Masterarbeit

SPARQL-Query Wizard

Karl Kappaun, BSc.

Knowledge Management Institute (KMI)
Technische Universität Graz
Vorstand: Univ.-Prof. Dipl.-Ing. Dr. Stefanie Lindstaedt

Begutachter: Univ.-Prof. Dipl.-Ing. Dr. Stefanie Lindstaedt
Betreuer: Dipl.-Ing. (FH) Patrick Höfler

Graz, im September 2012

Kurzfassung

Das Internet entwickelt sich von einer Sammlung miteinander verknüpfter Dokumente hin zu einem interaktiven Medium, in dem der Begriff der „Bedeutung“ mit der vermehrten Veröffentlichung von strukturierten, untereinander verlinkten und für Maschinen verständlichen Daten eine große Rolle spielt. Im Kontext dieser Arbeit wird die Entwicklung eines „Semantic Web“ und der damit verwandten Technologien, wie das „Resource Description Framework“ (RDF) oder die Abfragesprache SPARQL erläutert, und ein Wizard zur automatisierten Generierung von Abfragen an Repositories der „Linked Open Data Cloud“ entwickelt. Mit diesem SPARQL-Wizard soll es für einen User auf möglichst einfache Art und Weise möglich sein, die Vorteile des Semantic Web bei der Informationsbeschaffung zu nutzen.

Abstract

The Internet evolves from a simple collection of linked documents and a user which is only a consumer of the offered information, to an more interactive space, where the concept of „meaning“ gains more and more importance. The Semantic Web is born and comes with an evolution of interlinked, machine understandable structured data. In the context of this work the development of the Semantic Web and its related technologies such as the Resource Description Framework (RDF) or its query language SPARQL is described in detail, and a wizard for creating and processing SPARQL queries in a simple and intuitive way has been developed, so that every user could take advantage of the benefits of the Semantic Web.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz,am ....
(Unterschrift)

Danksagung

Diese Diplomarbeit wurde im Jahr 2012 am Institut für Wissensmanagement an der Technischen Universität Graz durchgeführt.

Ich möchte mich recht herzlich bei meinem Betreuer Dipl.-Ing. (FH) Patrick Höfler für sein Engagement, seine Zeit und die Hilfestellung bei der Umsetzung dieser Arbeit bedanken. Ebenso bei Prof. Dr. Michael Granitzer, der mich im Laufe meiner Studienzeit immer wieder mit spannenden Themen begeistern konnte und mir am Beginn dieser Arbeit bei Fragen und Unklarheiten zur Seite stand. Ein besonderes Dankeschön geht an Univ.-Prof. Dr. Stefanie Lindstaedt, die nach dem Abgang von Dr. Granitzer die Betreuung meiner Arbeit übernommen hat.

Weiterer Dank gilt meinen Eltern, die mich immer bei meinen, hin und wieder durchaus auch fragwürdigen, Vorhaben unterstützt haben, sowie allen Freunden und weiteren Lieben, die doch auch das eine oder andere mal meine Launen im Zuge der Fertigstellung dieser Arbeit zu spüren bekommen haben.

Danke!

Graz, im September 2012 Karl Kappaun

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Das „Semantic Web“ hat sich in den letzten Jahren zu einem nicht mehr wegzudenkenden Teil des Internets entwickelt und die damit einhergehenden Technologien sind für den User bewusst oder unbewusst fixer Bestandteil vieler Anwendungen geworden. Die Idee des Konzepts von „Linked Open Data“, des freien öffentlichen Zugangs zu untereinander verlinkten, über das Netz zugänglichen Datensätzen in Form von RDF-Repositories, ist faszinierend, und bietet eine Vielzahl von Möglichkeiten in der Verwendung dieser Informationen. Für den Zugriff auf diesen enormen Datenpool existieren eine Vielzahl von Applikationen und Schnittstellen, deren Großteil allerdings nur für erfahrene Anwender und Programmierer sinnvoll anwendbar ist, ebenso wie der direkte Zugriff über die Abfragesprache für RDF-Daten, SPARQL. Halevy (2012) bringt das Problem auf den Punkt:

> ... we need to create a ecosystem that makes it easier for users to discover, manage, visualize and publish structured data on the Web...

Technisch unbedarfte User können sich zwar über spezielle Browser oder Interfaces Zugang zu den Daten verschaffen, diese sind aber oft eingeschränkt in ihrer Funktionalität, nicht sehr benutzerfreundlich und oft für einen speziellen Anwendungsfall vorgesehn. Es herrscht ein Bedarf an einem allgemeinen Tool zur Erstellung individueller, repositoryübergreifender Abfragen, das überdies noch einfach in der Bedienung ist. Motivation und Ziel dieser Arbeit ist die Entwicklung eines einfachen Abfragesystems für RDF-Repositories mit dem ein User in der Lage ist individuelle Anfragen an Repositories zu definieren, zu bearbeiten, und auch zu speichern, ohne mit der darunterliegenden Abfragesprache SPARQL in Berührung zu kommen.

1.2 Zielsetzung

Das Ziel der vorliegenden Arbeit ist die Erarbeitung eines SPARQL-Query Wizard. Darunter ist ein System oder einen Assistent zur Erstellung von SPARQL-Abfragen über ein einfach zu bedienendes Userinterface zu verstehen.

Der Assistent soll dabei so einfach wie möglich aufgebaut, und auch von Benutzern ohne Vorwissen im Bezug auf RDF und der dazugehörigen Abfragesprache SPARQL, bedienbar sein. Zu diesem Zweck soll der Wizard den User über eine iterative Vorgehensweise

durch den Prozess führen. Der User soll ausgehend von einem Startpunkt seine Abfrage aufbauen und auf Basis von Teilergebnissen weiter ausbauen, bis er die von ihm erwarteten Informationen erhalten hat.

Der Wizard soll webasierend und mit gängigen Browsern kompatibel sein. Die Generierung der Abfragen soll vom User unabhängig im Hintergrund erfolgen. Dies hat zur Folge , dass der User mit der eigentlichen Query nicht in Berührung kommen muss. Die Kontrolle über die Abfrage erfolgt rein über einfache Userinterface-Elemente wie Suchfelder, Drag- and Drop-Elementen oder Dropdownmenüs, wie sie in den meisten Webanwendungen Anwendung finden und dem Großteil der Internetuser bekannt sein dürften.

Der Wizard soll in seiner ersten Version einfache Sprachkonstrukte der Abfragesprache SPARQL implementieren, und über ein entsprechendes Interface für den User zugänglich machen.

Hat der User seine Abfrage zu seiner Zufriedenheit generiert, ist er in der Lage diese zu speichern und zu einem späteren Zeitpunkt fortführen zu können.

Auf Administrationsebene soll der Wizard mit möglichst vielen Repositories ohne aufwendige Konfiguration kompatibel sein, und auch einfach mit weiteren Repositories erweitert werden können.

1.3 Gliederung

Die weitere Arbeit orientiert sich an folgendem Aufbau:

Der Abschnitt 2 beginnt mit einem theoretischen Überblick über den inhaltlichen Kontext der Arbeit. Es werden die Hintergründe und die Motivation, die zur Entstehnung des „Semantic Web“ geführt haben, erörtert, und es wird auf die zur Umsetzung des Semantic Web entwickelten Technologien eingegangen. Insbesondere wird in den jeweiligen Unterkapiteln auf das „Resource Description Framework (RDF)“, dessen Beschreibungssprache „RDF Schema“ (RDFS), die Abfragesprache „SPARQL“ und das „Linked Open Data“ Projekt beschrieben. Diese Technologien stellen die Grundlagen des in dieser Arbeit entwickelten SPARQL-Wizard dar. Zum Abschluss dieses Abschnittes wird DBpedia als ein spezielles RDF-Repository vorgestellt. DBpedia dient auch als Referenzrepository für die im Abschnitt 5 beschriebene Implementierung.

Der nächste Abschnitt 3 beschreibt einige bereits existierende Systeme und Tools, die für die Generierung von SPARQL-Queries herangezogen werden können, und diskutiert deren Stärken und Schwächen im Bezug auf den in dieser Arbeit entwickelten Query-Wizard. Es werden unterschiedliche Herangehensweisen für verschiedene Problemstellungen an Hand von Beispielen, angefangen bei simplen RDF-Browsern und Suchmaschinen in ihren verschiedenen Ausprägungen, über visuelle SPARQL Query Builder, bis hin zu Lösugen mit Künstlicher Intelligenz, besprochen.

Im weiteren Verlauf der Arbeit werden in den Abschnitten 4 und 5 auf die eigentliche Umsetzung des Query-Wizard im Sinne von Design/Architektur und Implementierung eingegangen. Es werden auf Grund der Erfahrungen im letzten Abschnitt Requirements

erarbeitet und diskutiert, die technische Umsetzung mit den für die Entwicklung des Prototypen verwendeten Technologien beschrieben, sowie die Arbeitsweise des Wizards anhand von expliziten Beispielen erläutert.

Der Letzte Abschnitt 6 beschäftigt sich mit den tatsächlichen Ergebnissen der Arbeit in Hinblick auf die am Anfang definierten Ziele und leitet daraus mögliche Erweiterungen und Änderungen für zukünftige Arbeiten ab.

Kapitel 2

Semantic Web

Dieses Kapitel gibt einen Überblick über die Hintergründe des Semantic Web und beschreibt seine Entstehungsgeschichte sowie dessen Begrifflichkeit. Dabei wird eine Abgrenzung des Semantic Web vom Begriff der Semantischen Technologien getroffen, und auf die mit dem Begriff Semantic Web verbundenen Technologien, wie beispielsweise das Resource Description Framework (RDF)[1], oder die Abfragesprache SPARQL[2] eingegangen. Den Abschluss dieses Kapitels bildet ein Überblick über das Linked Data Konzept mit DBpedia[3] als Beispiel.

2.1 Hintergründe

Die Entwicklung des Internets geht von einer reinen Sammlung miteinander verknüpfter Dokumente, hin zu einem interaktiven Medium, in dem der „User“ nicht mehr nur die Rolle des reinen Konsumenten innehat, sondern selbst zum Schöpfer neuer Inhalte wird. Er erschafft „user generated content“ und interagiert mit anderen Usern über diverse Plattformen und soziale Netzwerke. Der User wird aktiver, will mitgestalten und sich kundtun.

Ebenso wie sich das Benutzerverhalten zu einem immer mehr interaktiven Verhalten entwickelt, entwickelt sich auch die Webentwicklung selbst. Sie entfernt sich immer weiter von der Erstellung rein statischer Inhalte. Viel mehr geht es heute um die Generierung von dynamischen Webapplikationen, die immer komplexere Aufgaben erfüllen. Der Bedarf an immer umfangreicheren und informationshungrigen Anwendungen führte schließlich dazu, dass einige Anbieter ihre Daten in strukturieter Form über Web Service Standards für andere Applikationen zur Verfügung stellen. Beispiele hierfür sind bekannte Unternehmen wie Ebay[4] oder Amazon[5].

Trotz des geänderten Userverhaltens und der damit verbundenen Reaktion der Content-Provider kann man die grundlegenden Aktionen die wir mit dem Internet durchführen laut Yu (2011) in drei Kategorien zusammenfassen:

[1] `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`
[2] `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`
[3] `http://dbpedia.org`
[4] `http://www.ebay.at/`
[5] `https://www.amazon.de/`

- **Suche**
  Die Suche nach Informationen ist wohl die meist genutzte Anwendung des Webs. Dabei geht es um das Auffinden von spezifischen Daten oder Resourcen. Da die meisten Suchmaschinen nach dem Prinzip der Schlüsselwortsuche arbeiten, ist es Sache des Users relevante Informationen von irrelevanten zu trennen. Die Maschine weiß nichts über die Domäne der genannten Suchanfrage und liefert sämtliche Dokumente in denen der Suchbegriff vorkommt. Das Ziel muss es sein, nur die relevanten Suchresulate zu erhalten. Da das Web in seiner Urform aber rein auf die (menschlichen) User ausgerichtet ist, weiß es nichts über seine eigentlichen Inhalte und kann diese auch nicht nach deren Domäne unterscheiden.

  Dieses Mantra der „dummen“ Schlüsselwortsuche galt bis vor kurzem. Der Suchmaschinenbetreiber Google[6] geht mit dem seit 16. Mai 2012 in den USA aktivierten „Google Knowledge Graph“[7] einen neuen Weg. Der „Knowledge Graph“ ist eine von Google betriebene Wissensbasis mit dem Ziel, Suchergebnisse und Anfragen nicht mehr rein als Strings zu interpretieren, sondern vielmehr als Dinge zu verstehen. Entitäten, die mit anderen Dingen in Beziehung stehen. Der User profitiert von diesem semantischen Ansatz, indem Google dem User in einem eigenen Bereich der Resultatseite und auch schon während der Eingabe des Suchbegriffes, die Möglichkeit der Filterung der Ergebnisse nach einer bestimmten Domäne anbietet, und so bessere Ergebnisse zu seiner Anfrage erhält.

- **Informationsintegration**
  Unter Informationsintegration versteht das Zusammenführen von verschiedenen Datenquellen wie beispielsweise unterschiedliche Web-Services zur Erfüllung eines Tasks. Auch hier liegt das Problem bei der manuellen Auswahl und der Auffindung der benötigten Web Services. Ein automatisiertes Auffinden und Integrieren der benötigten Services ist nich möglich, da das Web wie schon oben erwähnt keine Information über sich selbst aufweist.

- **Web Data Mining**
  Web Data Mining beschreibt einfach gesagt den Vorgang, sinnvolle Daten aus den gesammelten Informationen des World Wide Web, das man auch als riesige dezentralisierte Datenbank ansehen kann, zu extrahieren. Hierfür bedient man sich meist hochspezialisierter Software, so genannter Crawler oder Agents die für genau den einen benötigten Anwendungsfall geschrieben wurden. Genau in dieser Eigenschaft liegt ihr Problem. Ihre Spezialisierung bedarf eine vordefinierte Spezifikation der Parameter, nach denen die Applikation das Netz durchsuchen soll. Diese stehen von vornherein fest und ihre Bedeutung für das gesuchte Ergebnis ist durch die Programmierer definiert. Die Applikation kann nicht selbstständig entscheiden ob eine Information für das gewünschte Resultat relevant ist, oder nicht.

Zusammenfassend kommt man zu dem Schluss, dass ein Dokument wie man es im Internet findet, in seiner jetzigen Form für einen Menschen durchaus Sinn ergibt, für eine Maschine aber nur genau so viele Informationen beinhaltet, wie sie benötigt um dieses

[6] `http://www.google.com`
[7] `http://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html`

Dokument in einer für den Menschen angenehmen Form anzeigen zu können, aber nicht um den Inhalt des Dokumentes zu verstehen (siehe Yu, 2011).

Der Begriff der Semantik

Der Philosoph Ludwig Wittgenstein hat in seinem posthum veröffentlichten Werk „Philosophische Untersuchungen“ den Begriff der Semantik (Bedeutungslehre) als Teilgebiet der Sprachwissenschaft (Linguistik) definiert. Laut Wittgenstein (1984) ist Linguistik die Wissenschaft, die sich mit Sinn und Bedeutung von Sprache, beziehungsweise sprachlichen Zeichen befasst.

Genau dieser Unterschied zwischen Sinn und Bedeutung ist entscheidend, können doch verschiedene Dinge dieselbe Bedeutung aufweisen, aber komplett unterschiedliche Dinge meinen. Beispielsweise haben die Begriffe „Morgenstern“ und „Abendstern“ die gleiche Bedeutung (die Venus), meinen aber verschiedene Dinge (siehe Pellegrini, 2006).

Die Definition und Auffassung der Begriffe Sinn und Bedeutung im Kontext der Semantik sind im Bereich der Semiotik, also der allgemeinen Lehre von den Zeichen essentiell, im Kontext der Informatik im Bereich des Semantic Web ist laut Pellegrini (2006) die Unterscheidung der drei Teilgebiete der Semiotik, „Syntax“, „Semantik“ und „Pragmatik“ bedeutender. Diese drei Begriffe bilden die Bausteine des so genannten semiotischen Dreiecks. Jedes Zeichen sei untrennbar mit dem Bezeichneten (Gegenstand) und dem Interpret (Referent) in einer triadischen Struktur verbunden.

Unter Syntax versteht man laut Pellegrini (2006) die Beziehung der Zeichen untereinander, während man unter Semantik die Beziehung der Zeichen zu Objekten bzw. Gegenständen der Außenwelt versteht. Mit Pragmatik ist wiederum die Beziehung zwischen den Zeichen und deren Interpreten und Kontexten gemeint.

Mit Hilfe dieses ersten Schrittes einer Definition des Semantikbegriffes fällt es nun leichter sich dem Begriff des Semantic Web anzunähern.

Der Begriff des Semantic Web

Berners-Lee et al. (2001) beschreibt im Artikel „The Semantic Web“ aus dem Jahr 2001 den Begriff Semantic Web folgendermaßen:

> The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperations.

Das Semantic Web soll nach Berners-Lee et al. (2001) also die Informationen des vorhandenen Internets mit „Bedeutung“ anreichern. Mit Semantik die auch von Maschinen verstanden werden soll.

Maschinen sollen demnach laut Pellegrini (2006) auf semantischer Ebene miteinander arbeiten können. Dies ist schon lange ein Thema im Bereich der künstlichen Intelligenz. Die Anreicherung von Dokumenten mit semantischer Information kann entweder händisch erfolgen, oder aber im Idealfall durch halb- oder vollautomatische Systeme. Die Entwicklung solcher automatischer Systeme zur Generierung von semantischer Information aus Dokumenten, bzw. der Informationsextraktion aus großen Datenmengen, wird im Bereich der Künstlichen Intelligenz und im Knowledge Engeneering schon länger behandelt. Diese Systeme sind ein wesentlicher Bestandteil der Entwicklung und Umsetzung der Idee eines Semantic Web.

Unter Berücksichtigung der im Abschnitt 2.1 angesprochenen sprachwissenschaftlichen Bedeutung von Semantik ist der Begriff Semantic Web allerdings an sich zu ungenau. Nach Sowa (2000) sollte man den Begriff Semantic Web ausweiten auf „Semiotic Web“, da es die Konzepte der Syntax, Semantik, und das der Pragmatik zu einem einzigen Konzept vereint.

