das gestellte Szenario. Das Ziel dieses Projektes ist schließlich die Abfrage mehrerer verteilter Triple Stores, die in Beziehung zu einander stehen. Eine Lösung für das Problem bietet *Linked Data*. Dieser Ansatz von Tim Berners-Lee [BL06] für die Modellierung von Wissen in maschinenlesbarer Form fordert die explizite Verknüpfung von Daten durch Links ähnlich wie sie aus dem herkömmlichen Hypertext bekannt sind. Nach Meinung der Autoren dieser Projektdokumentation entsteht erst dadurch ein Semantik *Web*. Ohne Verknüpfung ist es eher eine lose Ansammlung von Wissensdatenbanken, die per SPARQL auf standardisierte Art und Weise abgefragt werden können.

Linked Data ist hierbei keine neue Technik, sondern beschreibt etablierte Methoden des Webs zur Vernetzung von publizierten Daten miteinander. Durch Einhaltung der folgenden vier Prinzipien der Wissensbeschreibung in RDF können *Dinge*, *Konzepte* usw. verknüpft werden:

- Alle Entitäten sollten mittels URI Referenzen identifiziert werden.
- URI Referenzen sollten über HTTP dereferenzierbar sein.
- Die Daten sollten in RDF/XML oder Turtle Syntax aufbereitet sein.

- Die Daten sollten möglichst reichhaltig mit anderen Informationen im Semantik Web mittels dereferenzierbarer URIs verlinkt sein (rdfs:seeAlso, owl:sameAs)

Laut Tim Berners-Lee sind dies die Regeln die zur Vernetzung des Semantik Web notwendig sind. Die erste Regel schreibt die Verwendung von URIs als Referenzen für alle Entitäten vor, trotz der im Semantik Web gegebenen Möglichkeit Blank Nodes als Referenzen zu verwenden. Blank Nodes führen sozusagen ins Leere und sind damit zu vermeiden.

Durch die Dereferenzierung von URIs mittels HTTP wird es möglich die damit identifizierten Ressourcen z.B. per HTTP GET aufzulösen und damit zu beziehen. So erhält man leichten Zugang zu den verknüpften Daten und kann über die gesetzten Assoziationen traversieren. Die hierbei aufgelösten Daten sollten laut der dritten Regel in den standardisierten Semantik Web Formaten RDF oder Turtle vorliegen. Erst durch die Verwendung dieser Formate können alle Daten auf die gleiche Art und Weise verarbeitet werden.

Der Einsatz von Links in Form von *owl:sameAs* und *rdfs:seeAlso* verbindet Ressourcen unterschiedlicher Wissensdatenbanken bzw. Triple Stores auf der Instanzebene. Auf diese Weise können verschiedene Repräsentationen von Wissen unterschiedlicher Autoren in Beziehung zu einander gebracht werden. Um auf Modellierungsebene einzelne Klassen und Properties zu verbinden wird *owl:equivalentClass* sowie *owl:equivalentProperty*eingesetzt.

Durch Umsetzung dieser Regeln bei der Formulierung von Wissen mittels der Semantik Web Techniken wird eine maschinenbasierte Traversierung des Semantik Web möglich, wie sie z.B. von Software Agents angewendet werden könnte. Zum Einstieg wird die Beschreibung einer Ressource benötigt. Alle hier angegebenen URIs werden per HTTP aufgelöst und dienen als weitere Informationsquellen. Durch die semantische Modellierung dieser Daten kann der Software Agent Rückschlüsse über die erhaltenen Informationen ziehen und so relevante Ergebnisse liefern. Je mehr Verknüpfungen zwischen unterschiedlichen Datenbanken vorhanden sind, desto leichter fällt die Verarbeitung und Aggregation der dort verfügbaren Informationen.

4.1. Geonames Datenbank

Im Zuge der Einführung von Linked Data in diesem Projekt wird auch die frei zugängliche Geonames[1] Datenbank als weitere Wissensdatenbank verwendet. Geonames Daten liegen ebenfalls in semantischer Form vor und unterstützen Linked Data. Unter anderem sind Verknüpfungen zu Orten in DBPedia hinterlegt. Die Geonames Datenbank verfügt über Informationen zu über 8 Millionen geographischen Orten beinhaltet. (Stand 03.2010) Diese Datenbank wird verwendet, da hier deutlich mehr Orte als bei DBPedia enthalten sind und die Datenbank zu einem Ort weitere nahe gelegen Orte

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

liefern kann, wodurch den Nutzern des Urlaubsszenarios mehr Kontext Informationen präsentiert werden können.

Der folgende Abschnitt beschreibt wie Linked Data in diesem Projekt eingesetzt wird.

4.2. Anpassung der Ontologie für Linked Data

Die Ontologie in der ersten entwickelten Version besitzt bereits Verknüpfungen zwischen der FlickrO Ontologie und der DBPedia Ontologie, vgl. Abbildung 2. Um jedoch auf Instanzebene Linked Data zu erreichen, müssen alle *flickr:Place* Instanzen aus dem lokalen Triple Store um eine Verknüpfung mittels *owl:sameAs* Prädikat zu den entsprechenden Instanzen aus der Geonames Wissensdatenbank erweitert werden. Abbildung 4 veranschaulicht das neue Schema. Jede Instanz der Klasse *flickr:Place* besitzt ein *owl:sameAs* Prädikat mit einer Ressource aus Geonames als Objekt. Die Geonames Ressourcen haben wiederum eine Verknüpfung zu der entsprechenden DBPedia Ressource, falls diese vorhanden ist. Die Änderungen an der Ontologie betreffen also hauptsächlich nur die Verknüpfungen der Klasse *flickr:Place* zu dem Geonames Pendant und die Verwendung von *owl:sameAs* bei allen *flick:Place* Instanzen.