Interoperabilität als Schlüssel

Eine Grundvoraussetzung für das Zustandekommen eines Semantic Web ist die Erreichung einer semantischen Interoperabilität, die nur über die Entwicklung von geeigneten Sprachen und Standards für den Austausch von semantischen Informationen erreicht werden kann. Zu diesem Zweck hat das W3C[8] 1997 die erste Spezifikation des „Resource-Description-Framework“ (RDF) vorgestellt. RDF ist eine wirkungvolle, einfach aufgebaute, auf Tripel basierende Repräsentationssprache für URIs (Universal Resource Identifieers). Das Konzept der URIs hat eine Schlüsselrolle im Kontext des Semantic Web, auf deren Bedeutung an späterer Stelle noch genauer eingegangen wird. Im Jahr 1999 wird RDF schließlich vom W3C offiziell als Standard aufgenommen (siehe Shadbolt et al., 2006). Das Resource Description Framework wird in Abschnitt 2.2 noch genauer behandelt.

Mit einer geglückten Anreicherung des konventionellen Webs mit maschinenlesbaren semantischen Daten in Form von RDF steht nun der Vernetzung dieser Daten untereinander nichts mehr im Wege. Ähnlich wie die Verbindung von einzelnen Dokumenten des auf HTML (Hyper Text Markup Language) basierenden Hyperwebs über Hyperlinks, können mittels RDF verschiedenste Datenstände zum so genannten „Web of Data“ verknüpft werden, das nach Bizer et al. (2009a) als

> ... a web of things in the world, described by data on the web ...

beschrieben werden kann, und allgemein als „Linked Data“ bekannt ist. (Siehe auch Abschnitt 2.6.)

Semantic Web vs Semantic Technologies

Laut Pellegrini (2006) versteht sich das Semantic Web als Erweiterung des bestehenden Internets und fußt im Wesentlichen auf der Anwendung von Standards zur Beschreibung

[8] `http://www.w3.org/`

von Prozessen, Dokumenten und Inhalten, sowie entsprechenden Metadaten. Diese Standards wurden vorwiegend vom W3C vorgeschlagen. Im Gegensatz dazu stehen Semantische Technologien als Mittel zur Lösung und Bewältigung komplexer Arbeitsprozesse, Informationsmengen- bzw. Retrievalprozessen und Vernetzungs- oder Integrationsaktivitäten, die nicht nur im Internet, sondern auch innerhalb von Organisationsgrenzen in Angriff genommen werden, bereit. Ein weiteres zentrales Anliegen des Semantic Web ist laut Pellegrini (2006) die maschinelle Verarbeitung von Information zur Automatisierung von Prozessschritten. Semantik, die sich als Mittel zum Transport jener Informationen, die es Maschinen erlauben zu „verstehen“ und zu „entscheiden“ versteht, verfolge andere Ziele als die Anreicherung von Information mit semantischen Inhalten.

Semantic Web Stack

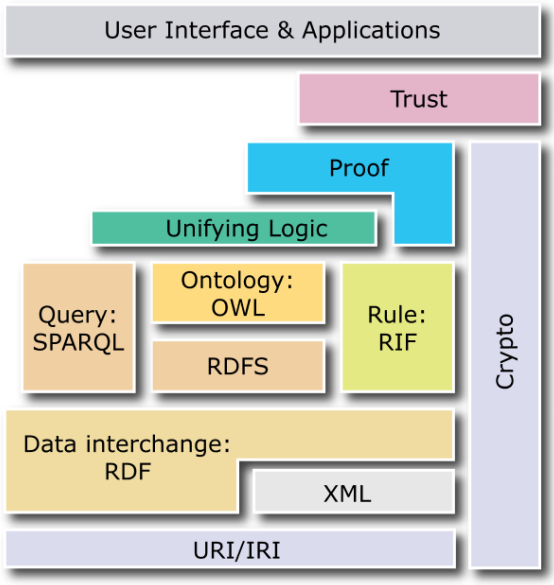

Abbildung 2.1: Semantic Web Stack (`http://www.w3.org/2007/03/layerCake.png`)

Das Semantic Web wird also aus einem Konglomerat standardisierter Sprachen/Technologien gebildet. Diese sind im „Semantic Web Stack„ in grafischer Form dargestellt (siehe Abbildung 2.1). Wie der Abbildung zu entnehmen ist, beschreibt der Semantic Web Stack ein Schichtenmodell und verläuft von unten nach oben. Die Technologie einer oberen Ebene baut auf den Technologien der unteren Ebenen auf, indem sie deren Features benutzt und erweitert, um neue Möglichkeiten für die oberen Ebenen zu generieren (siehe Horrocks et al., 2005). Auf unterster Stufe steht das Konzept der „Universal Resource Identifier“ (URI). Sie dienen als eindeutige Identifikation für jedes Element im Semantic Web und auf Ihnen baut jede weitere Stufe auf (siehe auch Abschnitt 2.2.1).

Die nächste Stufe beschreibt die verwendete Syntax und die Form des Datenaustausches. Hier wird einerseits die „Extensible Markup Language“ (XML) als Syntax, und das „Resource Description Framework“ (siehe Abschnitt 2.2) definiert. Das Vokabular für die semantische Beschreibungen wird über das „RDF Schema“ (siehe Abschnitt 2.3) bestimmt, und von der „Web Ontology Language“ (siehe Abschnitt 2.4.3) erweitert.

Das letzte im Kontext dieser Arbeit beschriebene Element des Semantic Web Stack ist SPARQL (SPARQL Protocol And RDF Query Language, siehe Abschnitt 2.5). SPARQL dient als Abfragesprache für die mit den vorher genannten Werkzeugen annotierten Daten.

2.2 Resource Description Framework (RDF)

RDF steht für „Resource Description Framework“ und wurde erstmals von Berners-Lee (1998) als Methode zur Anreicherung des Webs mit Metadaten beschrieben . Im Jahr 1999 wurde RDF vom W3C als Standard für Metadaten publiziert. RDF ist eine Sprache die darauf ausgelegt ist, Aussagen über Resourcen zu definieren. Unter Resourcen versteht man eine Entität, die über einen einzigartigen „Universal Resource Identifier“ oder URI identifiziert wird. Was als Entität bezeichnet werden kann ist laut Weiss (2009), ist sehr offen. Eine Entität kann eine Webpage, ein physikalisches Objekt (Ding), aber auch eine Person sein.

Mit RDF ist eine Beschreibung jeder erdenklichen Art von Resource und deren Beziehungen untereinander möglich. Am 10. Februar 2004 hat die „RDF Core Working Group“[9], eine Unterabteilung der „W3C Semantic Web Activity“[10], eine aktualisierte Fassung der „RDF W3C Recommendation“, bestehend aus 6 Dokumenten, herausgebracht, die je einen Teilaspekt von RDF beschreiben. In ihnen wird RDF wie folgt definiert:

- RDF ist eine Sprache mit der Aussagen über Resourcen des World Wide Web getätigt werden können. (Beschrieben im Dokument „RDF Primer “[11])
- RDF ist ein Framework das Informationen über das Web bereitstellt. (Beschrieben im Dokument „RDF Concept“[12])
- RDF ist eine universelle Sprache für die Repräsentation von Informationen im Web. (Beschrieben in den Dokumenten „RDF Syntax“[13] und „RDF Schema“[14]).
- Mit RDF kann man Aussagen mit Hilfe eines präzisen formalen Vokabulars, das über RDFS (siehe Abschnitt 2.3) spezifiziert ist, tätigen, die über das Internet verbreitet werden und als Basis für höhere Sprachen mit einem ähnlichem Ziel dienen. (Beschrieben im Dokument „RDF Semantics“)

[9] http://www.w3.org/2001/sw/RDFCore/
[10] http://www.w3.org/2001/sw
[11] http://www.w3.org/TR/rdf-primer/
[12] http://www.w3.org/TR/rdf-concepts/
[13] http://www.w3.org/TR/REC-rdf-syntax/
[14] http://www.w3.org/TR/rdf-schema/

Weiss (2009) beschreibt RDF als eine der Kernkomponenten des Semantic Web Stack (siehe Abbildung 2.1) und auch Yu (2011) bezeichnet RDF als „the building block for the Semantic Web“, da es die maschinenlesbare Annotation von Resourcen gleich welcher Art mit Metadaten erlaubt, die absolut nötig ist, um die benötigte Interoperabilität zwischen verschiedenen Applikationen zu ermöglichen, und somit die grundlegende Idee des Semantic Web zu erfüllen (siehe auch Abschnitt 2.1).

2.2.1 Das abstrakte Modell von RDF

Das Resource Description Framework (RDF) erzeugt Statements in Form von Subjekt, Prädikat und Objekt (s p o). Das Objekt kann dabei wie das Subjekt eine Resource darstellen, und selbst wieder in weiteren Statements als Subjekt benutzt werden (siehe Abbildung 2.2). Durch diese Relationen entsteht ein gerichteter Graph aus Resourcen und deren Eigenschaften, die durch URIs definiert, und mit dem Web verbunden sind (siehe Pellegrini, 2006).

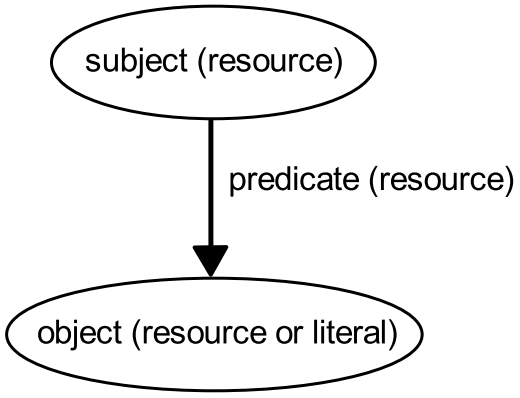

Abbildung 2.2: RDF Statement (`http://www.daml.org/2002/03/tutorial/statement.gif`)

Resourcen und URIs

Wie schon oben erwähnt, werden Resourcen in RDF über URIs definiert. Eine URI, wie sie zur Identifizierung von Subjekt, Prädikat und Objekt in einem RDF-Tripel verwendet wird, ist einmalig. Es wird nach Yu (2011) zwischen zwei verschiedenen Arten von URIs zur Identifizierung von Resourcen unterschieden. „Hash URIs“ und „Slash URIs“. Eine Slash URI entspricht der geläufigen Notation für Webadressen und Bedarf keiner weiteren Erklärung. Eine Hash URI hingegen, besteht aus einer normalen URI, gefolgt von einem „#“ und einem so genannten „Fragment Identifier“. Im Listing 2.1 ist ein Beispiel für dieselbe Resource, einmal als Hash URI und einmal in der Slash URI Notation, dargestellt (siehe Yu, 2011).

```
1 http://www.someurl.com/things/Thing_A
```

```
2 http://www.someurl.com/things#Thing_A
```

Listing 2.1: Hash und Slash URI

Mit dem Aufkommen des *Linked Data Projekts* im Jahre 2007 (siehe auch Abschnitt 2.6) ist nach Yu (2011) festgelegt, dass RDF URIs, wenn man sie in die Adressleiste eines Webbrowser eintippt, auch verlässlich zu einem für den Menschen lesbaren Ergebnis führen. An sich ist das keine Voraussetzung für eine gültige RDF-URI, da sie eigentlich nur zur eindeutigen Identifikation einer Resource dient. Aber im Kontext von Linked Data ist das Dereferenzieren von RDF-URIs eine nötige Voraussetzung. Dieser Umstand begünstigt die Hash URI Notation bei der Wahl des passenden URI-Schemas, da es einfacher ist sicherzustellen, dass eine Hash URI dereferenzierbar ist als eine Slash URI. Für letztere würde ein Negotiations-Mechanismus benötigt werden (siehe hierzu auch Abschnitt 2.6.3), was bei der Hash-Notation nicht der Fall ist.

Bisher wurden URIs nur mit Subjekten und Objekten in Verbindung gebracht, da sie die Repräsentation einer Resource bzw. einer Entität übernehmen. Nach Yu (2011) ist aber auch durchaus sinnvoll Prädikate in URI Notation zu verfassen. Einerseits aus dem gleichen Grund wie schon oben erwähnt: Eine URI ist einmalig.

```
1 http://www.someurl.com/camera#model
2 http://www.someurl.com/tv#model
```

Listing 2.2: Prädikate in URI Notation (siehe Yu, 2011)

In Listing 2.2 wird dies veranschaulicht. Das Prädikat „model“ wird einmal für die Resource „camera“ und einmal für die Resource „tv“ zur Angabe einer Modellnummer verwendet. Einmal im Kontext von TV-Geräten und einmal im Kontext von Kameras. Durch die Notation als URI ist eindeutig in welchem Kontext das jeweilige Prädikat verwendet wird, da allein über den Namen des Prädikats keine Unterscheidung zwischen den beiden Prädikaten getroffen werden kann. Eine Applikation würde die beiden Prädikate mit dem Namen „model“ als ident ansehen.

Ein weiterer Vorteil der URI-Notation bei Prädikaten liegt laut Yu (2011) in der Möglichkeit, dieses Prädikat als Subjekt einzusetzen und somit weitere Aussagen über dieses Prädikat machen zu können, und in der Möglichkeit der Entwicklung und Verwendung von „Shared-Vocabularies“, also von bereits existierenden Modellen zur Beschreibung von Resourcen (siehe dazu auch Abschnitt 2.3).

Blank-Nodes, Strings und Datentypen

Ein Objekt kann neben einer Resource auch einen literalen Wert (String) annehmen. Wird eine Resource nicht über eine URI identifiziert, bezeichnet man dieses Objekt im Kontext der Graphendarstellung als „Blank Node“. Ein möglicher Einsatz von Blank Nodes ist der Ausbau einer grundsätzlich binären Relation einer Aussage in eine n-äre Relation. Dies wird durch die Einführung einer weiteren Resource bewerkstelligt, die als Objekt der Basisaussage dient. Diese neue Resource bildet daraufhin wieder eine Reihe von Aussagen in der Rolle des Subjekts.

Diese als Hilfsmittel zum Aufbau einer n-ären Relation zur Basisresource verwendete Resource wird niemals von außerhalb des Graphen referenziert und benötigt deswegen auch keine eigene URI. Damit ist klar, dass die Bezeichnung Blank Node durchaus wörtlich zu nehmen ist. Abbildung 2.3 zeigt die Verwendung eines Blank Node zur Erreichung einer n-ären Relation in einer Graphendarstellung.

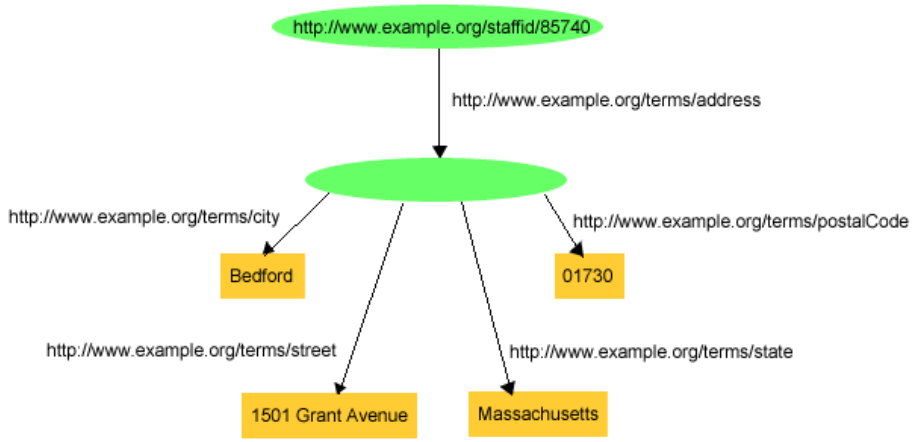

Abbildung 2.3: n-äre Relation über *Blank Nodes* (`http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/fig6may19`)

Die Resource mit der URI *http://www.example.org/staffid/85740* bildet ein Statement mit einem Prädikat mit der URI *http://www.example.org/terms/address* und einem Objekt. Dieses Objekt stellt zwar eine Resource dar, besitzt aber keine eindeutige URI. Mit dieser Resource werden daraufhin weitere Aussagen gebildet. Tabelle 2.1 übersetzt den Graphen aus Abbildung 2.3 in seine einzelnen Aussagen.

Subjekt	Prädikat	Objekt
http://www.example.org/staffid/85740	http://www.example.org/terms/address	BLANK_NODE
BLANK_NODE	http://www.example.org/terms/city	„Bedford“
BLANK_NODE	http://www.example.org/terms/street	„1501 Grant Avenue“
BLANK_NODE	http://www.example.org/terms/state	„Massachusetts“
BLANK_NODE	http://www.example.org/terms/postalCode	„011730“

Tabelle 2.1: Aussagen aus Abbildung 2.3

In den in Tabelle 2.1 dargestellten Aussagen werdem Strings als Objekte eingesetzt. Stringwerte bei Objekten können beispielsweise Werte für Prädikate wie „name“, „isbn number“ oder Ähnliches darstellen, und können über einen zusätzlichen „language tag“ lokalisiert werden. Daneben kann der String auch noch mit einer URI versehen werden, die Aufschluss über seinen Datentyp gibt. Listing 2.3 zeigt ein Beispiel für einen lokalisierten String mit *language tags* für die abgekürzte Form des akademischen Grades Doktor und einer Datentypdefinition.

```
1 "Dr."
2 "Dr."@de
```

```
3 "Dr." @en
4 "Dr."^^<http://www.w3.org/2001/XMLSchema#string>
```

Listing 2.3: Lokalisierung mit „language tags“ und Typdefinition

Der Wert in der ersten Zeile ist ein reiner Text ohne Informationen über die Sprache oder den Datentyp. In den Zeilen 2 und 3 ist der String mit je einer Lokalisierungsinformation versehen, womit die Zuordnung des Strings zu einer bestimmten Sprache möglich ist. In Zeile 4 wird eine Typinformation beigefügt, die den Wert eindeutig als „String“ ausweist. Der Vorteil der Typisierung liegt auf der Hand. Ein Literal mit einer Typdefinition kann nach Yu (2011) von einem Parser oder einer Applikation seinem Typ gerecht interpretiert, dargestellt und verarbeitet werden.

Zusammenfassung

Mit RDF ist es möglich eine beliebige Resource mit Hilfe von RDF-Tripel in der Form Subjekt, Prädikat und Objekt zu beschreiben, und somit eine Aussage über das Subjekt zu treffen. Mehrere Aussagen bilden einen gerichteten Graph. Subjekt, Prädikat und Objekt werden über URIs definiert. Ein Subjekt wird auch als „node“ oder „start node“, ein Objekt als „node“ oder „end node“ und ein Prädikat als „edge“ im Kontext des Graphen bezeichnet. Ein Objekt oder Prädikat kann ebenfalls wieder als Subjekt in einem anderen Statement verwendet werden, um weitere Informationen über die Resource bereitzustellen und den Graphen zu vernetzen.

2.2.2 Die RDF Syntax

Für die maschinelle Verarbeitung von RDF-Daten ist die oben besprochene Graphenrepräsentation von RDF nicht geeignet. Hierfür wird eine geeignete Serialisierungsform der RDF-Statements benötigt. Aus dem Semantic Web Stack aus Abbildung 2.1 ist ersichtlich, dass XML[15] die für RDF verwendete Syntax darstellt. Laut Yu (2011) ist und bleibt XML die dem Web zugrundeliegende Sprache und wurde auch vom W3C für die Integration von RDF-Daten spezifiziert.

Die vom W3C definierte und auf XML basierende Syntax für RDF nennt sich „RDF/XML“[16], ist aber laut Yu (2011) nicht die einzige Möglichkeit einer Serialisierung. Der Nachteil der XML-basierten Syntax sei die komplizierte Schreibweise und der XML-typische Overhead. Daher gibt es neben RDF/XML noch weitere Serialisierungsformen für RDF wie Notation 3[17], Turtle (Terse RDF Triple Language)[18] oder N-Triples[19].

Syntaxbeispiele

Das Listing 2.4 zeigt ein Beispiel für die RDF/XML Notation. Es beginnt mit einer Definition der verwendeten Namespaces bzw. RDF-Vokabulare (für mehr Informationen

[15] http://www.w3.org/XML/
[16] http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/
[17] http://www.w3.org/DesignIssues/Notation3.html
[18] http://www.w3.org/TeamSubmission/turtle/
[19] http://www.w3.org/TR/rdf-testcases/#ntriples

über Vokabulare siehe Abschnitt 2.3), gefolgt von einer Aussage markiert mit dem Tag „<rdf:Description/>“. Das Attribut „rdf:about“ definiert die Resource, also das Subjekt, über das die Aussage getroffen wird. Die nachfolgenden Zeilen 6 und 7 stellen Prädikat und Objektpaare dar. Jedes Prädikat wird als Tag notiert („dc:title“, „dc:publisher“).

```
1 <rdf:RDF
2     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3     xmlns:dc="http://purl.org/dc/elements/1.1/">
4   <rdf:Description
5         rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
6     <dc:title>Tony Benn</dc:title>
7     <dc:publisher>Wikipedia</dc:publisher>
8   </rdf:Description>
9 </rdf:RDF>
```

Listing 2.4: Notationsbeispiel RDF/XML (`http://en.wikipedia.org/wiki/Notation3`)

In Listing 2.5 wird der gleiche RDF-Datensatz in der N3-Notation dargestellt, eine weitere Serialisierungsform die nicht auf XML basiert und ihren Fokus auf eine einfache Produzierbarkeit und Lesbarkeit für den Menschen legt. Am Beginn steht wieder eine Namespace-Definition mit Einführung eines Präfix für das verwendete Vokabular. Danach folgt das Subjekt als Resource in spitzen Klammern, gefolgt von Prädikat, Objekt Paaren ohne spezielle Kennzeichnung.

```
1 @prefix dc: <http://purl.org/dc/elements/1.1/>.
2
3 <http://en.wikipedia.org/wiki/Tony_Benn>
4   dc:title "Tony Benn";
5   dc:publisher "Wikipedia".
```

Listing 2.5: Notationsbeispiel N-triple (`http://en.wikipedia.org/wiki/Notation3`)

Turtle ist eine vereinfachte Untermenge von Notation 3 und wird deswegen nicht extra erläutert. Die hier gegebenen Beispiele sind nur exemplarische Auszüge. Eine genaue Spezifikation der Syntax der einzelnen Notationsformen ist in dieser Arbeit nicht vorgesehen. Für genauere Informationen wird auf die Spezifikationen der Notationsformen verwiesen.

2.3 RDF Schema (RDFS)

Laut Yu (2011) lässt sich das RDF-Schema als Beschreibungssprache für RDF- Vokabularien verstehen. Mit RDF lassen sich einfache Aussagen über Resourcen mittels Eigenschaften und Werten in Form von Tripel der Form (Subjekt, Prädikat, Objekt) definieren. In diesen Aussagen oder Statements wird ein bestimmtes Vokabular zur Beschreibung von Resourcen einer bestimmten Domäne verwendet. Dieses Vokabular wird wiederum über das „Resource Description Framework Schema“ (RDFS) definiert. Mit der Verwendung eines bestimmten Vokabulars weist man darauf hin, dass die beschriebene Resource zu einer bestimmten Klasse von Resourcen hinzuzurechnen ist, und dementsprechende Eigenschaften aufweist.

Die Werkzeuge oder Sprachkonstrukte zur Definition dieser anwendungspezifischen Klassen und Eigenschaften werden vom RDF-Schema in Form eines Vokabulars bereitgestellt. Es bietet Möglichkeiten zur Definition von Zusammengehörigkeiten von bestimmten Klassen und Eigenschaften ähnlich einem Typsystem von Objektorientieren Programmiersprachen. Es können einzelne Resourcen als Instanzen einer oder mehrerer Klassen definiert werden und darüberhinaus können diese Klassen in einer hierarchischen Struktur aufgebaut sein. Eine bestimmte Klasse kann somit als Subklasse einer Elternklasse definiert werden und so fort (siehe Brickley and Guha, 2004). Die Definitionswerkzeuge zur Generierung eines eigenen RDF-Vokabulars sind in einem speziellen Vokabular untergebracht und besteht aus einer Sammlung vordefinierter RDF-Resourcen mit speziellen Bedeutungen. Die Resourcen in diesem RDF-Schema Vokabular werden über URIs mit dem Präfix „http://www.w3.org/2000/01/rdf-schema#“ definiert, der üblicherweise mit dem QName „rdfs:“ betitelt wird (siehe Manola and Miller, 2004).

Ein mittels RDFS erzeugtes RDF-Vokabular für eine spezifische Domäne kann nun laut Yu (2011) von jeder Applikation, die dieses Vokabular kennt, benutzt werden und somit Aussagen über Dokumente treffen, die mit RDF-Daten mit Bezug auf dieses Vokabular angereichert wurden. Die Anwendung „kennt“ über das Vokabular die Beziehungen der Resourcen und Eigenschaften der RDF-Statements, und hat somit ein erhöhtes „Wissen“ über das Dokument, und ist in der Lage Schlüsse daraus zu ziehen. Beispielsweise kann eine Anwendung auf Grund der hierarchischen Struktur der verwendeten Klassen unter anderem Resourcenrelationen der Form „is a“ erkennen . Antoniou et al. (2005) beschreibt die wichtigsten Statements des RDF-Schema wie folgt:

- Eine Klasse beschreibt eine Gruppe von Individuen die sich dieselben Eigenschaften teilen. (e ist Typ der Klasse c)

  Individual(e type(c))

- Das nächste elementare Statement beschreibt eine Unterordnungsrelation zwischen Klassen. Die Klasse c_i ist eine Subklasse der Klasse c_j.

  SubClassOf(c_i c_j)

- Eine Instanz steht mit anderen Instanzen über Eigenschaften in Beziehung.

  Individual(e_i value(p e_j)

- Eigenschaften werden über Domäne und Wertebereiche definiert.

  ObjectProperty(p domain(c_i) range(c_j))

- Eigenschaften können wie Klassen in einer hierarchischen Struktur aufgebaut sein.

  SubPropertyOf(o_1 : p_i o_2 : p_j)

RDFS bietet einige Möglichkeiten zur Abbildung einer Domäne in ein maschinenlesbares Vokabular. Allerdings ist es auch etwas eingeschränkt in seiner Ausdrucksmöglichkeit. Nach Antoniou et al. (2005) ist es beispielsweise nicht möglich Disjunktionen von Klassen zu definieren. Wenn man ausdrücken möchte, dass die Klassen „männlich“ und „weiblich“ disjunkt sind, ist das in RDFS, das mit hierarchischen Strukturen arbeitet, nicht möglich. Man kann die Klassen nur als Subklasse der Klasse „Person“ definieren. Auch ist es nicht möglich neue Klassen über Kombinationen von anderen Klassen zu erzeugen. Damit sind Operationen wie die Vereinigung, Schnittmenge oder das Komplement von verschiedenen Klassen. In RDFS ist es ebenfalls nicht möglich Beschränkungen bei der Anzahl möglicher Werte einer Eigenschaft zu definieren. Ein Beispiel wäre die Aussage, dass ein Auto maximal vier Räder aufweist. (Eine Minimalbeschränkung ist allerdings bei Instanzen durch den Einsatz von Blanknodes möglich. Siehe hierzu auch Abschnitt 2.2.1). Neben diesen Beispielen gibt es noch weitere Einschränkungen von RDFS die eine vollständige Abbildung einer Domäne erschweren. Die Lösung bietet die dem RDF Schema übergeordnete Ontologiesprache OWL. Siehe dazu Abschnitt 2.4.3.

2.4 Ontologien

Der Begriff Ontologie wird in verschiedensten Bereichen angewandt und dementsprechend unterschiedlich interpretiert. Laut Antoniou et al. (2005) haben Ontologien eine lange Tradition im Bereich der Künstlichen Intelligenz (KI) und die klassische Definition einer Ontologie aus diesem Bereich beschreibt eine Ontolgie als eine formale Spezfikation einer Konzeptualisierung, also eine abstrakte vereinfachte Sichtweise der Welt, die wir mit Hilfe einer Ontologie in einer Sprache mit einer formalen Semantik beschreiben wollen.

Im Bereich der Wissensrepresentation wird eine Ontologie als eine Beschreibung von Konzepten und Beziehungen in einer Domäne beschrieben und im Kontext des Semantic Web gilt nach Heflin (2004) die Definition des W3C aus dem Jahr 2004:

> An ontology formally defines a common set of terms that are used to describe and represent a domain ... An ontology defines the terms used to describe and represent an area of knowledge.

Eine wichtige Eigenschaft einer Ontologie ist nach Yu (2011) ihre Zugehörigkeit zu einer bestimmten Wissensdomäne. Sie beschreiben ein bestimmtes Wissensgebiet wie beispielsweise Medizin oder Fotographie und besteht aus Ausdrücken aus dieser Domäne und deren Beziehungen untereindander. Diese Ausdrücke werden als „Konzepte“ oder „Klassen“ bezeichnet. Die Beziehungen dieser Klassen oder Konzepte verstehen sich als hierarchische Struktur, ähnlich dem Konzept der Vererbung in objektorientierten Programmiersprachen. Die Superklasse A beschreibt ein Konzept auf einer höheren Abstraktionsebene mit allgemeinen Eigenschaften, während seine Subklassen B und C konkretere Beschreibungen des Konzepts mit detaillierteren Eigenschaften darstellen. Die Subklassen erben dabei alle Eigenschaften der Superklasse. Neben der hierarchischen Beziehung zwischen Klassen oder Konzepten gibt es noch Beziehungen auf Grund von Eigenschaften. Eigenschaften beschreiben die verschiedenen Attribute einer Klasse und können ebenso verwendet werden um verschiedene Konzepte miteinander zu verbinden.

Auf Grund der Beschreibung einer Domäne mittels Konzepten und deren Beziehungen zueinander ist nach Yu (2011) eine Maschine mit Hilfe einer Ontologie in der Lage diese Domäne zu verstehen.

2.4.1 KOS

Im Bereich der Wissensorganisation gibt es Klassifikationsschemen die dem der Ontology sehr nahe kommen. So genannte „Knowledge Organisation Systems“ (KOS) verwenden Organisationsformen wie beispielsweise Taxonomien oder Thesauri, um Konzepte zu organisieren, und damit Aussagen über diese Konzepte treffen zu können. Dies beschreibt auch einen großen Unterschied zwischen einem KOS-Modell und einer Ontologie. Laut Yu (2011) dienen KOS-Modelle zur Organisation von Wissen, während Ontologien der Repräsentation von Wissen dienen.

KOS Modelle stellen laut Sowa (2000) „Lexikalische Ontologien“ dar, also Ontologien, deren Konzepte und Relationen nicht vollständig spezifiziert sind. Konzepte können nur über Beziehungen wie Über- und Unterklassen in Relation zu anderen Konzepten gebracht werden, können aber nicht komplett definiert werden. Im Gegensatz dazu stehen laut Sowa (2000) „Formale Ontologien“, deren Konzepte und Beziehungen mit Axiomen und Definitionen in einer automatisiert in Logik übersetzbaren Sprache spezifiziert sind. Der Unterschied der beiden Typen ist vor allem ein Unterschied den Grad ihrer Komplexität betreffend, und ihrer Unterstützung von automatisierten Schlussfolgerungen. Eine formale Ontologie unterstützt komplexere Schlussfolgerungen wie eine lexikalische Ontologie.

Schmitz-Esser and Sigel (2006) beschreiben die verschiedenen Stufen zu einer Ontologie ausgehend von Modellen mit sehr geringer semantischer Aussagekraft, bis hin zu logiktheoretische Modellen mit sehr hoher Semantik. Der Fokus der Kategorisierung von Schmitz-Esser and Sigel (2006) liegt in der semantischen Interoperabilität (siehe dazu auch Abschnitt 2.1).

Nachfolgend werden einige Modelle zur Wissensorganisation, gereiht nach ihrem semantischen Informationsgehalt, kurz charakterisiert.

Controlled Vocabularies, Glossare

Lacasta et al. (2010) beschreibt ein „Controlled Vocabulary“ als einfachste Organsiationsform von Wissen. Es besteht aus einer endlichen Liste von Begriffen über ein bestimmtes Thema. Ein einfaches Beispiel für ein Controlled Vocabulary ist beispielsweise die Einteilung einer Bücherei nach bestimmten Themenbereichen. Jeder Themenbereich ist mit einem speziellen Code ausgezeichnet, mit dem jedes Buch klassifiziert wird.

Ein Glossar ist definiert sich ebenfalls über eine Liste von Begriffen denen Definitionen oder Querverweise in natürlicher Sprache begefügt wurden.

Taxonomien

Taxonomien, Kategorisierungsschemen oder Klassifikationen gehen einen Schritt weiter. Sie sind nicht mehr nur reine Listen von Begriffen, sondern sind in einer hierarchischen Struktur aufgebaut. Spezifischere Unterbegriffe leiten von einem allgemeineren Oberbegriff ab. Sie dienen zur Klassifizierung von Resourcen und können nach Lacasta et al. (2010) auch polyhierarchisch aufgebaut sein.

Eine Variante der Taxonomie ist die Folksonomie. Im Gegensatz zur Taxonomie, die in der Regel von Experten auf dem jeweiligen Gebiet definiert wurde, wird eine Folksonomy von Leuten ohne Fachwissen, meist im Kontext von sozialen Webseiten aufgebaut.

Thesauri

Ein Thesaurus gehört nach Lacasta et al. (2010) zur Klasse der formalen Ontologien. Der Hauptzweck eines Thesaurus besteht in der Bereitstellung eines kontrollierten Vokabulars für die Indexierung, und der Unterstützung des Users bei der Suche nach bestimmten Ausdrücken. Ein Thesaurus hat eine hierarchische Struktur und ist so strukturiert, dass Beziehungen zwischen einzelnen Konzepten klar ersichtlich sind (zB. Synonyme, allgemeinere oder speziellere Begriffe). Diese Beziehungen sind in verschiedenen Standards eindeutig definiert und können von Maschinen verarbeitet werden. Notationsbeispiele für die Definition von Beziehungen in Thesauri sind nach Lacasta et al. (2010) „BT“ für allgemeiner (broader term), „NT“ für spezieller (narrow term), “SYN“ für Synonym oder „RT“ für die Anzeige von Assoziationen (association or relatedness).

Semantische Netze

Laut Lacasta et al. (2010) kann man ein Semantisches Netzwerk als eine Verallgemeinerung eines Thesaurus verstehen. Es bietet wie der Thesaurus eine Struktur mit Beziehungen zwischen Konzepten und Ausdrücken. Doch anders als beim Thesaurus ist diese Struktur nicht strikt hierarchisch aufgebaut, sondern entspricht der eines Netzwerkes mit weitergefassten und genauer definierten Beziehungen als die im Thesaurus verwendeten Standardbziehungen „BT“, „NT“ oder „RT“.

Knowledge Organisations Systmens (KOS) wie die oben beschriebenen Modelle der Taxonomien oder Thesauri sind besonders in folgenden Bereichen hilfreich:

- **Suche:** Neben der Suche nach Schlüsselwörtern können auch andere Kriterien in das Ergebnis einfließen. So können zum Beispiel dem Suchbegriff verwandte Begriffe in die Suche einbezogen werden.
- **Erstellung von intelligenten Suchinterfaces:** Einbeziehung der hierarchischen Struktur.
- **Organisation und Wiederverwendung von Wissen einer bestimmten Domäne**

Darüber hinaus fördert nach Yu (2011) der Einsatz von Wissensorganisationssystemen die Dateninteroperabilität.

2.4.2 SKOS als Brücke zur Ontologie

Im Vergleich zu einer Ontologie ist nach Yu (2011) die Beschreibung einer Domäne mit KOS sehr beschränkt. Mit Hilfe einer Taxonomie können nur Aussagen über die hierarchische Struktur mittels „BT“ und „NT“ Relationen getroffen werden. Ein Thesaurus ergänzt diese Relationen, ist aber auch sehr limitiert in seinen Möglichkeiten. KOS ist also wegen seiner geringen semantischen Information nur sehr eingeschränkt zur Repräsentation von Wissen geeingnet, da keine formale Schlussfolgerungen aus KOS gezogen werden kann.