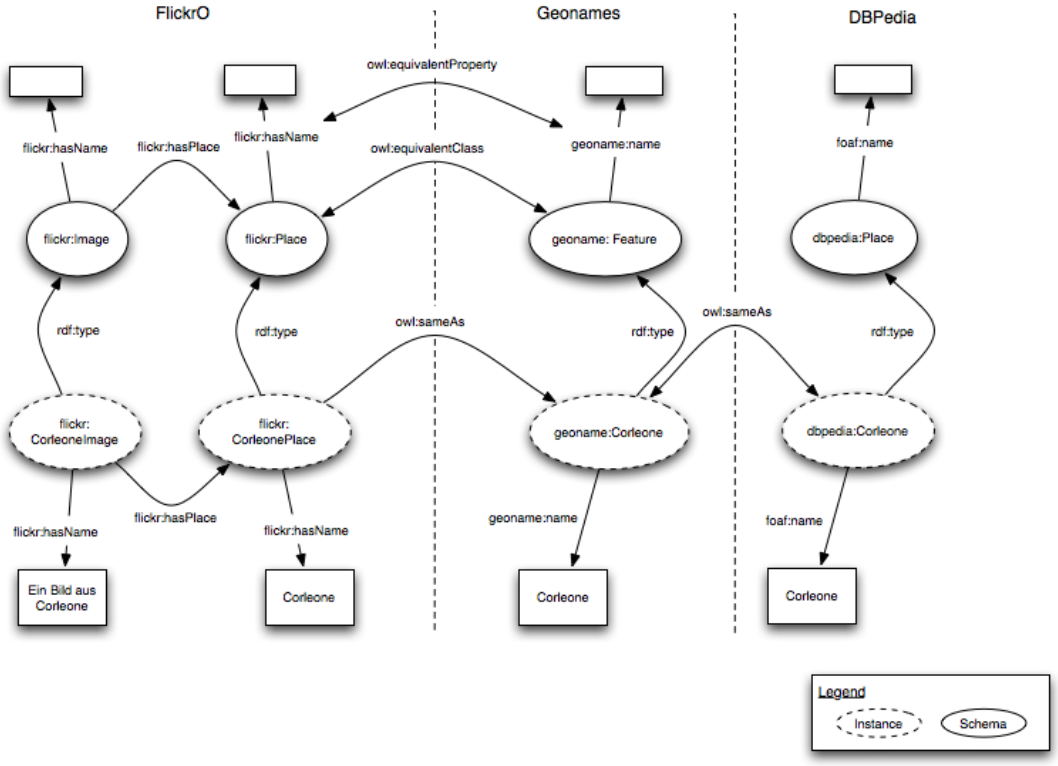

Abbildung 4: Erweiterte Ontologie

Ein entsprechender Eintrag ist in Listing 4.2 zu sehen. Hier besitzt das Subjekt *flickr:CorleonePlace* das Prädikat *owl:sameAs* mit dem Objekt *http://sws.geonames.org/2524928/*. Die URI

ist hierbei die eindeutige Adresse für den Ort Corleone innerhalb der Geonames Datenbank. Beim Aufruf dieser Adresse durch einen Software Agent oder mittels eines herkömmlichen Browser erfolgt eine Weiterleitung. Der Grund dafür liegt darin, dass die Stadt Corleone selbst nicht in der Datenbank enthalten ist, jedoch über eine Beschreibung für diesen Ort bzw. diese Ressource verfügt. Im Falle eine Browsers, welcher als ACCEPT Attribut im Header des HTTP GET Requests *text/html* sendet, wird auf eine HTML Seite weitergeleitet, die die Informationen über den Ort anzeigt. Falls als ACCEPT Header Attribut *application/rdf+xml* gesetzt wird, leitet der Webserver von Geonames auf eine RDF Ausgabe der Informationen weiter. Dieses Verfahren wird als *Content Negotiation* bezeichnet und ist eine bewährte Technik des HTTP, um das passende Format für den Client zu liefern. Weitere Techniken beim Umgang mit Linked Data sind dem Tutorial von Chris Bizer et al. [CB] zu entnehmen.

```
<owl:Thing rdf:about="#CorleonePlace">
        <rdf:type rdf:resource="#Place"/>
        <owl:sameAs rdf:resource="http://sws.geonames.org
            /2524928/"
        <hasName>Corleone</hasName>
        <hasImage rdf:resource="llums-colors-3432680882"/>
    </owl:Thing>
</rdf:RDF>
```

4.3. Architektur des Prototypen

Um Linked Data zu unterstützen muss der lokale Triple Store um eine Linked Data Schnittstelle erweitert werden. Diese Schnittstelle soll HTTP Anfragen mit einer entsprechenden RDF Repräsentation der angeforderten Ressource beantworten oder wie oben beschrieben eine Weiterleitung durchführen. Die Daten von DBPedia und Geonames können bereits per HTTP abgefragt werden und enthalten *owl:sameAs* Links auf gleiche Ressourcen in weiteren Datenbanken des Semantik Web.

Der Software Agent nimmt vom Frontend vorverarbeitete Suchparameter entgegen und sucht zunächst per herkömmlicher SPARQL Query über den Endpoint des Flickr Triple Stores nach einer *flickr:Place* Instanz die den Parametern entspricht. Falls eine Instanz gefunden wurde, wird eine SPARQL Query mit dieser Instanz formuliert und mittels Linked Data beantwortet. Hierbei wird jede in der SPARQL Anfrage verwendete Ressource per HTTP abgefragt und gegebenenfalls weiterverwendet. Alle Ergebnisse der Linked Data Query werden wiederum prozessiert und an das Frontend zurück gegeben. Das Frontend bereiter die gefunden Informationen schließlich für den Benutzer auf.

Zum besseren Verständnis soll kurz die in Listing 4.3 angegebene Linked Data SPARQL Query beschrieben werden. Hier werden zunächst alle Objekte, die über das Prädikat *owl:sameAs* mit der Instanz *flickr:Corleone* verbunden sind, selektiert. Im vorliegenden Fall würde also *?other_place* auf die Ressource *http://sws.geonames.org/2524928/*

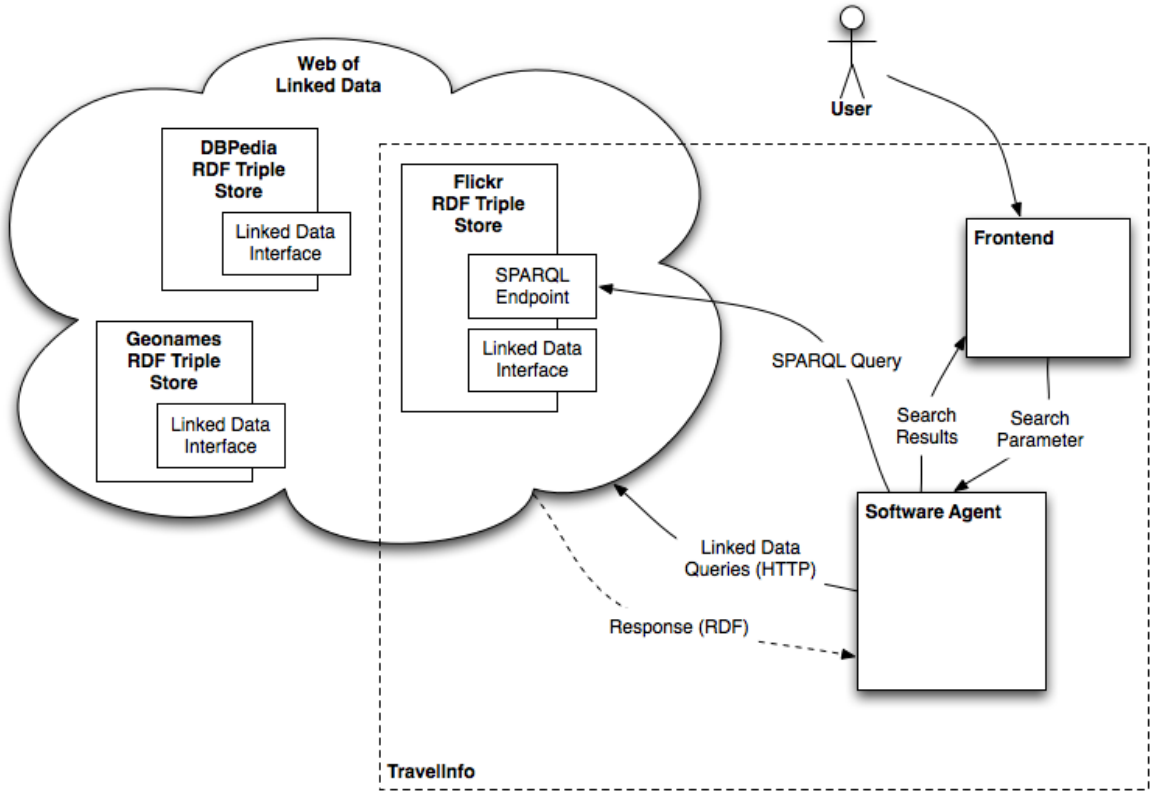

Abbildung 5: Architektur des Prototypen

verweisen, vgl. Listing 4.2. Daraufhin wird per HTTP GET Befehl diese Adresse aufgelöst und die dahinter liegende RDF Beschreibung geladen. Diese enthält das Prädikat *rdfs:label* und wird damit als Teil der Antwort der Query zurück gegeben. Sobald also eine URI, die für die Beantwortung der Query notwendig ist, gegeben ist, wird diese per HTTP GET aufgelöst. Dies geschieht solange die maximale Anzahl an HTTP Requests nicht überschritten wird oder die Query beantwortet wurde.

```
SELECT DISTINCT *
WHERE {
  flickr:Corleone owl:sameAs ?otherPlace .
  ?otherPlace rdfs:label ?label
}
```

4.4. Umsetzung der Architektur

Im folgenden wird die Umsetzung der in Abschnitt 4.3 beschriebenen Architektur geschildert.

4.4.1. Triple Store

Als Triple Store für die FlickrO Ontologie und die Flickr Instanzen wird die Datenbank Virtuoso[2] eingesetzt. Der Virtuoso Server ist in in der Lage Daten in Form von RDF Triples vorzuhalten und bietet einen SPARQL Endpoint zur Abfrage und Manipulation dieser Daten. Der Hersteller bietet den Server sowohl in einer kommerziellen als auch einer freien, quelloffenen Version an. Das Dokument [sta] beschreibt die Installation und den Betrieb des Servers. Nach dem die RDF Triples in den Server geladen wurden, können diese z.B. über das Webinterface des SPARQL Endpoints unter http://localhost:8890/sparql abgefragt werden. 8890 ist der standardmäßig eingestellte Port.