Ontologien basieren auf Beschreibungslogik, weswegen aus ihnen logische Rückschlüsse gebildet werden können. Die Beziehungen zwischen Konzepten in KOS beinhalten nur wenig semantische Information. Beispielsweise können Ontologien eine „is-a“ Beziehung definieren, die in einem Thesaurus durch die hierarchische Struktur gegeben ist. Das Problem dabei ist aber, dass diese herarchische Beziehung verschiedene Bedeutungen haben kann. Sie kann genauso eine „is-a“ Beziehung wie auch eine „part-of“ Beziehung darstellen. Wie die Beziehung verstanden wird hängt von der Interpretation der Applikation und der Domäne ab.

Trotzdem ist es manchmal nötig bestehende Wissensorganisationssysteme in das Semantic Web zu integrieren. Diese Aufgabe erfüllt ein spezielles vom W3C definiertes und 2009 zum Standard erklärtes RDF Vokabular mit dem Namen SKOS (simple knowledge organization systems)[20] (siehe Yu, 2011).

SKOS ist ein Vokabular das speziell für die Überleitung von Wissensorgansisationssystemen wie Thesauri oder Taxonomien in den Bereich des Semantic Web definiert wurde. Die Systeme sind somit im Web publiziert und durch ihre Notation in RDF für Maschinen verarbeitbar und kann damit von verschiedensten Applikationen verwendet werden. Das bedeutet für ein in das Semantic Web integrierte Wissensorganisationssystem, dass jedes seiner Konzepte über eine URI identifiziert wird und über RDF-Statements ausgedrückt wird. Die Identifizierung der KOS Konzepte über URIs macht es möglich, Konzepte von verschiedenen Wissensorganisationssystemen semantisch zu verbinden, und so beispielsweise in einen Suchprozess einzubinden. Diese Möglichkeit der semantischen Verlinkung stellt nach Yu (2011) einen der größten Vorteile einer Übertragung eines KOS in das Semantic Web dar.

2.4.3 Web Ontology Language (OWL)

OWL steht für „Web Ontology Language “ und wird von Yu (2011) folgendermaßen definiert:

$$\text{OWL} = \text{RDF Schema} + \text{new constructs for better expressivness}$$

In Abschnitt 2.3 wurde RDFS als Mittel zur Wahl zur Erstellung von Ontologien vorgestellt, sodass Maschinen oder Applikationen, denen die Ontologie bekannt ist, Schlüsse aus Daten, die mit dieser Ontology annotiert wurden, ziehen können. Allerdings hat das RDFS noch einige Schwächen. Beispielsweise ist es nicht möglich zwei unterschiedliche Klassen zu definieren die das gleiche reale Konzept repräsentieren. Es ist auch nicht möglich eine Angabe über die zulässige Anzahl von Definitionen einer Eigenschaft zu machen. Ohne diese Einschränkungen lassen sich Eigenschaften, die im Bezug auf die Domäne sinnvollerweise

[20] `http://www.w3.org/2004/02/skos`

nur einmal pro Instanz vorkommen sollten, über RDF beliebig oft ein und derselben Instanz zuordnen. Es wird also eine Erweiterung des RDF-Schema benötigt, um diese und andere Tasks in einer Ontologie zu definieren. Diese Erweiterung sollte unter anderen folgende Eigenschaften erfüllen (siehe Yu, 2011):

- Sie sollte in der Lage sein Relationen von Klassen aus verschiedenen Dokumenten herzustellen,
- neue Klassen über die Vereinigung, Schnittmenge und Komplement mit einer anderen Klasse erzeugen,
- Einschränkungen über die Anzahl und den Typ von Eigenschaften zu definieren,
- festzulegen ob alle Zugehörige einer Klasse eine bestimmte Eigenschaften aufweisen müssen oder können.

Diese und mehr Eigenschaften werden laut Lacasta et al. (2010)in der Sprache OWL vereint. Sie stellt einen weiteren Baustein des Semantic Web Stack aus Abbildung 2.1 dar, und versteht sich im Kontext des Semantic Web als Standard für die Erstellung von formalen Ontologien.

OWL wurde im Feber 2004 als W3C Recommendation mit einer Reihe von Dokumenten zu seiner Spezifikation herausgebracht[21]. Damit war die Entwicklung von OWL aber noch nicht abgeschlossen. Mit dem Einsatz von OWL für die Entwicklung von Ontologien in den verschiedensten Wissensdomänen kamen weitere Einschränkungen der Sprache zum Vorschein. Mit der Einwirkung verschiedener Arbeitsgruppen wie der „W3C OWL Working Group“[22], die sich mit dem Thema befasst hatten, wurde im Oktober 2009 OWL2 offizieller W3C Standard[23]. Auf eine Beschreibung der Spezifikation von OWL wird an dieser Stelle mit einem Verweis auf die W3C Recommendation[24] verzichtet.

2.5 SPARQL

In Abschnitt 2.2 wurde das Resource Description Framework (RDF) als Datenmodell für die Repräsentation von Informationen über Daten aus dem Internet eingeführt. Nach Pérez et al. (2006) bestand mit der ersten W3C Recommendation für RDF im Jahre 1998 der Bedarf nach einer geeigneten Abfragesprache für RDF-Daten. Im Jahr 2004 hat die *RDF Data Access Working Group* den ersten Entwurf einer Abfragesprache für RDF mit dem Namen SPARQL vorgestellt.

Nach Yu (2011) vereint SPARQL eine Abfragesprache und Protokoll für RDF-Daten. Der Name SPARQL ist ein rekursives Akronym und steht für „SPARQL Protocol and RDF Query Language“, die 2008 schließlich als offizieller W3C Standard von der nun in „SPARQL

[21] `http://www.w3.org/2004/OWL/\#specs`
[22] `http://www.w3.org/2007/OWL/wiki/OWL_Working_Grop`
[23] `http://www.w3.org/TR/2009/REC-owl2-overview-20091027`
[24] `http://www.w3.org/TR/owl2-overview/`

Working Group“[25] unbenannten Arbeitsgruppe herausgegeben wurde. Die W3C Recommendation besteht aus drei seperaten Spezifikationen. Die Sprache an sich wird in der „SPARQL Query Language specification“[26] beschrieben, während das Dokument „SPARQL Query XML Results Format specification“[27] ein XML-Format für die Serialisierung der Ergebnisse einer SPARQL Query definiert. Das dritte Dokument „SPARQL Protocol for RDF specification“[28] definiert einfache HTTP und SOAP Protokolle für Remoteabfragen an RDF-Datenbanken. Mit dieser Spezifikation ist SPARQL bestens die Anforderungen an eine Abfragesprache für RDF-Daten gerüstet. Mit SPARQL ist nun unter anderem möglich:

- RDF-Graphen nach spezifischen Informationen abzufragen;
- Remoteabfragen an entfernte RDF-Server zu senden und Resultate in geeigneter Form zurückzuerhalten;
- automatisierte Abfragen für die Generierung von Reports zu erzeugen;
- das Senden und Verarbeiten von Abfrage und Resultaten über Applikationen zu ermöglichen.

Laut Hartig (2012) definiert SPARQL Abfragen als Funktionen über ein RDF-Dataset, also eine vorher definierte Sammlung von RDF-Tripeln (siehe auch Abschnitt 2.5.1). Ist dieses Dataset nach den in Abschnitt 2.6.1 definierten Linked-Data Prinzipien publiziert worden, ist es mit SPARQL möglich, neben absolut aktuellen Daten, zusätzliche vorher nicht bekannte Informationen zu entdecken.

Auf die Sprachelemente von SPARQL und deren Verwendung wird in Abschnitt 2.5.2 eingegangen. Zuvor soll noch ein Überblick über die Adressaten einer SPARQL-Abfrage gegeben werden, den „RDF Data Stores“ oder „Triple Stores“.

2.5.1 RDF Data Store

Ein RDF Data Store ist laut Yu (2011) eine spezielles, für die Verarbeitung und Speicherung von RDF-Daten konzipiertes Datenbanksystem, das Schnittstellen für das Hinzufügen, Abfragen und Löschen von RDF-Tripel bereitstellt. Natürlich unterstützen viele Data Stores neben diesen Grundfunktionalitäten noch weitere Features wie das Laden von RDF-Tripel über verschiedene Sourcen. Beispiele für populäre Data-Store Lösungen sind laut Morsey et al. (2011) Virtuoso[29], Sesame[30], Apache Jena-TDB[31] oder BigOWLIM[32]. Nach einem Benchmark-Test von Morsey et al. (2011) liegt Virtuoso als performantester Data Store an an vorderster Stelle, gefolgt von BigOWLM, Sesame und Jena-TDB.

[25] http://www.w3.org/2009/sparql/wiki/Main_Page
[26] http://www.w3.org/TR/rdf-sparql-query
[27] http://www.w3.org/TR/rdf-sparql-XMLres
[28] http://www.w3.org/TR/rdf-sparql-protocol
[29] http://virtuoso.openlinksw.com
[30] http://www.openrdf.org
[31] http://jena.apache.org/documentation/tdb/index.html
[32] http://www.ontotext.com/owlim/editions

SPARQL-Endpoints

Ein SPARQL Endpoint ist ein Interface, das von Usern und Applikationen gleichermaßen zum Absetzen von RDF-Queries verwendet werden kann. Abbildung 2.4 zeigt den SPARQL-Endpoint von DBpedia[33]. (Auf DBpedia wird in Abschnitt 2.7 genauer eingegangen.)

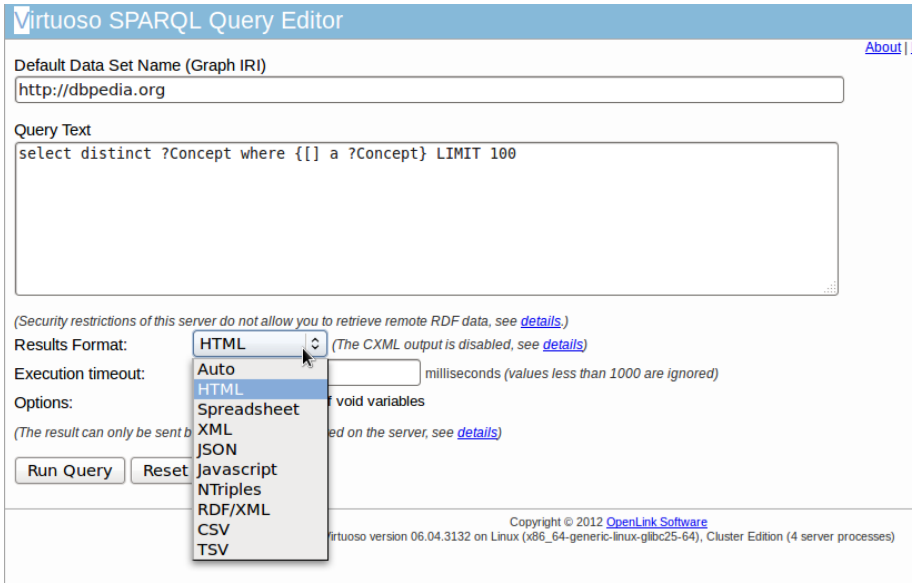

Abbildung 2.4: Virtuoso SPARQL Endpoint von Dbpedia (`http://dbpedia.org/sparql`)

Der Endpoint dient einzig und allein dazu, Queries im SPARQL-Format entgegenzunehmen und die Resultate entsprechend zurückzuliefern. Menschliche Nutzer verwenden für das Absetzen der Query entweder eine Weboberfläche wie sie in Abbildung 2.4 am Beispiel von Virtuoso gezeigt wird, oder verwenden eigene Anwendungen, die Anfragen über eine zur Verfügung gestellte API kommunizieren. Der SPARQL-Endpoint ist mit Parametern konfigurierbar. So können verschiedene Rückgabeformate wie zum Beispiel eine HTML-Tabelle (für menschliche User), oder aber auch serialisierte maschinenverarbeitbare Formate wie RDF/XML oder Turtle gewählt, oder ein Timeout gesetzt werden. (Für Informationen über die verschiedenen RDF Datenformate siehe Abschnitt 2.2.2).

2.5.2 Abfragesprache

Die Abfragsprache SPARQL hat nach Yu (2011) vier Hauptabfragetypen. SELECT, ASK, DESCRIBE und CONSTRUCT Abfragen. Alle diese Abfragen basieren auf zwei grundlegenden SPARQL-Konzepten:

- Tripel Pattern
- Graph Pattern

[33] `http://dbpedia.org/`

Wie in Abschnitt 2.2 erläutert, basierd das RDF Model auf dem Tripel-Konzept. Ein Tripel ist eine 3-Tupel Struktur mit der Form (Subjekt, Prädikat, Objekt). SPARQL basiert auf der gleichen Tripelstruktur.

Tripel Pattern

Laut Yu (2011) besteht der Unterschied gegenüber einem RDF Tripel darin, dass ein SPARQL-Tripel auch Variablen beinhalten kann. Variablen werden in SPARQL mit einem vorangestellten „?“ oder einem „$“ angezeigt.

Eine Variable kann man als Platzhalter für einen beliebigen Wert verstehen. Das Listing 2.6 gibt ein Beispiel eines vollständigen SPARQL Tripel in Turtle-Syntax. Zeile eins definiert den Präfix „foaf:“ mit dem Namespace von „Friend-of-a-Friend“ (FOAF)[34], einer Ontologie mit einem Vokabular speziell für die Beschreibung von Personen. Mit diesem Präfix können nun die in der FOAF-Ontologie spezifizierten Ausdrücke in den folgenden Statements verwendet werden. Die zweite Zeile entspricht nun dem SPARQL Tripel Pattern. Das Subjekt ist die URI der Resource „Dan Brickley“, das Prädikat is die FOAF Eigenschaft „foaf:name“ und das Objekt wird als Variable „?name“ definiert.

```
1 @prefix foaf: <http://xmlns.com/foaf/0.1/>.
2 <http://danbri.org/foaf.rdf#danbri> foaf:name ?name.
```

Listing 2.6: SPARQL-Tripel Yu (2011)

Bei einer Abfrage des Tripel Pattern aus Listing 2.6 an einen SPARQL-Endpoint werden laut Yu (2011) vereinfacht die folgende Schritte ausgeführt:

1. Im ersten Schritt wird ein neues RDF-Dokument mit dem Namen „resultSet“ erzeugt.
2. Im zweiten Schritt wird das nächste Tripel aus dem RDF-Graph geholt. Ist kein Tripel mehr vorhanden wird das Ergebnisdokument „resultSet“ zurückgegeben.
3. Das gefundenen RDF-Tripel aus dem Graph wird nun mit dem gegebenen Pattern verglichen. Wenn das Subjekt und das Prädikat aus dem RDFTripel des Graphen mit dem Subjekt und dem Prädikat des SPARQL Tripel übereinstimmen, wird der Wert des Objekts des RDF-Tripel des Graphen auf die Variable „?name“ gebunden, was in diesem Fall der Erzeugung eines neuen Tripels mit den übereinstimmenden Subjekt und Prädikat, sowie dem Wert des Objektes gleichkommt. Dieses neue Tripel wird dem „resultSet“hinzugefügt.
4. Beginne wieder mit Schritt 2.

Mit diesem Prozedere werden alle möglichen Bindings für die Variable „?name“ in das „resultSet“ inkludiert. Ist die Eigenschaft „foaf:name“ für die gegebene Resource mehrfach vergeben, werden auch alle Werte zurückgeliefert. Es kann auch mehr als eine Variable in einem Tripel vorhanden sein. Beispielsweise ist ein Tripel der Form

```
1 <http://danbri.org/foaf.rdf#danbri> ?predicate ?value
```

Listing 2.7: SPARQL-Tripel Yu (2011)

[34] http://www.foaf-project.org/

genauso zulässig wie das folgende Tripel:

```
1 ?subject ?predicate ?value
```

Listing 2.8: SPARQL-Tripel Yu (2011)

Das Tripel aus Listing 2.7 würde auf alle in diesem RDF-Graph definierten Eigenschaften und deren Werte für die Resource <http://danbri.org/foaf.rdf#danbri> matchen, während das Tripel aus Listing 2.8 jedes einzelne Tripel des RDF-Graphen matchen würde (siehe Yu, 2011).

Graph Pattern

Ein Graph Pattern dient nach Yu (2011), ähnlich wie ein Tripel Pattern, zur Selektion von RDF-Statements aus einem RDF-Graphen. Mit Hilfe eines Graph Pattern lassen sich allerdings komplexere Auswahlregeln definieren als mit einem einfachen Tripel Pattern. Ein Graph Pattern besteht aus einer Kombination von mehreren Tripel Pattern, die mit geschwungenen Klammern zusammengefasst werden (siehe Listing 2.9).

```
1 {
2   ?who foaf:name ?name.
3   ?who foaf:interest ?interest.
4   ?who foaf:knows ?others.
5 }
```

Listing 2.9: Beispiel für ein Graph Pattern

Wie in Listing 2.9 ersichtlich, scheint die Variable „?who“ in mehreren Tripeln des Graph Pattern auf. Der Wert für diese Variable muss in allen Tripeln in denen Sie vorkommt ident sein, damit die Resource aus dem RDF-Graph, an den die Abfrage gerichtet ist, in das Ergebnis aufgenommen werden kann.

Die Prozedur zum Matchen eines Graph Pattern erfolgt laut Yu (2011) in Analgoie zum Matchvorgang eines einzelnen Tripel Pattern (siehe Abschnitt 2.5.2) in folgenden Schritten:

1. Erzeuge ein neues Set mit dem Namen „resultSet“.
2. Hole die nächste Resource aus dem gegebenen RDF-Graph. Ist keine Resource mehr vorhanden, gebe das „resultSet“ zurück und beende den Matchingprozess.
3. Verarbeitung des ersten Tripel:
   - Wenn die aktuelle Resource keine Eigenschaft „foaf:name“ aufweist, gehe weiter zu Schritt 7.
   - Wenn die aktuelle Resource die Eigenschaft „faof:name“ besitzt, binde die Resource auf die Variable „?who“ und den Wert der Eigenschaft „foaf:name“ auf die Variable „?name“
4. Verarbeitung des zweiten Tripel:
   - Wenn die aktuelle Resource, die nun über die Variable „?who“ repräsentiert wird, keine Eigenschaft „foaf:interest“ aufweist, gehe zu Schritt 7.

- Wenn die aktuelle Resource eine Eigenscahft „foaf:interest“ besitzt, binde den Wert der Eigenschaft „foaf:interest“ auf die Variable „?interest“.