4.4.2. Import der Flickr Daten

Wie in Abschnitt 2 beschrieben, entsprechen die in Flickr abrufbaren Informationen nicht den Standards des Semantik Web, das heißt es können keine in RDF notierten Daten bezogen werden. Da jedoch laut Szenario Bilder zu den Reiseorten angezeigt werden sollen, war es ein Ziel die Photos aus Flickr mit einzubeziehen. Dies gab uns gleichzeitig die Möglichkeit Erfahrungen mit Triple Stores zu sammeln und es entstand die Notwendigkeit Daten aus Flickr in das RDF Format umzuwandeln sowie diese in den lokalen Triple Store zu importieren.

[2] http://virtuoso.openlinksw.com/

Um die Menge an Daten für den Prototypen pragmatisch zu halten, werden nicht zu allen in Geonames verfügbaren Orten Photos aus Flickr gesucht, sondern nur für alle Städte auf der Welt, die über 15.000 Einwohner haben. Diese Daten können von Geonames als Export Datei bezogen werden und wurden zwecks besserer Weiterverarbeitung in eine MySQL Datenbank importiert. Die Daten enthalten den Namen des Ortes, die eindeutige Geonames ID, Längen- und Breitengrad des Ortes, sowie weitere Metadaten. Insgesamt liegen so 21.603 Ortseinträge vor.

Das in der Programmiersprache Ruby geschriebene Skript, welches über eine REST Schnittstelle auf Flickr zugreift, sucht zu jedem Ort im Radius von 3 Kilometern um die Ortskoordinaten maximal 3 Photos. Zu jedem dieser Photos werden mehrere Triples erstellt. In Listing 4.4.2 ist ein Beispiel aufgezeigt. Hierin enthalten sind Angaben zu dem Aufnahmeort des Photos durch das Prädikat *flickr:takenIn* welches auf eine *flickr:Place* Instanz verweist sowie die URIs zu den Bildern bei Flickr. Gleichzeitig erstellt das Skript für jeden Ort Triples, die Referenzen zu den RDF Repräsentationen der Bilder enthalten und über ein *owl:sameAs* Link zu dem Geonames Ort verfügen, vgl. Listing 4.2.

```
<Image rdf:about="ninot-d-andorra-4234560235">
    <takenIn rdf:resource="les-escaldes-1"/>
    <hasName>&amp;apos;</hasName>
    <foaf:page rdf:resource="http://www.flickr.com/photos/
        jorditroguet/4234560235/"/>
    <foaf:depiction rdf:resource="http://farm5.static.flickr.
        com/4051/4234560235_4794067ccb_m.jpg"/>
</Image>
```

Das Skript sammelte so über 30.000 Photos von Flickr und erstellte für jeden Ort RDF Repräsentationen mit Verweisen auf die jeweils zugehörigen Photos. Die so entstanden RDF Dateien wurden anschließend in den Virtuoso Server importiert.

4.4.3. Linked Data Schnittstelle

Um die Flickr Instanzen über HTTP verfügbar zu machen, wird Pubby[3] von der FU Berlin verwendet. Die Pubby Webapplikation ist ein Linked Data Frontend für SPARQL Endpoints. Es ist als Java Servlet implementiert und somit in einem Java Web Server wie Tomcat[4] oder Jetty[5] lauffähig. Die Dokumentation von Pubby [pub] beschreibt die Funktion folgendermaßen: "Pubby will handle requests to the mapped URIs by connecting to the SPARQL endpoint, asking it for information about the original URI, and passing back the results to the client. It also handles various details of the HTTP interaction, such as the 303 redirect required by Web Architecture, and content negotiation between HTML, RDF/XML and N3 descriptions of the same resource."

[3] http://www4.wiwiss.fu-berlin.de/pubby
[4] http://tomcat.apache.org/
[5] http://www.mortbay.org/

Die in diesem Projekt eingesetzte Konfigurationsdatei findet sich im Anhang A.3. Als Web Server wurde Tomcat eingesetzt.

4.4.4. Software Agent

Der Software Agent dient als eine Art Steuerzentrale der Applikation. Dieser wird vom Frontend mit Suchparametern gestartet und führt alle Suchanfragen aus. Zwecks einfacher und schneller Umsetzung wurde in diesem Prototypen der Software Agent in das Frontend integriert, wobei dieser stark gekapselt wurde und somit leicht ausgelagert werden kann.

Zunächst sucht der Software Agent zu den Suchparametern, die zur Zeit nur aus dem Ort bestehen, per SPARQL Query auf den Endpoint des Virtuoso Server eine *flickr:Place*Instanz. Diese Instanz wird daraufhin für die Linked Data Query, wie in Abschnitt 4.3 beschrieben, verwendet. Die Query erfasst dabei, neben den in Geonames verfügbaren Daten, auch DBPedia Information. Im Prototypen wird exemplarisch der Abstract einer DBPedia Ressource erfasst.

Als Bibliothek für die Ausführung von Queries auf dem Web of Linked Data wird die Sematic Web Client Library NG4J[6] der FU Berlin verwendet. Diese Bibliothek nimmt eine reguläre SPARQL Query entgegen, führt die entsprechenden HTTP Requests aus und gibt die gefunden Daten in Form von RDF aus. Diese Daten werden wieder transformiert, so dass sie im Frontend angezeigt werden können.

4.4.5. Frontend

Das Frontend dient der Verarbeitung der Benutzereingaben und der Ausgabe der gefunden Daten. Zur Realisierung wurde das freie und quelloffene Web Framework Ruby On Rails[7] eingesetzt. Hiermit wurde eine Weboberfläche gestaltet, wie sie in Abbildung 6 dargestellt ist.

5. Ausblick

Wie gezeigt wird es mit Hilfe von Linked Data möglich mehrere Ontologien miteinander zu verbinden. Die hier dargestellten Aspekte bilden jedoch nur einen kleinen Ausschnitt und sollten eher als *Proof of Concept* verstanden werden. Dieses Proof of Concept kann nun als Basis für Folgearbeiten verwendet werden. Denkbar wäre beispielsweise die Ausdehnung der Funktionalität des Software Agents. Die Funktionalität des Software Agents

[6] http://www4.wiwiss.fu-berlin.de/bizer/ng4j/semwebclient
[7] http://www.rubyonrails.org

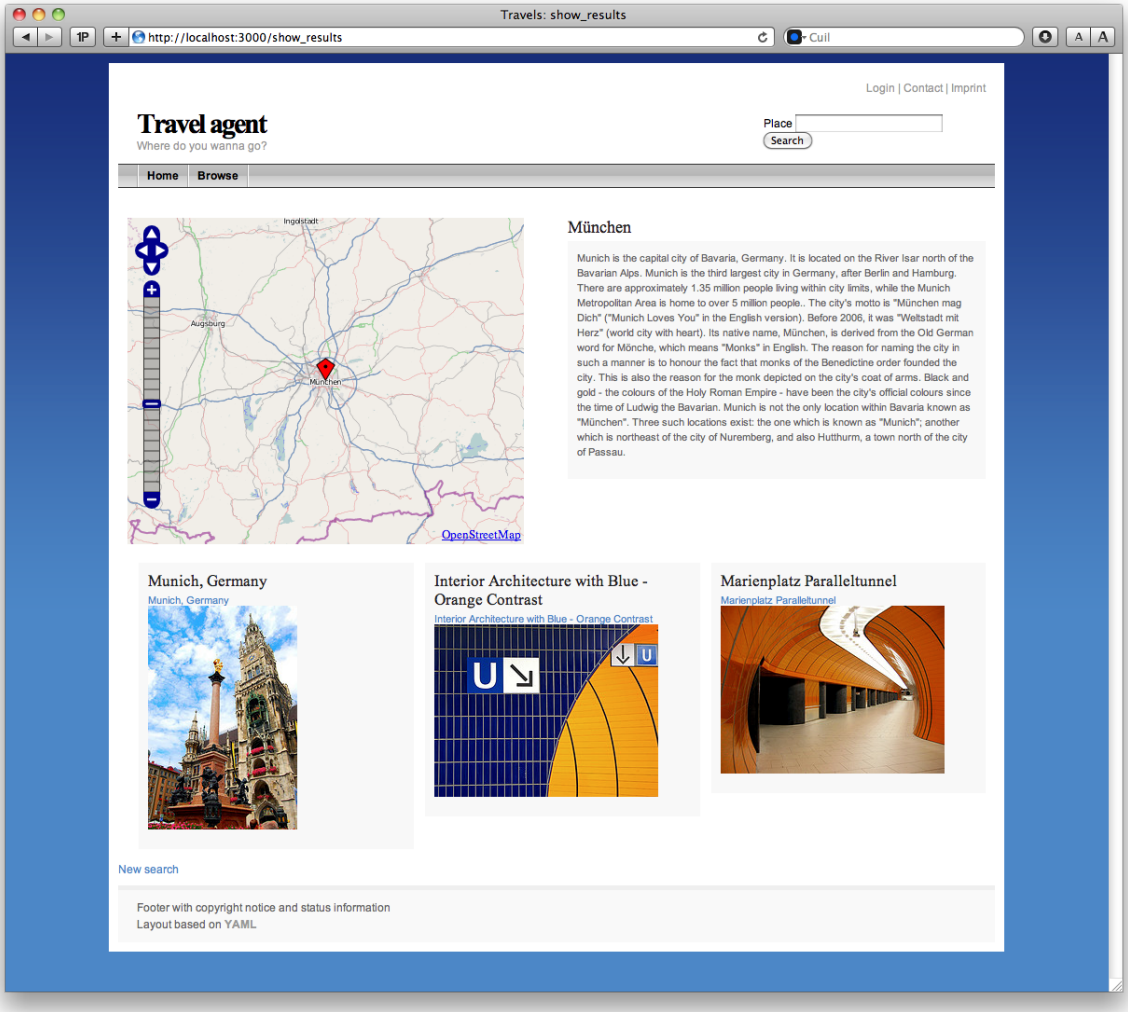

Abbildung 6: Frontend des Prototyps

beschränkt sich in der jetzigen Form auf das eher statische Abfragen mehrere Ontologien und dem anzeigen von Informationen zu Reisezielen. Dieses Szenario könnte nun entsprechend erweitert werden. So wäre ein Software Agent an dieser Stelle wünschenswert der in der Lage ist Anfragen zu interpretieren die in komplexerer Form vorliegen. Beispielsweise könnte eine Anfrage des Nutzers verschiedene Orte in Abhängigkeit von Eingrenzungen wie beispielsweise bestimmte Zeitzonen, Wetterlage , Bevölkerungsdichte.. ausgrenzen. Aspekte die in der Umsetzung dieses Projektes nicht realisiert wurden sind beispielsweise das Bilden der ursprünglich angedachten Meta Ontologie. Diese würde eine dynamische Erweiterung der verwendeten Quellen ermöglichen und wäre sicherlich eine wünschenswerte Ergänzung für das dargestellte Szenario. So wäre es denkbar das ein Nutzer Quellen die für die Suche relevant sind in die Suche mit übernimmt.