5. Verarbeitung des dritten Tripel:
   - Wenn die aktuelle Resource, die über die Variable „?who“ repräsentiert wird keine Eigenschaft „foaf:knows“ aufweist, gehe zu Schritt 7.
   - Wenn die aktuelle Resource die Eigenschaft „foaf:knows“ besitzt binde den Wert der Eigenschaft „foaf:knows“ auf die Variable „?others“.
6. Füge die aktuelle Resource dem resultSet hinzu.
7. Gehe zu Schritt 2.

Durch diesen Vorgang wird deutlich, dass bei einem Graph Pattern versucht wird, alle Resourcen zu finden, die die in Listing 2.9 definierten Eigenschaften aufweisen.

Die nachfolgenden Ausführungen über die Konstruktion einer SPARQL-Abfrage sollen einen Überblick über die Möglichkeiten von SPARQL und dessen Erweiterung SPARQL 1.1 im Bezug auf die am Anfang dieses Kapitels genannten Hauptformen einer SPARQL-Abfrage aufzeigen. Eine detailierte Beschreibung sämtlicher Funktionen der Abfragesprache SPARQL ist der W3C Recommendation von SPARQL[35], sowie der Recommendation von SPARQL 1.1[36] zu entnehmen.

SELECT-Anweisung

Laut Yu (2011) ist die SELECT-Abfrage die am meisten genutzte Abfrageform in Sparql. Der Grundlegende Aufbau einer SELECT-Abfrage wird in Listing 2.10 dargestellt.

Es folgt eine Beschreibung der einzelnen Abschnitte einer SELECT-Anweisung wie in Listing 2.10.

```
1  # base directive
2  BASE <URI>
3
4  #list of prefixes
5  PREFIX pref: <URI>
6  ...
7
8  # result description
9  SELECT...
10
11 # graph to search
12 FROM ...
13
14 # query pattern
```

[35] http://www.w3.org/TR/rdf-sparql-query/
[36] http://www.w3.org/TR/sparql11-query/

```
15 WHERE {
16    ...
17 }
18
19 # query modifiers
20 ORDER BY ...
```

Listing 2.10: Struktur einer SELECT-Abfrage in SPARQL (siehe Yu, 2011)

BASE: Am Beginn der Abfrage steht die BASE-Direktive und eine Liste von verwendeten PREFIX-Definitionen, bestehend aus aus einer gewissen Anzahl von PREFIX-Statements. Diese beiden Teile sind optional und werden für die Abkürzung von URIs verwendet, indem sie eine Art Label für eine gegebene URI definieren. Dieses Label kann in weiterer Folge innerhalb der Abfrage als Repräsentant der URI eingesetzt werden.

SELECT: Der Nächste Abschnitt beinhaltet die SELECT-Anweisung. Sie definiert welche Daten oder Variablen von der Abfrage zurückgegeben werden sollen. Manchmal ist es sinnvoll die Ergebnisliste der Abfrage abzuändern. Hierfür sind spezielle SPARQL-Funktionen verfügbar. So bereinigt das Keyword DISTINCT das Ergebnis von doppelten Einträgen. Weitere Schlüsselwörter zur Reorganisation der Ergebnistabelle können am Ende der Query angegeben werden.

FROM: Die FROM-Anweisung dient dazu dem SPARQL-Enpoint mitzuteilen mit Welchem Graph die Suche durchzuführen ist. Diese Anweisung ist ebenfalls optional. Der mittels FROM definierte RDF-Graph wird als „background graph“ bezeichnet.

WHERE: Im WHERE-Abschnitt der Abfrage befindet sich das Graph-Pattern, das die gewünschten Ergebnisse spezifiziert (siehe dazu auch Abschnitt 2.5.2). Die WHERE-Anweisung ist nicht optional, das Schlüsselwort an sich kann aber weggelassen werden. Hier können auch weitere Funktionalitäten wie das Keyword OPTIONAL eingesetzt werden. Graph Pattern, wie in Listing 2.9 aufgeführt, sind als UND-Verknüpfungen anzusehen. Jedes einzelne Tripel-Pattern muss erfüllt sein, damit die Resource in die Ergebnismenge aufgenommen wird. Da es in RDF aber durchaus vorkommen kann, dass zwei Instanzen der gleichen Klasse unterschiedliche Eigenschaften aufweisen, ist es nicht sicher ob ein im Pattern gefordertes Prädikat vorhanden ist. Dies würde einen sofortigen Ausschluss der Resource aus der Ergebnismenge bedeuten. Mit der Angabe des Schlüsselworts OPTIONAL wird das zugehörige Pattern optional, muss also nicht zwangsläufig von jeder Resource erfüllt werden. Der Gesamterfolg der Query hängt also nicht mehr von diesem einen, mit OPTIONAL annotierten, Pattern ab.

Neben dem Fall eines optionalen Match kann auch der Bedarf nach einem alterativen Match gegeben sein. Hierfür stellt SPARQL das Keyword UNION zur Verfügung. Es wird ähnlich wie das Schlüsselwort OPTIONAL angewandt. Mit dem Schlüsselwort UNION ist es möglich, verschiedene Pattern als Alternativen anzubieten. Es muss aber zumindest eines der mit UNION verknüpften Pattern auf die Resource zutreffen, damit sie der Ergebnismenge zugeordnet werden kann. Treffen mehr als eine Alternative zu, werden alle zutreffenden Pattern der Ergebnismenge angefügt.

SPARQL bietet die Möglichkeit neben dem definierten „background graph“ weitere RDF-Graphen als so genannte „named graphs“ in eine Query miteinzubeziehen. Hierzu müssen diese Graphen in folgender Form definiert werden:

```
1 from named <uri>
```

In obiger Definition bezeichnet „<uri>“ die Adresse des RDF-Graphen der hinzugefügt werden soll. Nun kann der Graph innerhalb der Query mit dem Schlüsselwort GRAPH verwendet werden.

Der WHERE-Abschnitt kann noch um weitere Angaben erweitert werden. So können mit dem Schlüsselwort FILTER Einschränkungen bezüglich der Wertebereiche von Eigenschaften getroffen werden. Diese Einschränkungen werden als logische Ausdrücke definiert, die zu boolschen Werten evaluieren, womit sie mit den logischen Operatoren „&&“ und „||“ verknüpfbar sind. Die Filteraussage muss auf „True“ evaluieren damit die Resource der Resultatmenge hinzugefügt wird. Tabelle 2.2 gibt eine Übersicht über die unterstützten Filterfunktionen und Operatoren. Für eine vollständige Auflistung und Beschreibung aller Operatoren und deren Verhalten bei unterschiedlichen Datentypen wird auf die „SPARQL Query Language for RDF Recommendation“[37] verwiesen.

Kategorie	Funktionen und Operatoren
Logik	!, &&,||
Mathematik	+, -, *, /
Vergleiche	=, !=, >, <
SPARQL Testers	isURL(), isBlanc(), isLiteral(), bound()
SPARQL Accessors	str(), lang(), datatype()
Andere	sameTerm(), langMatches(), regex()

Tabelle 2.2: Operatoren und Funktionen in SPARQL (`http://www.w3.org/TR/rdf-sparql-query/#OperatorMapping`)

MODIFIER: Im letzten Teil der Query werden dem SPARQL-Endpoint Angaben zur Organisation des Resultats gemacht. Hier finden sich Anweisungen zur Sortierung oder Limitierung der Resulte. Mit den Schlüsselwörtern ORDER BY und ASC() oder DESC() können die Ergebnisse nach einer Variable auf- oder absteigend sortiert werden. Diese Befehle werden oft mit den Schlüsselwörtern LIMIT und OFFSET kombiniert. LIMIT gibt dabei die Maximalanzahl von Ergebnissen an, während mit OFFSET die Anzahl der ignorierten Ergebnisse angegeben werden kann (siehe Yu, 2011).

CONSTRUCT-Anweisung

Eine CONSTRUCT-Anweisung wird laut Yu (2011) im Gegensatz zur SELECT-Anweisung, die dazu verwendet wird Informationen aus einem gegebenen RDF-Graph zu extrahieren, dazu verwendet, einen neuen RDF-Graph zu produzieren. Sie wird in der Regel dafür eingesetzt um einen bestehenden RDF-Graph in einen neuen RDF-Graph mit einer anderen Ontologie zu portieren.

[37] `http://www.w3.org/TR/rdf-sparql-query/#OperatorMapping`

DESCRIBE-Anweisung

Mit DESCRIBE-Anweisungen erhält man Informationen über einen RDF-Graph. Diese Abfrage weist den SPARQL Query-Prozessor an eine bestimmte Resource zu beschreiben. Die zurückgelieferte Beschreibung ist wiederum ein RDF-Graph. Der Inhalt dieses Graphen wird vom Query-Prozessor, und nicht von der Abfrage selbst bestimmt (siehe Yu, 2011).

ASK-Anweisung

Eine ASK-Anweisung kommt nach Yu (2011) einer Entscheidungsfrage gleich. Empfängt der Query Prozessor eine Abfrage die mit dem Schlüsselwort ASK gekennzeichnet ist, gibt er den boolschen Wert TRUE oder FALSE zurück, je nachdem, ob das in der Abfrage enthaltene Graph-Pattern Ergebnisse zurückliefern würde oder nicht.

SPARQL 1.1

SPARQL 1.1 ist eine von der „SPARQL Working Group“[38] entwickelte Erweiterung der ursprünglichen Abfragesprache SPARQL. Die Entwicklung ist noch nicht abgeschlossen und infolgedessen auch noch nicht als Standard etabliert. Im Dokument „SPARQL 1.1 Query Language; W3C Working Draft 05 January 2012“[39] sind diverse Ergänzungen zur Abfragesprache SPARQL beschrieben die auch schon von einigen SPARQL-Endpoints untertztützt werden.

Die neuen Funktionalitäten betreffen die folgenden Punkte:

- **Aggregationsfunktionen:** SPARQL unterstützt nun einen Satz von Aggregationsfunktionen, die auf eine Gruppe von Resultaten angewandt werden können. Es werden die Aggregationsfunktionen COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT und SAMPLE unterstützt.
- **Subqueries:** Es ist nun möglich Abfragen in andere Abfragen einzubetten.
- **Negation:** Es werden zwei Arten von Negation untersützt. Einerseits im Kontext von Filterfunktionen und andererseits die Negation von Resultaten die zu einem anderen Pattern gehören.
- **Ausdrücke im SELECT-Abschnitt:**

  In SPARQL wird die SELECT-Anweisung benutzt, um festzulegen, welche Eigenschaften über Variablen in der Ergebnismenge einer Abfrage vorkommen sollen. Mit SPARQL 1.1 kann die SELECT-Anweisung auch dafür verwendet werden, neue Variablen für die Ergebnismenge zu definieren. Dies wird mit der Anweisung „(Ausdruck AS variable)“ innerhalb des SELECT-Abschnittes bewerkstelligt.
- **Eigenschaftspfade:** Als Eigenschaftspfad wird eine Verbindung zwischen zwei beliebigen Knoten eines RDF-Graphen bezeichnet. Im einfachsten Fall hat ein solcher Pfad die Länge von 1. Dieser Pfad entspricht einem normalen Tripel der Form Subjekt (Knoten 1), Prädikat (Eigenschaft) und Ojekt (Knoten 2). Die Enden des Pfades

[38] http://www.w3.org/2009/sparql/wiki/Main_Page
[39] http://www.w3.org/TR/sparql11-query/

können als RDF-Terme oder Variablen definiert werden, wobei Variablen nur am Ende des Pfades vorkommen dürfen, und nicht innerhalb des Pfades. Mit Pfaden sind präzisere Ausdrücke in Graph Patterns möglich, sowie der Match eines Patterns mit der Verbindung von zwei Resourcen über eine beliebige Pfadlänge bzw. eine beliebige Anzahl von Eigenschaften.

- **Zuweisung:**

  Es ist möglich die Ergebnisse eines Ausdrucks mit Hilfe der Definition einer Variable mit dem Ergebnis des Ausdrucks zu verknüpfen und zum Resultat der Query hinzuzufügen. Die Verknüpfung erfolgt über drei verschiedene Möglichkeiten. Entweder über das Schlüsselwort BIND, direkt innerhalb der SELECT-Anweisung, oder in einem GROUP BY Block. Die Zuweisung erfolgt dabei immer in der Form „Ausdruck AS ?var“.

- **Kurzform von CONSTRUCT:**

  Es wird eine Kurzform der CONSTRUCT-Anweisung eingeführt die dann angewendet werden kann, wenn es sich beim Pattern um ein einfaches Graph-Pattern ohne FILTER-Anweisung handelt, und das Template mit dem Pattern identisch ist.

- Ein erweiterter Satz von Funktionen und Operatoren.

2.6 Linked Open Data (LOD)

Heath and Bizer (2011) bezeichnen Linked Open Data (LOD) als verknüpfte, öffentlich zugängliche Daten. Man versteht darunter Methoden für die Veröffentlichung und Verlinkung von strukturierten Daten im Internet. Dieses so erzeugte Netz von strukturierten Daten wird als „Linked Data“ bezeichnet und deckt eine Vielzahl von Domänen ab. Das Spektrum der angebotenen Daten verläuft nach Bizer et al. (2009a) über Personen, Firmen, Bücher, wissenschaftliche Arbeiten, bis hin zu Filmen, Genetik, Medikamente oder statistische und wissenschaftliche Daten, die nach bestimmten Prinzipien veröffentlicht wurden.

2.6.1 Das LOD Projekt

Linked Open Data ist nach Dengel (2012) eine Community-Bestrebung der es darum geht große Datensätze mit semantischen Technologien miteinander in Beziehung zu setzen und schnell und einfach maschinenlesbar zu veröffentlichen, damit einerseits weitere Datensätze mit den vorhandenen Daten verknüpft werden, und andererseits die Möglichkeit besteht auf diesen Daten aufbauend Anwendungen zu schreiben. Diese verknüpften Datensätze stellen Knoten in der so genannten „Linked Open Data Cloud“ dar. Abbildung 2.5 zeigt einen Stand der LOD-Cloud vom 19. September 2011 mit 295 Datenknoten[40], die aus circa 31 Milliarden Datensätzen und mehr als 503 Millionen Verlinkungen bestehen (siehe Chris Bizer and Cyganiak, 2012). Verlinkungen zwischen den Datensätzen werden als Pfeile dargestellt. Die Größe der Knoten bzw. die Dicke der Pfeile spiegelt die Anzahl der Datensätze bzw. der Verlinkungen wider.

Prominente Datensätze sind beispielsweise DBpedia[41](DBpedia ist eine RDF-Version

[40] http://richard.cyganiak.de/2007/10/lod/
[41] http://www.dbpedia.org

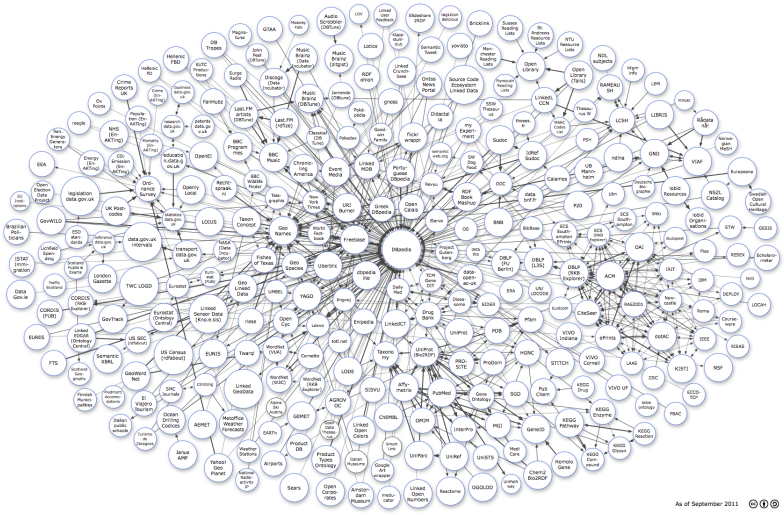

Abbildung 2.5: Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. (`http://lod-cloud.net/`)

von Wikipedia[42], siehe Abschnitt 2.7), der auf Personen spezialisierte Datensatz „Friend of a Friend“[43] (FOAF) oder GeoNames[44] (spezialisiert auf geographische Informationen).

LOD Designgrundlagen

Tim Berners Lee definierte in seinem Artikel „Design Issues: Linked Data note“, Berners-Lee (2006), vier Designgrundlagen, die man bei der Generierung eines neuen Linked Data Knoten beachten sollte:

1. Verwende URIs zur Identifizierung von Dingen.
2. Verwende HTTP-URIs, um die Daten einerseits für Maschinen, und andererseits auch für Menschen erreichbar zu machen.
3. Verwende Standards wie RDF und SPARQL um sinnvolle Informationen auf diesen URIs anzubieten.
4. Die mit diesen URIs verlinkten Informationen sollten weitere URIs beinhalten, um weiterführendes Wissen verfügbar zu machen.

2.6.2 Publikation von Linked Data

Dengel (2012) beschreibt eine typische Anwendung von Linked Open Data in der Publikation von bereits bestehenden, strukturierten Daten und der Vernetzung dieser Daten mit der Linked Open Data Cloud (siehe Abbildung 2.5). Bei der Umwandlung von existierenden

[42] `http://www.wikipedia.org/`
[43] `http://www.foaf-project.org/`
[44] `http://www.geonames.org`

Datensätzen in semantische Datensätze sind die vier Designgrundlagen von Tim Berners Lee aus Abschnitt 2.6.1 zu beachten. Der erste Schritt der Umwandlung besteht in der Wiederverwendung der bereits vorhandenen Resourcen. Im Kontext von Linked Data bzw. dem Semantic Web bedeutet das, dass man sich über die jeweilige Domäne der vorhandenen Daten erkundigt, und eventuell bereits existierende Vokabulare oder Ontologien der Domäne ausfindig macht. Der nächste Schritt ist die Verbindung mit bereits existierenden Konzepten. Das bedeutet die Verlinkung von vorhandenen Konzepten in der LOD-Cloud mit den neuen lokalen Konzepten, zum Beispiel über die Vewendung des OWL-Statements „owl:sameAs“. Mit diesem Statement kann man eigene lokale Konzepte mit anderen Konzepten verbinden, ohne die Kontrolle über das eigene Konzept zu verlieren. Man kann ohne Probleme eigene Daten zum lokalen Konzept hinzufügen. Web-Services wie „sameas.org“ helfen bei der späteren Aufschlüsselung dieser Fremdlinks und der Zusammenführung der verschiedenen Daten eines Konzeptes (siehe Dengel, 2012). Hat man nun seine Daten mit Semantik angereichert und in die LOD-Cloud integriert, besteht der nächste Schritt in der Einrichtung eines SPARQL-Endpoints um den Benutzern das Durchführen von komplexer SPARQL-Abfragen zu ermöglichen. Der letzte Schritt nach Dengel (2012) besteht idealerweise im Anlegen eines VoiD-Profil (Vocabulary of Interlinked Datasets[45]) für die eigene Datenquelle. VoiD ist ein RDF-Schema zur Beschreibung von Linked Datasets mit dem Informationen über die Domäne, der verwendete Klassen, deren Eigenschaften und des technischen Zugriffs auf die Daten, sowie Zusatzinformationen wie beispielsweise die Urheber der Daten und die Verlinkung mit anderen Datensätzen definiert und veröffentlicht werden können. Die Beschreibung erfolgt über die zwei Hauptklassen „void:Dataset“ und „void:Linkset“.