6. Fazit

In der hier dargestellten Projektdokumentation wurde eine prototypische Anwendung implementiert die der Kombination von verschiedenen dezentralen Ontologien dient. Dabei wurden insgesamt drei Projektiterationen diskutiert. Bei der Projektumsetzung hat sich gezeigt das sich die theoretischen Konzepte des Semantic Web in der Realität zwar umsetzten lassen dies sich jedoch als wesentlich schwieriger darstellte als zuvor angenommen. Innerhalb der Projektphase hat sich zudem gezeigt das sich einige Konzepte nur über Umwege realisieren lassen. Hier sei noch einmal die Problematik bei dem Bilden einer Meta Ontologie erwähnt. Ziel war es eine dynamisch erweiterbare Meta Ontologie zu bilden die Informationen zu Klassenbeziehungen aus verschiedenen Ontologien beinhaltet. Die Realisierung dieser Meta Ontologie war an dieser Stelle nicht umzusetzen. Jedoch hat sich das Konzept der Linked Data als vielversprechende Alternative erwiesen. Mittels Linked Data kommt man der Vision eines *Web of Data* bereits sehr nahe.

Zusammenfassend kann man sagen das das Projekt so wie es zu Beginn geplant war erfolgreich umgesetzt werden konnte und das die theoretischen Konzepte die uns bereits in Veranstaltungen vermittelt wurden nun durch Praxiserfahrung ergänzt wurden. Dennoch ist das Gebiet des Semantic Web sehr umfangreich so das in dieser Arbeit lediglich ein kleiner Ausschnitt aufgezeigt werden konnte. Offen bleibt beispielsweise wie man komplexe, auf Reasoning beruhenden Anfragen generieren kann. Dieser Bereich bietet sicherlich Potential für Folgeprojekte.

Literatur

[BL06] Tim Berners-Lee. Linked data - design issues. 2006. http://www.w3.org/DesignIssues/LinkedData.html, Zuletzt besucht am 06.03.10.

[CB] Tom Heath Chris Bizer, Richard Cyganiak. How to publish linked data on the web. FU Berlin. http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/, Zuletzt besucht am 06.03.10.

[pub] Pubby dokumentation. http://www4.wiwiss.fu-berlin.de/pubby, Zuletzt besucht am 05.03.10.

[sta] Getting started with virtuoso. http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOS Zuletzt besucht am 05.03.10.

A. Anhang

A.1. Szenarien

Szenario: Urlaubsplanung Die Familie Kunz plant einen Urlaub in nächster Zeit durchzuführen. Hierzu möchten die Kunzes einen groben Eindruck von dem Zielland in Bild und Schrift bekommen. Sie starten die Software und geben in die Suchmaske den Namen eines Landes, einer Stadt, Gegend ein.
Daraufhin analysiert der Software–Agent die Anfrage und liefert dazu aus den ihm bekannten Quellen verschiedene Informationen wie Texte und Bilder. Gleichzeitig bietet die Software weitere Einstellungmöglichkeiten an, um z.B, den Zeitraum für den Urlaub einzuschränken oder den Umkreis der angezeigten Informationen zu dem gefunden Ort zu begrenzen.

Stefan: Stefan R. ist regelmäßiger Flickr Nutzer. Mehrmals die Woche betrachtet er Fotos von Freunden oder auch anderen Mitgliedern von Flickr. Von Zeit zu Zeit tritt jedoch der Fall auf das Stefan ein Bild betrachtet dessen Hintergrund ihm nicht bekannt ist. Um weitere Informationen über das Bild zu erhalten betrachtet Stefan die Tags die dem Bild hinzugefügt wurden. Möchte Stephan weiter Informationen zu einem der Tags ermitteln ruft er entweder Google auf oder besucht Wikipedia um detaillierte Infos zu erhalten. Hack soll es nun Stefan ermöglichen komortabel Bilder auf Flickr zu betrachtet und ohne Umwege zu den Bildinformationen zu gelangen. Zu diesem Zweck ruft Stefan Hack auf bewegt sich wie von Flickr gewohnt durch die Bildersammlung und falls er Zusatzinformationen benötigt kann er sich diese Anzeigen lassen ohne Hack zu verlassen.
Frank :
Frank ist seit jahren glühender Heavy Metall Fan. Was auch immer er an neuer Musik in die Finger bekommt wird von ihm regelrecht verschlungen. Er betrachtet sich wie er selbst sagt als aktiver Fan der versucht so oft wie möglich einen Liveauftritt der Bands zu besuchen statt die Musik nur über Umwege zu konsumieren. V1: Oft kommt es vor das sich Frank gerne ein detaillierteres Bild von der Band / dem Künstler machen möchte den er grade hört. Dazu ruft er zunächst Wikipedia auf und liest sich den entsprechenden Artikel durch. Um einen Einblick in die Bühnenperformance des Künstlers zu erhalten besuchter er Flickr und sucht dort

nach den Tags "Name␣des␣Künstlers" und "Konzert" bzw. "Live" . Das neue Hack–Plugin gibt Frank die Möglichkeit all diese Informationen direkt aus seinem Player aufzurufen. Zu diesem Zweck gibt Frank lediglich den Namen der Band ein und der Player präsentiert ihm sowohl Informationen über die Band als auch Fotos die von Konzerten geschossen wurden. V2: Ein Bekannter hat Frank von einem wahnsinns Konzert einer Metall Band in Stockholm erzählt. Frank möchte sich nun einen Eindruck davon verschaffen und ruft zu diesem Zweck Hack auf . Leider wusste sein Bekannter den Namen der Band nicht mehr so das Frank mit den ihm bekannten Infos versucht den Namen der Band zu ermitteln. Er öffnet sein HackInterface und gibt als Suchbegriff "Hevymetal" "Konzert" "Live" "Stockholm " "27.03.2009" ein. Hack findet 3 passende Interpreten und liefert Frank sowohl Hintergrundinformationen zu der Band als auch Bilder von dem Auftritt.