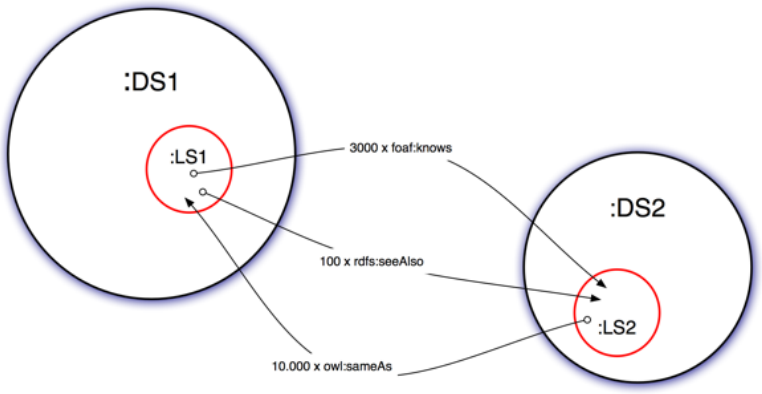

Abbildung 2.6: VoiD Verlinkungskonzept (`http://semanticweb.org/images/8/8c/Void-linkset-conceptual.png`)

Die Klasse Dataset beinhaltet Eigenschaften für die Beschreibung der Daten wie beispielsweise „void:classes“, „void:properties“ oder „void:sparqlEndpoint“. Die Klasse Linkset ist eine Subklasse der Klasse Dataset und beinhaltet Statements die die Verlinkung mit anderen Datensätzen beschreiben. Bei diesen Tripeln ist nach Richard Cyganiak and Hau-

[45] `http://semanticweb.org/wiki/VoiD`

senblas (2012) das Subjekt die Resource die in einem Datensatz enthalten ist, und das Objekt eine Resource in einem anderen Datensatz. Mit Hilfe dieser Tripel können sehr detaillierte Beschreibungen über die Verlinkungen der Datensätze gegeben werden, zB Infomationen über die Anzahl und die Art der verwendeten Links (siehe auch Abbildung 2.6).

Eine vollständige Beschreibung des VoiD-Vokabulars ist unter `http://vocab.deri.kie/void` zu finden.

LOD-Wrapping

Wie erfolgt nun die Umwandlung von bereits existierenden Daten in semantische Datensätze wie sie in Abschnitt 2.6.2 beschrieben wird? Eine Möglichkeit besteht laut Dengel (2012) in der Bereitstellung der Daten von existierenden Anbietern über unabhängige Services, dem so genannten „Wrappen". Beim Wrappen von Fremddaten muss darauf geachtet werden, dass die Daten über eine Freie Lizenz wie der „GNU Free Documentation License"[46] oder einer der „Creative Common (CC)" Lizenzen[47] zur Verfügung stehen. Das Wrappen von vorhandenen Web Services hat den Vorteil, dass der Originalservice nicht beeinflusst wird, da die resultierenden Linked-Data Dienste vollkommen unabhängig von den Ursprungsseiten sind, wodurch auch sehr komplexe Web-Services gewrappt werden können. Durch das Wrappen können neue Extraktionsmechanismen in den Service integriert werden, ohne die Verfügbarkeit und Integrität des Ursprungsservice zu gefährden, und es kann von spezialisierten Communities durchgeführt werden, die sich mehr auf den technischen Aspekt der Daten konzentrieren können als der ursprüngliche Anbieter der Daten.

Dengel (2012) beschreibt verschiedene Ansätze für das Wrappen von bereits vorhandenen Services:

- **Dumpkonvertierung:** Als Dumpkonvertierung versteht man die Konvertierung einer kompletten, bereits existierenden Datenbank auf einmal. Der so resultierende RDF-Datensatz wird dann über einen Web-Service zur Verfügung gestellt. Ein Beispiel für einen Service der über Dumpkonvertierung erzeugt wird ist DBpedia (siehe auch Abschnitt 2.7). Ein Problem ist die mangelhafte Aktualität der so generierten RDF-Daten. Änderungen und Fehler in den Daten können erst mit dem nächsten Konvertierungszyklus eingebracht werden.

- **Onlinekonvertierung:** Dieser Ansatz geht laut Dengel (2012) einen anderen Weg. Im Gegensatz zur Dumpkonvertierung die alle Datensätze auf einmal konvertiert, arbeitet der Ansatz der Onlinekonvertierung on-demand. Ein Datensatz wird erst dann konvertiert wenn eine Anfrage nach einer entsprechenden Linked Data Resource vorliegt. Dieser Ansatz ist besonders bei Daten mit geringer Abhängigkeit untereinander anwendbar, da jeder Datensatz selbstständig konvertiert werden kann. Bei einer starken Abhängigkeit können aber Probleme auftreten. Die Abhängigkeit der Datensätze untereinander lässt sich über die Ontologie der die Daten zugrunde liegen abschätzen.

[46] `http://www.gnu.org/copyleft/fdl.html`
[47] `http://creativecommons.org/licenses/`

Ein weiterer Vorteil der Onlinekonvertierung ist die Möglichkeit Feedback der Nutzer sehr zeitnah in Form von Vorschlägen, Fehlermeldungen oder Zusatzinformationen die in den Originaldaten nicht berücksichtigt wurden sowie weitere Verlinkungen zu neuen LOD-Resourcen zu ermöglichen.

- **Spiegelung in ein Linked Data System:** Bei der Spiegelung werden die Daten nach Dengel (2012) direkt in ein semantisches System wie beispielsweise ein Semantic Wiki gespiegelt, und dort mit semantischer Information angereichert. Allerdings stellt hier die Erhaltung der Synchronität zwischen den beiden Datensätzen ein Problem dar. Die semantischen Datensätze sind anders aufgebaut als die originalen Datensätze, wodurch eine Änderung sehr schwer in beiden Datensätzen umzusetzen ist.

Laut Dengel (2012) wird derzeit der Ansatz der Dumpkonvertierung am häufigsten eingesetzt. (Paradebeispiel hierfür ist DBpedia.) Der Trend gehe allerdings in Richtung Onlinekonvertierung.

2.6.3 Zugriff auf Linked Data

Nach Dengel (2012) bietet die Publikation von Datensätzen nach den in Abschnitt 2.6.1 beschriebenen Prinzipien von Tim Berners Lee einen...

> ... einheitlichen und integrierten Zugriff auf die Daten verschiedener Anbieter. Damit wird die Entwicklung von Anwendungen möglich, welche es Nutzern erlauben, von dem Wissen verschiedener Quellen zu profitieren. Aufgrund der Verknüpfung zwischen Daten und unterschiedlichen Anbietern können sogar Quellen berücksichtigt werden, welche einer Anwendung zunächst unbekannt sind.

Für den Zugriff auf die Daten über menschliche Nutzer oder Anwendung muss ein Web-Server in der Lage sein zu erkennen ob er einen Menschen oder eine Maschine mit Daten versorgen, und diese dementsprechend aufbereiten soll. Hierfür wird der Begriff der „Content Negotiation“ eingeführt.

HTTP Content Negotiation

Laut Dengel (2012) ist ein Web-Server über sogenannte „Inhaltsvereinbarungen“ in der Lage zu erkennen, ob ein Mensch oder ein Programm eine Anfrage an ihn gestellt hat. Ein Programm wird die Daten die es erhält anders vearbeiten als ein Mensch, der ein für ihn lesbares Format der Daten erwartet. Eine Maschine hingegen erwartet ein maschinenlesbares Format wie RDF. Das HTTP-Protokoll enthält bereits einen Mechanismus um dies umzusetzen. Ein HTTP-Request enthält eine Liste von Inhaltstypen (MIME-Types), die das Format definieren, das der Client als Antwort erwartet. Grundsätzlich verlangen menschliche Nutzer den MIME-Type „text/html“ und Anwendungen den MIME-Type „application/rdf+xml“. Am Server werden die Daten dann im entsprechenden Format aufbereitet und an den Client zurückgegeben.

Abbildung 2.7 zeigt den Vorgang am Beispiel von DBpedia, in dem zwei verschiedene Clients die URI eines bestimmten Konzeptes abfragen. Der Server leitet die Anfragen je nach MIME-Type auf die dem Format entsprechende URL weiter.

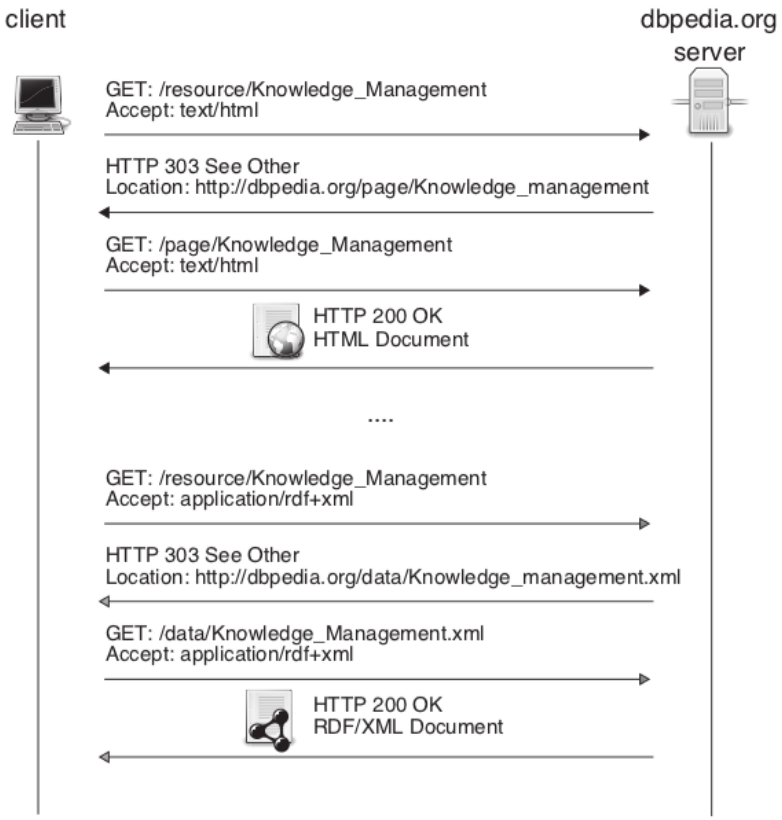

Abbildung 2.7: HTTP Content Negotiation am Beispiel von DBpedia (siehe Dengel, 2012)

Der Zugriff auf Linked Data erfolgt also auf verschiedene Weisen. Einerseits kann der Zugriff auf das „Web of Linked Data“ auf die gleiche Art wie im gewöhnlichen WWW über einen Browser erfolgen, mit dem man sich manuell von einer Resource zur anderen klicken kann, andererseis können die Daten automatisiert über Applikationen ausgelesen und verarbeitet werden.

Linked-Data Browser

Diese Browser orientieren sich nach Dengel (2012) an der Funktionsweise von konventionellen Webbrowsern. Man gibt entweder einen Suchbegriff oder eine URI in ein Adressfeld ein und erhält daraufhin, meist in tabellarischer Darstellung, die als Linked-Data verfügbaren Daten angezeigt. Jeder Eintrag entspricht dabei einem RDF-Tripel. Man unterscheidet zwischen zwei verschiedenen Arten von Browsern.

- **Entitätsbasierte Browser:** Diese Browser sind dadurch gekennzeichnet, dass die Navigation auf Ebene der Entitäten beruht. Hat der User einen Startlink eingegeben oder gewählt, werden dessen Daten in Form von Tripel angezeigt. URIs innerhalb eines Tripels werden wiederum als Links dargestellt, die vom User gewählt werden können und führen zu weiteren Daten, die mit dieser URI verknüpft wurden. Es steht immer die aktuelle, zuletzt angeklickte Entität im Fokus. Beispiele für diese Art von Browser sind „Marbles“[48], „Sig.ma“[49] oder der „Open Link Data Explorer“[50].
- **Entitätsmengenbasierte Browser:** Wie der Name schon sagt zeigen entitätsmengenbasierte Browser nicht nur eine Entität, sondern mehrere Entitäten gleichzeitig an. Der User kann im Bezug auf die Werte ihrer Eigenschaften die Menge der angezeigten Entitäten über Filterkriterien einschränken. Man spricht hierbei von einer „facettenbasierten Suche“. Ein Beispiel für diese Art von Browser ist der „Wikipedia-Facet-Browser“[51].

Suchmaschinen

Nach Bizer et al. (2009a) unterscheidet sich eine Suchmaschine für Linked Data nicht wesentlich von einer Suchmaschine des WWW. Der User gibt ein oder mehrere Schlüsselwörter an und bekommt ein Ergebnis aufgrund einer Abfrage von indizierter Daten. Beispiele für Linked Data Suchmaschinen sind „Falcons“[52]„SWSE“[53] oder „Sindice“[54]. Die Ergebnisse beinhalten neben Links meist noch weitere Informationen zu gefundenen Resourcen. So werden die wichtigsten Eigenschaft des jeweiligen Typs der Ergebnisse zusammengefasst, oder zusätzliche Informationen aus dem Web angezeigt. Abbildung 2.8 zeigt ein Suchergebnis der Semantic Web Suchmaschine Falcons.

Neben diesen für menschliche Nutzer gedachten Suchmaschinen beschreibt Bizer et al. (2009a) noch weitere Services, die auf Anfragen von Applikationen spezialisiert sind und

[48] http://marbles.sourceforge.net/
[49] http://sig.ma/
[50] http://linkeddata.uriburner.com/ode/
[51] http://dbpedia.neofonie.com/browse/
[52] http://ws.nju.edu.cn/falcons/objectsearch/index.jsp
[53] http://swse.deri.org/
[54] http://sindice.com/

dementsprechende Programmierschnittstellen anbieten um RDF Dokumente, die einer speziellen URI angehören oder bestimmte Schlüsselwörter enthalten, zu finden. Diese Services bieten eine fertige Infrastruktur für die Suche im Web of Data, sodass Applikationen die Arbeit des Crawlens und Indizierens abgenommen wird. Beispiele für Applikationsorientierte Suchmaschinen sind „Watson“[55] oder „Swoogle“[56].

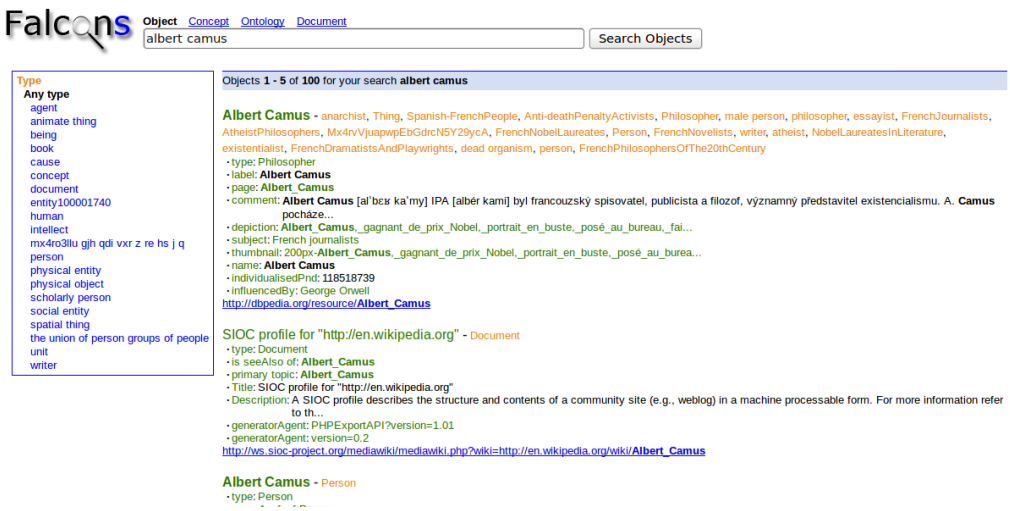

Abbildung 2.8: Linked Data Suchmaschinen unterscheiden sich nicht wesentlich von anderen Suchmaschinen des WWW. Hier das Suchinterface der Semantic Web Suchmaschine Falcons.

Linked Data basierte Applikationen

Die oben beschriebenen Linked Data Browser und Suchmaschinen verfolgen einen sehr generischen Ansatz in ihrer Funktionalität. Daneben haben sich nach Bizer et al. (2009a) eine Reihe von sehr spezialisierten Anwendungen entwickelt, die Mashups aus Daten einer bestimmten Domäne von verschiedenen Linked-Data Quellen erzeugen. Diese Anwendungen sind für spezifische Anwendungsfälle gedacht und bieten dafür angepasste Funktionen. Beispiele hierfür sind die Bewertungsplattform „Revyu[57], oder „DBpedia Mobile“[58], eine mobile Anwendung die sich mittels GPS-Koordinaten Informationen über interessante Orte aus DBpedia inklusive Bewertungen von Revyu und Fotos über den Linked Data Wrapper der „Flickr photosharing API“ beschafft und dem User in einem entsprechenden Interface präsentiert.

Ein weiteres Beispiel ist das Musikportal „BBC-Music“[59], das RDF-Repräsentationen seiner Daten Usern und Applikationen zur Verfügung stellt.

[55] http://watson.kmi.open.ac.uk/REST_API.html
[56] http://swoogle.umbc.edu/
[57] http://revyu.com/
[58] http://wiki.dbpedia.org/DBpediaMobile
[59] http://www.bbc.co.uk/music

2.7 DBpedia

DBpedia ist nach Bizer et al. (2009b) ein Projekt der Freien Universität Berlin[60], der Universität von Leipzig[61], Openlink Software[62] und der Linking Open Data Community (W3C SWEO)[63] mit dem Ziel strukturierte Information aus der freien Enzyklopädie Wikipedia[64] zu extrahieren und im Web verfügbar zu machen.

Der erste RDF Datensatz wurde nach Bizer (2011) 2007 unter dem Namen DBpedia veröffentlicht und wächst seitdem stetig an. Die aktuelle DBpedia-Version 3.7 und beinhaltet Beschreibungen von mehr als 3.64 Millionen Dingen in 97 verschiedenen Sprachen, 2.7 Millionen Verweise auf Bilder, 6.2 Millionen externe Links zu anderen RDF- Datensätzen und 740 tausend Wikipediakategorien. Er besteht aus mehr als 1 Milliarde einzelner RDF-Tripel.

DBpedia ist nach Bizer et al. (2009b) ein gewaltiger RDF Datensatz und ein wichtiger Bestandteil der Linked Open Data Cloud. In Abbildung 2.5 ist DBpedia als ein zentraler Punkt der LOD-Cloud nicht zu übersehen.

2.7.1 Datenextraktion aus Wikipedia

Um strukturierte Daten aus den unstrukturierten Datensätzen von Wikipedia zu bekommen, benötigt man einen Extraktionsmechanismus. Das DBpedia-Team hat ein Extraktionsframwork entwickelt, mit dessen Hilfe es möglich ist verschiedene Arten von strukturierten Daten aus Wikipedia zu extrahieren.[65]

Das Framework besteht aus einigen Haupt- und Nebenkomponenten. Die Hauptpomponenten sind in Abbildung 2.9 dargestellt.

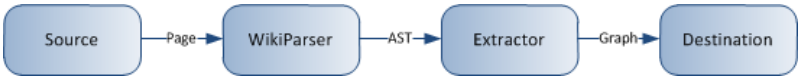

Abbildung 2.9: DBpedia Extraktionsframework Komponenten (`http://wiki.dbpedia.org/Documentation`)

- **Source:** Die Komponente Source bereitet Inputs aus verschiedenen Datenquellen zu einheitlichen „WikiPage“-Instanzen auf. Datenquellen können beispielsweise XML-Dumps, MediaWikis, Dateien und mehr darstellen.
- **WikiParser:** Die Parserkomponente übernimmt die WikiPage-Instanz und erstellt daraus einen „Abstract Syntax Tree“ (AST).
- **Extractor:** Der Extraktor wandelt die Knoten des AST in einen Graph mit Aussagen über den Knote um. Es existieren verschiedene Extraktoren mit unterschiedlichen Aufgaben, wie das Extrahieren von Label-Informationen, Links auf verwandte

[60] `http://www.fu-berlin.de/en`
[61] `http://www.zv.uni-leipzig.de/`
[62] `http://www.openlinksw.com/`
[63] `http://www.w3.org/blog/SWEO/`
[64] `http://www.wikipedia.org/`
[65] `http://wiki.dbpedia.org/Documentation`

Wikipedia-Artikel oder auf interne DBpedia Instanzen, Geoinformationen wie Koordinaten, Bildinformationen oder Infoboxen, denen eine besondere Rolle bei der Extraktion zukommt (siehe auch Abschnitt 2.7.1). Die extrahierten Informationen stehen nun in einer Graphstruktur zur Verfügung.

- **Destination:** Die letzte Komponente übernimmt einen Graph von der Extractor-Komponente und wandelt ihn in ein bestimmtes Format für das gewünschte Ziel um. Als Beispiele für mögliche Ziele sind Dateien oder Stringbuffer zu nennen.

Infoboxes als Quelle

Wie können nun automatisiert strukturierte Informationen aus einzelnen Wikiseiten gewonnen werden? Die Antwort darauf liefert nach Yu (2011) das Konzept der Infoboxen in Wikipedia und deren zugrundeliegenden Templatestruktur. Infoboxen dienen dazu, Informationen zu einem bestimmten Thema zusammenzufassen, und so dem Benutzer einen schnellen Überblick über das Thema der aktuellen Seite zu verschaffen ohne dass er den ganzen Text gelesen haben muss. Infoboxes werden über ein zentrales Template generiert, was den Vorteil der Einheitlichkeit im Bezug auf Struktur und Layout aller Boxen mit sich bringt. Listing 2.11 zeigt Auszüge aus dem Code hinter einer Infobox über die Stadt Graz. Man erkennt schnell, dass es sich dabei um eine Sammlung von Eigenschaften mit ihren Werten handelt. Es existieren unterschiedliche Templates für Infoboxen verschiedener Kategorien. Das Template das in Listing 2.11 zur Beschreibung der Stadt Graz verwendet wurde ist ein Template speziell für die Kategorie „Town“. Nach Yu (2011) liegt es nahe, diese Eigenschaften direkt in RDF Statements umzuwandeln. Dabei übernehmen die Eigenschaften die Rolle der Prädikate, die zugehörigen Werte die Rolle der Objekte und das Thema der Seite ist das zu allen Statements gehörige Subjekt.

```
{{Infobox Town AT
  |name=Graz
  |name_local=
  |image_photo = Graz clock tower1.jpg
  ...
  |image_caption= Graz Clock Tower in Schlossberg
  |state = [[Styria (state)|Styria]]
  |regbzk =
  |district = [[Statutory city]]
  | popkey= 60101
  | population = {{Metadata_population_AT-6|60101}}
  | population_as_of = {{Metadata_population_AT-6|date}}
  ...
  |area = 127.56
  |elevation = 353
  ...
  |website = [http://www.graz.at www.graz.at]
}}
```

Listing 2.11: Codeausschnitt der Wikipedia Infobox über die Stadt Graz

Die auf diese Art aus den Wikipedia-Artikeln extrahierten RDF-Statements haben nach Yu (2011) allerdings ein schwerwiegendes Problem. Sie wurden ohne eine Ontologie, die genaue Definitionen von Klassen und Eigenschaften sowie deren Abhängigkeiten bereitstellt erzeugt, und die damit beschriebenen Resourcen haben somit keine Aussagekraft. Es ist nicht möglich aus ihnen Schlussfolgerungen zu ziehen oder sie mit anderen Datenquellen zu verbinden. Frühere Versionen von DBpedia hatten genau dieses Problem. Daher wurde mit der DBpedia-Version 3.2 ein neues Extraktionsverfahren, basierend auf von Hand erzeugten Mappings der DBpedia-Infoboxen auf die DBpedia Ontologie[66], entwickelt.

Die DBpedia-Ontologie

Die DBpedia Ontologie ist laut Jakob (2011) eine domainübergreifende, auf Basis einer aus den Templates der meistgenutzten Infoboxen in Wikipedia von Hand erzeugten Ontologie. Sie beschreibt über 320 Klassen in einer hierarchischen Struktur die zusammen über 1.600 Eigenschaften aufweisen. Seit Version 3.5 existiert ein öffentlich zugängliches Mapping-Wiki[67] zur Erweiterung bzw. Abänderung der bestehenden Ontologie. So können neue Mappings von Klassen und Eigenschaften auf bestimmte Infoboxen von Interessierten selbstständig in die Ontologie integriert werden. Die über das Wiki erweiterte Ontologie ist mit dem nächsten offiziellen Realease des Datensatzes verfügbar, bzw. kurz nach der Bearbeitung über den DBpedia Live Endpoint[68] (siehe hierfür Abschnitt 2.7.1).

Die mit dieser mappingbasierten Methode erzeugten RDF-Dokumente sind nach Yu (2011) leichter abzufragen, da sie auf der einheitlichen Grundlage der Ontologie beruhen, und Applikationen sind in der Lage Schlussfolgerungen aus den Daten zu ziehen.

Extraktionsarten

Laut Bizer et al. (2009b) ist das Extraktionsframework von DBpedia für zwei verschiedene Workflows geeignet:

- **Dump-basierte Extraktion:** Die Wikimedia Foundation[69] publiziert monatlich einen SQL-Dump aller Wikipedia-Editionen, die monatlich als Basis für eine Update des DBpedia-Datasets dienen. Der Output der Dump-Konvertierung wird als Downlaod angeboten und über Linked-Data sowie den DBpedia eigenen SPARQL-Endpoint (siehe Abbildung 2.4) zugänglich gemacht.

- **Live Extraktion:** Das DBpedia Projekt erhielt nach Bizer et al. (2009b) Zugriff auf den „Wikipedia OAI-PMH live feed“, der über Änderungen in Wikipedia über einen Update-Stream informiert. Dieser Stream wird für die Extraktion neuer RDF-Daten herangezogen und bestehende Datensätze werden mit den neuen ausgetauscht. Dieser Ansatz ist derzeit in der Versuchsphase und Updates benötigen zwischen 5 und 10 Minuten um wirksam zu werden. Es wurde ein eigener Server für die Live-Extraktion eingerichtet und ist unter `http://dbpedia-live.openlinksw.com` erreichbar.

[66] `http://wiki.dbpedia.org/Ontology?v=181z`
[67] `http://mappings.dbpedia.org/index.php/Main_Page`
[68] `http://live.dbpedia.org/`
[69] `https://wikimediafoundation.org/wiki/Home`

2.7.2 Zugriff auf DBpedia

Der Zugriff auf den DBpedia Datensatz kann über verschiedene Wege erfolgen. Abbildung 2.10 beschreibt die Bereitstellung von Daten in verschiedenen Formaten über den „OpenLink Virtuoso Universal Server“[70]. Die Virtuoso-Infrastruktur bietet Zugang zu den RDF-Daten über einen SPARQL Endpoint sowie RDF- und HTML-Repräsentationen der DBpedia-Resourcen über HTTP. Daneben steht der gesamte Datensatz zum Download bereit.

- **Download:** Es stehen verschiedene Versionen und Teile des Datensatzes in unterschiedlichen Formaten zur Verfügung. Der Vorteil im direkten Download des Datensatzes liegt zum Beispiel im Performancegewinn der eigenen Anwendungen bei der Verwendung eines lokalen Datensatzes. Die Datensätze der aktuellen Version 3.7 sind unter `http://wiki.dbpedia.org/Downloads37` verfügbar.
- **Webbrowser und Linked Data:** DBpedia URIs wie beispielsweise `http://dbpedia.org/resource/Graz` können laut Bizer et al. (2009b) je nach Anfrage des Clients verschiedene Formate zurückliefern. Erfolgt die Anfrage über einen Semantic Web Agent wie einen Crawler oder Semantic Web Browser wird ein RDF Dokument erwartet, handelt es sich aber um einen traditionellen Web-Browser wird eine HTML Version des gleichen Inhalts zurückgegeben. Die Unterscheidung erfolgt über HTTP-Inhaltsvereinbarungen wie in Abschnitt 2.6.3 erläutert.
- **Sparql-Endpoint:** DBPedia bietet einen SPARQL-Endpoint für die Abfrage des DBpedia Datensatzes mit der Abfragesprache SPARQL über das SPARQL Protokoll. Der Endpoint ist erreichbar unter `http://dbpedia.org/sparql`. Der Enpoint unterstützt einige Features von SPARQL 1.1 sowie weitere Erweiterungen der Abfragesprache wie eine Volltextsuche über die Werte bestimmter Prädikate oder Aggregationsfunktionen. Der Endpoint wird über einen Virtuoso Universal Server betrieben[71] (siehe Bizer et al., 2009b).

[70] `http://dbpedia.org/resource/Virtuoso_Universal_Server`
[71] `http://virtuoso.openlinksw.com`

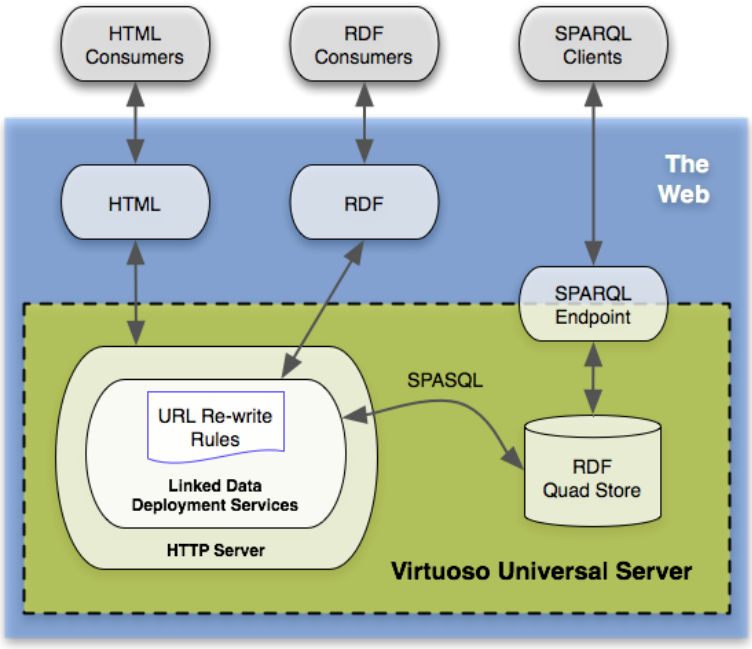

Abbildung 2.10: DBpedia Data Provision Architecture (`http://wiki.dbpedia.org/Architecture?v=14cg`)

Kapitel 3

Related Work

Es ist ein bekanntes Problem des Semantic Web, dass es ohne fundiertes Hintergrundwissen über die Struktur der verwendeteten Repositories und die Abfragesprache SPARQL sogar für erfahrene Nutzer teilweise sehr schwer und mühsam sein kann, benötigte Daten aus den verschiedensten Repositories zu extrahieren. Daher existieren mittlerweile viele Systeme die einen User bei der Formulierung von SPARQL-Abfragen unterstützen. Dieses Kapitel nimmt aus der Vielzahl an Systemen einige heraus, gibt einen Überblick über deren Funktionaltät und diskutiert deren Stärken und Schwächen. Es werden Eigenschaften herausgearbeitet, die für einen einfach zu benutzbaren Query-Wizard unerlässlich sind. Im Vordergrund bei der Analyse der Tools stehen:

- **Nutzbarkeit:** Damit ist die intuitive Benutzbarkeit des Systems durch den User gemeint. Das System soll ohne längere Einarbeitungszeit in seinem vollen Funktionsumfang für den User erfassbar und auch anwendbar sein.
- **Vorwissen:** Der Grad des benötigten Vorwissen im Bezug auf RDF, Linked-Data und die Abfragesprache SPARQL soll möglichst gering ausfallen. Das System soll für User ohne jegliches Vorwissen als auch für erfahrenen Benutzer anwendbar sein.
- **Interoperabilität:** Das Zusammenspiel mit verschiedenen Repositories soll durch das System unterstützt werden. Mit diesem Punkt ist einerseits die Anwendung des Systems auf verschiedene Wissensbasen oder RDF-Repositories unabhängig voneinander als auch die Kombination von Daten aus verschiedenen Quellen gemeint.
- **Wiederverwendbarkeit:** Die Möglichkeit der Weiterverwendung der generierten Abfragen und damit aquirierter Daten ist ein wichtiger Bestandteil eines Abfrageassistenten. Es sollte eine Möglichkeit bestehen, die erstellten Abfragen bzw. generierten Ergebnisse zu sichern, zu exportieren oder auch in externen Anwendung einzubinden.
- **Plattformunabhängigkeit:** Ein weiterer wichtiger Aspekt ist die Verfügbarbeit der Anwendung. In dieser Arbeit liegt der Fokus auf plattformunabhängigen, über das Web erreichbare Anwendungen. Standalone-Applikationen und Frameworks für die unterschiedlichsten Programmiersprachen werden in diesem Abschnitt nicht berücksichtigt.

3.1 Semantic Web Search Engines und Browser

Die erste Möglichkeit um Informationen über den Aufbau eines Repositories zu erhalten ist die Verwendung von Semantic Web Browser und Suchmaschinen. Mit ihnen ist es, meist unter Verwendung einer Schlüsselwortsuche, möglich, einzelne Resourcen aufzuspüren, deren Struktur zu erkunden und Verlinkungen zu folgen. Im Folgenden werden zwei unterschiedliche Systeme beschrieben. Die Suchmaschine „Falcons“ und der entitätsmengenbasierte, eigens für die Abfrage von DBpedia entwickelte „Wikipedia Facet Browser“.

3.1.1 Falcons

Falcons ist eine Semantic Web Suchmaschine und bietet die Möglichkeit über eine Schlüsselwortsuche Informationen über einen Begriff zu erhalten. Die Suche des Begriffs erfolgt dabei auf mehreren Ebenen. Einerseits auf der Konzeptebene, also im Kontext der unterstützten Vokabulare, in Klassen und Properties, und andererseits auf der Objektebene. Objekte stehen hier für einzelne Resourcen oder Entitäten des Repositories. Auf dieser Objektebene ist auch eine Filterung nach Typ möglich. Als Ergebnis erhält man eine Auflistung von Resourcen mit einigen Eigenschaften. Weitere Einschränkungen der Ergebnismenge sind nicht möglich. Der Vorteil liegt in der intuitiven Handhabung des Systems, da keinerlei Vorwissen verlangt wird und sich das System sehr am Konzept einer herkömmlichen Suchmaschine orientiert, das jedem Internetnutzer hinlänglich bekannt sein dürfte. Diese Eigenschaft ist ein typisches Merkmal aller Semantic Web Suchmaschinen. Sie nutzen die Fähigkeiten des Semantic Web im Sinne von Linked Data, indem sie viele unterschiedliche Datenquellen abdecken, und die aus der Suche resultierende Ergebnismenge eine Kombination von Resourcen aus unterschiedlichen Quellen darstellt (siehe auch Abschnitt 2.6). In Abbildung 3.1 wird ein Teil eines Resultats einer Schlüsselwortsuche in Falcons dargestellt. Man erkennt die Einschränkung der Suche auf den Typ „mountain“ und die Einbeziehung der Suchbegriffe auf Objektebene und Vokabularebene.