Bernd:

Bernd ist schon von Kindesbeinen an glühender Werder Bremen Fan . Wann immer ihm sein Job und seine Freundin die Möglichkeit geben versucht er in das Weser–Stadion zu fahren und sich ein Live–Spiel seiner Werder–Elf anzuschauen. V1: Vor einigen Tagen entbrannte zwischen Bernd und Marc, seinem besten Freund, eine heftige Diskussion über die Spieler der Meisterschaftssaison 2002/2003. Um den Streit zu schlichten ruft Bern schliesslich Wikipedia auf und sucht Informationen zu dieser Saision. Als erster gibt er als Suchbegriff " Werder␣Bremen" ein. Dort findet er nach einigem Suchen einen Hinweis auf den Gewinn der Meisterschaft 2002/2003. Bernd Klickt auf den Link und betrachtet den Artikel. Dort findet er die Namen der Spieler. Mit Hack wird Ihm der Umweg erspart. Er ruft sein HackInterface auf und gibt als Suchbegriff Werder Bremen Spieler der Saison 2002/2003 auf. Hack präsentiert ihm die Namen der Spieler und auch ein Bild zu jedem der Spieler V2: Vor einigen Tagen entbrannte zwischen Bernd und seinem Besten Freund Ulf eine heftige Diskussion über den Spieler der in der Meistersaison 2002/2003 die Rückennummer 7 trug. Bernd war sich sicher das es Torben Marx war wohingegen Ulf sich sicher war das es sich um Krassimir Balakov gehandelt haben muss. Um den Streit zu schlichten ruft Bernd sein Hack Interface auf und gibt als Suchbegriff ein Werder Bremen Saison 2002/2003 Spieler Rückennummer 7 ein. Hack präsentiert das Ergebnis inklusive einem Bild des Spielers und zusätzlichen Hintergrundinformationen zu dem Spieler und der Saison.

A.2. Quellcode

Der Quellcode für das Flickr Import Skript, das Frontend und den Software Agent befindet sich in Versionskontrolle unter `http://github.com/elitau/semantic_hack`.

Der erste Prototyp für den Software Agent als Webservice befindet unter `http://github.com/elitau/SemanticJavaFlickr`.

A.3. Pubby Konfigurationsdatei

```
# Pubby Example Configuration
#
# This configuration connects to the DBpedia SPARQL endpoint
   and
# re-publishes on your local machine, with dereferenceable
# localhost URIs.
#
# This assumes you already have a servlet container running
# on your machine at http://localhost:8080/ .
#
# Install Pubby as the root webapp of your servlet container,
# and make sure the config-file parameter in Pubby's web.xml
# points to this configuration file.
#
# Then browse to http://localhost:8080/ .

# Prefix declarations to be used in RDF output
@prefix conf: <http://richard.cyganiak.de/2007/pubby/config.rdf
   #> .
@prefix meta: <http://example.org/metadata#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix dbpedia: <http://localhost:8080/resource/> .
@prefix p: <http://localhost:8080/property/> .
@prefix yago: <http://localhost:8080/class/yago/> .
@prefix units: <http://dbpedia.org/units/> .
```

```
@prefix␣geonames:␣<http://www.geonames.org/ontology#>␣.
@prefix␣prv:␣␣␣␣␣␣<http://purl.org/net/provenance/ns#>␣.
@prefix␣prvTypes:␣<http://purl.org/net/provenance/types#>␣.
@prefix␣flickr:␣<http://localhost:8080/flickr/>␣.

#␣Server␣configuration␣section
<>␣a␣conf:Configuration;
␣␣␣␣#␣Project␣name␣for␣display␣in␣page␣titles
␣␣␣␣conf:projectName␣"DBpedia.org";
␣␣␣␣#␣Homepage␣with␣description␣of␣the␣project␣for␣the␣link␣in␣
    the␣page␣header
␣␣␣␣conf:projectHomepage␣<http://neuezone.org>;
␣␣␣␣#␣The␣Pubby␣root,␣where␣the␣webapp␣is␣running␣inside␣the␣
    servlet␣container.
␣␣␣␣conf:webBase␣<http://localhost:8080/>;
␣␣␣␣#␣URL␣of␣an␣RDF␣file␣whose␣prefix␣mapping␣is␣to␣be␣used␣by␣
    the
␣␣␣␣#␣server;␣defaults␣to␣<>,␣which␣is␣*this*␣file.
␣␣␣␣conf:usePrefixesFrom␣<>;
␣␣␣␣#␣If␣labels␣and␣descriptions␣are␣available␣in␣multiple␣
    languages,
␣␣␣␣#␣prefer␣this␣one.
␣␣␣␣conf:defaultLanguage␣"de";
␣␣␣␣#␣When␣the␣homepage␣of␣the␣server␣is␣accessed,␣this␣
    resource␣will
␣␣␣␣#␣be␣shown.
␣␣␣␣#␣conf:indexResource␣<http://dbpedia.org/resource/Wikipedia
    >;
␣␣␣␣conf:indexResource␣<http://localhost:8080/flickr/Corleone>;

#␣Dataset␣configuration␣section␣#1␣(for␣flickr␣resources)
#
#␣URIs␣in␣the␣SPARQL␣endpoint:␣http://dbpedia.org/resource/*
#␣URIs␣on␣the␣Web:␣␣␣␣␣␣␣␣␣␣␣␣␣http://localhost:8080/flickr/*
␣␣␣␣conf:dataset␣[
␣␣␣␣␣␣␣␣#␣SPARQL␣endpoint␣URL␣of␣the␣dataset
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣conf:sparqlEndpoint␣<http://localhost:8890/
    sparql>;
␣␣␣␣␣␣␣␣#␣Default␣graph␣name␣to␣query␣(not␣necessary␣for␣most␣
    endpoints)
␣␣␣␣␣␣␣␣#␣conf:sparqlDefaultGraph␣<http://www.neuezone.org
    /2009/11/17/flickr.owl#>;
␣␣␣␣␣␣␣␣#␣Common␣URI␣prefix␣of␣all␣resource␣URIs␣in␣the␣SPARQL␣
    dataset
```

```
        conf:datasetBase <http://localhost:8080/flickr/>;
        # Will be appended to the conf:webBase to form the
            public
        # resource URIs; if not present, defaults to ""
        conf:webResourcePrefix "flickr/";
        ##################################### conf:addSameAsStatements "
            true";
        ################ conf:addSameAsStatements "true";
        # Fixes an issue with the server running behind an
            Apache proxy;
        # can be ignored otherwise
        conf:fixUnescapedCharacters "(),'!$&*+;=@";
    ];


# Dataset configuration section #2 (for DBpedia resources)
#
# URIs in the SPARQL endpoint: http://dbpedia.org/resource/*
# URIs on the Web: ############ http://localhost:8080/resource/*
#     conf:dataset [
#         # SPARQL endpoint URL of the dataset
#         conf:sparqlEndpoint <http://dbpedia.org/sparql>;
#         # Default graph name to query (not necessary for most
            endpoints)
#         conf:sparqlDefaultGraph <http://dbpedia.org>;
#         # Common URI prefix of all resource URIs in the
            SPARQL dataset
#         conf:datasetBase <http://dbpedia.org/resource/>;
#         # Will be appended to the conf:webBase to form the
            public
#         # resource URIs; if not present, defaults to ""
#         conf:webResourcePrefix "resource/";
#         # Fixes an issue with the server running behind an
            Apache proxy;
#         # can be ignored otherwise
#         conf:fixUnescapedCharacters "(),'!$&*+;=@";
#
#         # include metadata
#         conf:metadataTemplate "metadata.n3";
#
#         # configure your metadata here
#         # Use properties with the meta: prefix where the
            property name
```

```
#␣␣␣␣␣␣␣␣#␣corresponds␣to␣the␣placeholder␣URIs␣in␣metadata.n3␣
    that␣begin
#␣␣␣␣␣␣␣␣#␣with␣about:metadata:metadata:
#␣␣␣␣␣␣␣␣#␣Examples␣for␣such␣properties␣are:
#␣#␣␣␣␣␣␣␣meta:pubbyUser␣<URI␣of␣the␣data␣publisher␣who␣uses␣
    this␣Pubby>;
#␣#␣␣␣␣␣␣␣meta:pubbyOperator␣<URI␣of␣the␣service␣provider␣who␣
    operates␣this␣Pubby>;
#␣#␣␣␣␣␣␣␣meta:endpointUser␣<URI␣of␣the␣data␣publisher␣who␣
    uses␣the␣SPARQL␣endpoint␣queried␣by␣this␣Pubby>;
#␣#␣␣␣␣␣␣␣meta:endpointOperator␣<URI␣of␣the␣service␣provider␣
    who␣operates␣the␣SPARQL␣endpoint>;
#␣␣␣␣␣];

#␣Dataset␣configuration␣section␣#3␣(for␣DBpedia␣classes␣and␣
    properties)
#
#␣URIs␣in␣the␣SPARQL␣endpoint:␣http://dbpedia.org/class/*
#␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣http://dbpedia.org/property/*
#␣URIs␣on␣the␣Web:␣␣␣␣␣␣␣␣␣␣␣␣␣␣http://localhost:8080/class/*
#␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣http://localhost:8080/property/*
␣␣␣␣conf:dataset␣[
␣␣␣␣␣␣␣conf:sparqlEndpoint␣<http://dbpedia.org/sparql>;
␣␣␣␣␣␣␣␣conf:sparqlDefaultGraph␣<http://dbpedia.org>;
␣␣␣␣␣␣␣␣conf:datasetBase␣<http://dbpedia.org/>;
␣␣␣␣␣␣␣␣#␣Dataset␣URIs␣are␣mapped␣only␣if␣the␣part␣after␣the
␣␣␣␣␣␣␣␣#␣conf:webBase␣matches␣this␣regular␣expression
␣␣␣␣␣␣␣␣conf:datasetURIPattern␣"(class|property)/.*";
␣␣␣␣␣␣␣␣conf:fixUnescapedCharacters␣"(),'!$&*+;=@";
␣␣␣␣];
␣␣␣␣.
#␣Dataset␣configuration␣section␣#2␣(for␣flickr␣classes␣and␣
    resources)
#
#␣URIs␣in␣the␣SPARQL␣endpoint:␣http://dbpedia.org/class/*
#␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣http://dbpedia.org/property/*
#␣URIs␣on␣the␣Web:␣␣␣␣␣␣␣␣␣␣␣␣␣␣http://localhost:8080/class/*
#␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣http://localhost:8080/property/*
␣␣␣␣#␣conf:dataset␣[
␣␣␣␣#␣␣␣␣␣conf:sparqlEndpoint␣<http://localhost:2020/sparql>;
␣␣␣␣#␣␣␣␣␣#␣conf:sparqlDefaultGraph␣<http://www.neuezone.org/
    semweb/>;
␣␣␣␣#␣␣␣␣␣conf:datasetBase␣<http://www.neuezone.org/semweb/>;
```

```
    #                              conf:webResourcePrefix "flickr/
    ";
    #     #  Dataset URIs are mapped only if the part after the
    #     #  conf:webBase matches this regular expression
    #     conf:datasetURIPattern "(semweb)/.*";
    #     conf:fixUnescapedCharacters "(),'!$&*+;=@";
    # ];
    # .
```