Bei der direkten Eingabe eines Suchbegriffs erhält man nach Auswahl des richtigen Typs („Mountain“) zwar eine Übersicht über eine oder mehrere Resourcen, es ist aber nicht möglich einzelne Eigenschaften von Resourcen gezielt auszuwählen und im Ergebnis anzeigen zu lassen. Eine Unterstützung in Form einer Autovervollständigung ist nicht gegeben. Man kann sich nur nach Webbrowser-Manier zur HTML-Repräsentation der einzelnen Resourcen durchklicken (siehe auch Abbschnitt 2.6.3).

3.1.2 Faceted Wikipedia Search (Neofonie)

„Faceted Wikipedia Search“[1] ist ein entitätsmengenbasierender Browser (siehe Abschnitt 2.6.3), der speziell für den DBpedia-Datensatz entwickelt wurde.

Die „Faceted Wikipedia Search“ erlaubt laut Hahn et al. (2010) komplexe Abfragen wie „Welche Flüsse fließen in den Rhein und sind länger als 50 Kilometer?“ oder „Welche Wolkenkratzer in China haben mehr als 50 Stockwerke und wurden vor dem Jahr 2000 erbaut?“. Um diese Art von Anfragen beantworten zu können benötigt man ein entsprechendes Interface. *Neofonie* setzt dabei auf das Paradigma der facettenbasierten Suche. Dabei teilt der User eine Menge von Entitäten in mehrere Untermengen ein, wobei jede

[1] `http://dbpedia.neofonie.de/browse`

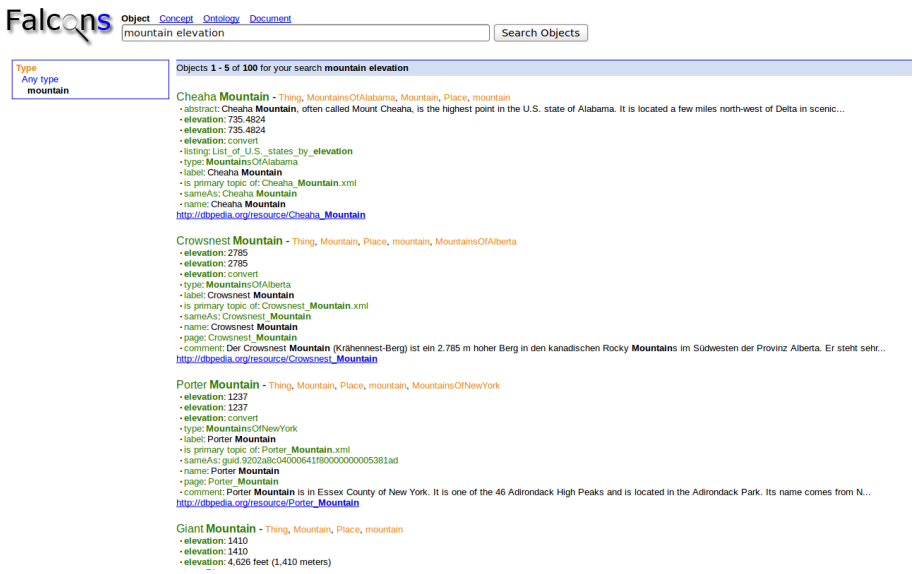

Abbildung 3.1: Suchergebnis der Semantic Web Suchmaschine Falcons

Untermenge mit einer Einschränkung einer Eigenschaft gekennzeichnet ist. Diese Eigenschaften werden als *Facette* bezeichnet.

Ein facettenbasiertes Suchinterface für Wikipedia muss nach Hahn et al. (2010) folgende Anforderungen erfüllen:

1. Die Extraktion von sinnvollen strukturierten Daten aus Wikipedia.

2. Das System muss in der Lage sein mit der großen Anzahl an unterschiedlichen Typen der in Wikipedia angebotenen Entitäten und der damit verbundenen hohen Anzahl an verschiedenen Facetten zurechtzukommen. Außerdem muss das System in der Lage sein bei einer eventuell hohen Anzahl von Facetten eines Types geeignete Heuristiken anzuwenden um relevante Facetten von unrelevanten Facetten zu trennen.

3. Eine facettenbasierte Suche produziert unter Umständen Resultate mit sehr vielen Entitäten. Ein solches System muss mit einer hohen Anzahl von zurückgelieferten Daten umgehen können um die Antwortzeit gering zu halten.

Die *Faceted Wikipedia Search* löst diese Herausforderungen nach Hahn et al. (2010) über die Verwendung von zwei Softwarekomponenten. Für die Datenextraktion wird das „DBpedia Information Extraction Framework“ (siehe hierzu auch Abschnitt 2.7.1) verwendet. Die zweite Komponente ist die kommerzielle Suchengine „neofonie search“ die für die Implementierung einer effizienten facettenbasieten Suche herangezogen wird.

Interface

In Abbildung 3.2 wird das Userinterface der Faceted Wikipedia Search angezeigt. Das Interface bietet die Möglichkeit einer freien Textsuche und der facettenbasierten Navigation.

Die Navigation erfolgt über ein Seitenpanel worin einige Eigenschaften des gewählten Entitätstypes angezeigt werden. Der User erhält zu jeder Eigenschaften Vorschläge mit den häufigsten Werten, hat aber auch die Möglichkeit eigene Filterkriterien für jede Eigenschaft zu definieren.

Das Resultat der Abfrage wird in Form einer Kurzfassung der gefundenen Wikipedia-Artikel dargestellt. Angezeigt wird die Überschrift des Artikels, ein Teasertext und, falls vorhanden, ein dem Artikel zugehöriges Bild.

Auswahl der relevanten Facetten

Der User hat zwei Möglichkeiten seine Suche zu beginnen. Entweder er verwendet die Möglichkeit der Volltextsuche oder er benutzt die am Beginn des Suchprozesses angezeigte Typfacette, und schränkt sein Ergebnis auf einen bestimmten Typ ein. Beides hat eine Ergebnismenge von Dokumenten zur Folge die nun mit der Verwendung weiterer Facetten weiter eingeschränkt werden kann. Die Auswahl der Dokumente erfolgt über die Neofonie-Suchengine, die alle Funktionalitäten zur Selektion der mit der Auswahl korrelierenden Dokumenten ermöglicht. (Hierbei ist die Funktionalität der „Boolean Queries“ besonders hervorzuheben.)

Die Auswahl der dem User präsentierten Facetten und der vorgeschlagenen Werte erfolgt auf Grund von Häufigkeitsdaten in der Resultatmenge. Die vorgeschlagenen Werte einer Facette werden dabei über die Anzahl der Dokumente mit dem gleichen Wert der jeweiligen Facette ermittelt. Die Antwort auf die Frage welche der eventuell recht vielen Facetten eines Typs dem User zur Auwahl präsentiert werden ist zum Zeitpunkt der Abfrage schon bekannt. Diese Facetten werden als „target factes“ bezeichnet.

Bei der Auswahl der „target facets“ werden nach Hahn et al. (2010) folgende drei Fälle unterschieden:

1. Am Beginn einer Suche wird nur die Typfacette angezeigt.

2. Wenn kein Typ über die Typfacette ausgewählt wird (zB wenn der User die Volltextsuche benutzt), werden die allgemeinsten Facetten wie „item-type“, „location“ oder „year-of-appearance“ als „target facets“ ausgewählt, da diese Facetten in den meisten Dokumenten vorkommen.

3. Wenn der User einen Typ ausgewählt hat werden die häufigsten Facetten des gewählten Typs als „target facets“ ausgewählt und dem User präsentiert.

Die Reihung der einzelnen Facetten erfolgt daraufhin auf Grund der Anzahl ihrer jeweiligen häufigsten Werte.

3.1.3 Fazit

Der große Vorteil von den an Hand zweier Beispiele beschriebenen Systemen zur Abfrage von RDF-Repositories liegt in der Anlehnung an für den User bekannte Suchparadigmen wie sie von aktuellen Suchmaschinen verwendet werden, womit sich auch mit RDF unerfahrene User leicht mit den Systemen zurechtfinden können. All diese Systeme sind webbasierend und somit plattformunabhängig. Semantische Suchmaschinen wie Falcons bieten die Möglickeit der Entitätensuche über die Grenzen eines Repositories hinaus. Es werden

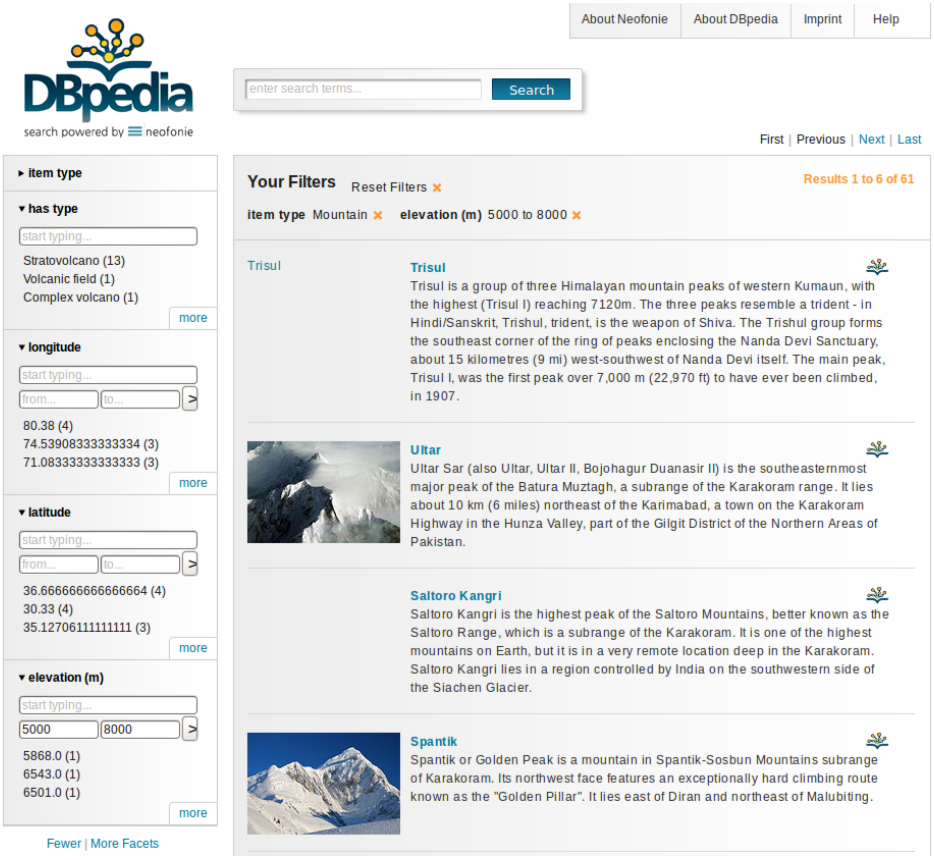

Abbildung 3.2: Suchergebnis facettenbasierten Wikipedia Suchmaschine Neofonie

Ergebnisse von verschiedenen Repositories die zuvor bei Falcon registriert und von dessen Crawler besucht wurden angezeigt. Ein direkter Vergleich verschiedener Entitäten und eine gezielte Auswahl der anzuzeigenden Prädikate bzw der möglichen Eigenschaften ist nicht möglich. Eine Hilfe des Users bei der Auswahl der Schlüsselwörter durch Vorschläge oder eine unterstützende Autovervollständigung ist nicht gegeben. Die Darstellung der Daten der getesteten Systeme eignet sich nicht für den Export, bzw. wird die Funktionalität der Speicherung oder Extraktion der generierten Abfrage nicht angeboten.

Semantic Web Search Engines und Browser eignen sich in ihrer derzeitigen Form für das gezielte Auffinden von einzelner Entitäten und zur allgemeinen Informationsbeschaffung über Repositories und Resourcen. Sie dienen als Ausgangspunkt und bieten dem Benutzer Ansatzpunkte für weitere Recherchen, indem sie Informationen über den Aufbau von Repositories, deren Klassenstrukturen und mögliche Eigenschaften von Resourcen bereitstellen. Sie sind aber nicht für die Generierung einer kontrollierten SPARQL-Abfrage geeignet.

3.2 Visuelle SPARQL Query Builder

Die in Abschnitt 3.1 vorgestellten Interfaces zur Abfrage oder Suche von RDF-Daten in Repositories verfolgen das Ziel den User vor der Komplexität der darunterliegenden Wissensbasis zu bewahren. Sie decken laut Lehmann and Bühmann (2011) die meisten Bedürfnisse der User mit vorbereiteten Abfragen für die üblichsten Anwendungsfälle ab, erlauben aber meist nicht den Zugang zur Struktur des darunterliegenden RDF-Graphen Struktur, womit die Breite der möglichen Abfragen für den User eingeschränkt ist, oder sie sind auf spezielle Repositories oder Datasets zugeschnitten, und somit nicht für andere Datensätze anwendbar bzw. verlangen einen hohen Aufwand im Falle einer Änderung des im Datensatz verwendetetn RDF-Schemas.

Visuelle SPARQL-Query Builder unterstützen den User bei der Erstellung von SPARQL-Abfragen über die Visualisierung des Generierungsprozesses, der User muss aber über ein Basiswissen über RDF, den Aufbau von SPARQL-Queries und außerdem eine Vorstellung über das darunterliegende Schema besitzen, um Queries visuell aufbauen zu können.

3.2.1 NITELIGHT: A Graphical Editor for SPARQL Queries

Die meisten Versuche den User bei der Erstellung einer Abfrage zu unterstützen orientieren sich laut Catarci et al. (1997) an grafischen oder visuellen Techniken in Form von „Visual Query Systems“ (VQS). VQS haben einige Vorteile die über den augenscheinlichen Nutzen der Unterstützung des Users bei der Erstellung syntaktisch korrekter Queries hinausgehen. So erhöhen sie laut Smart et al. (2008) die Effizienz und das Verständnis für die Abfragesprache SPARQL. Darüberhinaus verkürzen sie die benötigte Lernzeit.

„NITELIGHT“ ist eine webbasierte Applikation zur Erstellung von SPARQL-Queries mit Hilfe einer grafischen Notationssprache mit dem Namen „vSPARQL“. Die Notationssprache vSPARQL unterstützt alle syntaktischen Elemente der SPARQL-Spezifikation und ist dementsprechend umfassend. Laut Smart et al. (2008) ist das Tool nur für erfahrene SPARQL-Nutzer geeignet und ist auch dementsprechend ausgelegt.

vSPARQL

Die auf Tripel basierende Struktur von RDF (Subjekt, Prädikat und Objekt) eignet sich sehr gut für eine grafische Repräsentation. Subjekt und Objekt werden durch Knoten, und Prädikate als Kanten dargestellt. Damit lassen sich die meisten SPARQL-Features abbilden. Die Notationssprache vSPaRQL benutzt einen Farbcode zur Kennzeichnung verschiedener Knotentypen (siehe Abbildung 3.3). Die Abbildung zeigt die verschiedenen Knotentypen die zur Darstellung in vSPARQL verwendet werden. Knoten können gebundene oder ungebundene Variablen, oder definierte Resourcen darstellen, wobei man unter gebundene Variablen Variablen versteht, deren Werte mit dem Resultat der Query verknüpft sind. Ungebundenen Variablen wird nicht über das Resultat der Abfrage ein Wert zugewiesen, sondern sie werden beim Abfrageprozess selbst verwendet. Der Typ der „Non-Variable“ beschreibt eine URI, Literal oder einen Blank-Node. (Für mehr Informatioen über die Zuweisung von Variablen, die Verwendung von Blank-Nodes oder den generellen Abfrageprozess in SPARQL wird auf Abschnitt 2.5 verwiesen.) Prädikate werden, wie schon oben kurz erwähnt, als Kanten zwischen zwei Knoten dargestellt. Die Kante ist dabei mit einem Label verknüpft, das die URI des Prädikats beinhaltet.

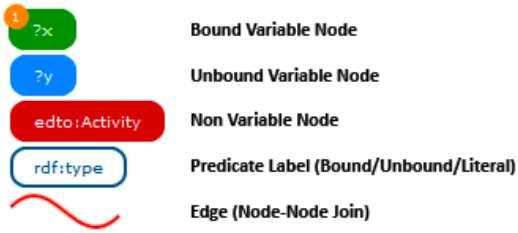

Abbildung 3.3: Grafische Notation einiger SPARQL-Komponenten

Für die Repräsentation von Subjekt und Objekt eines RDF-Tripels ist die Anordnung der Knoten entscheident. Knoten die Subjektknoten darstellen sind im Tripel links angeordnet. Die Verbindungskante verläuft von der rechten Seite des Knotens Richtung Objekt-Knoten, der sich rechts vom Subjekt-Knoten befindet. Die Verbindungskante schließt also mit der linken Seite des Objekt-Knotens ab. Abbildung 3.4 zeigt ein Beispiel einer einfachen SELECT-Abfrage.

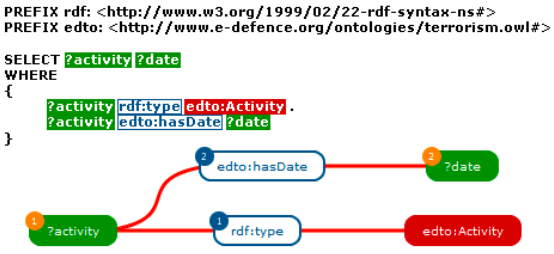

Abbildung 3.4: Grafische Repräsentation einer SELECT Query

Das zur Query gehörige Graph-Pattern ist folgendermaßen aufgebaut. Gesucht werden Resourcen vom Typ „edto:Activity“ und deren Daten. Die Variablen „?activity“ und „?date„ sind gebundene Variablen und damit vom Resultat abhängig. Das Objekt ist vordefiniert und daher eine Non-Variable. Wie in der Abbildung 3.4 ersichtlich besteht das Pattern aus zwei RDF-Tripel. Das gleichbleibende Subjekt wird in der grafischen Repräsentation zu einem Knoten zusammengefasst.

Neben diesen einfachen Pattern unterstützt vSPARQL nach Smart et al. (2008) noch weitere Pattern wie „group-patterns“ oder „union-graph-paterns“, sowie andere SPARQL-Konstrukte wie zum Beispiel „ORDER BY“, „FILTER“-Statements oder den Querytyp „CONSTRUCT“. Einige Konstrukte lassen sich nur sehr schwer in einer VQL abbilden und sind nicht in vSPARQL enhalten. Dazu gehören der Abfragentyp „DESCRIBE“ und „ASK“, sowie die Abfragezusätze „DISTINCT“, „LIMIT“ und „OFFSET“.

Interface

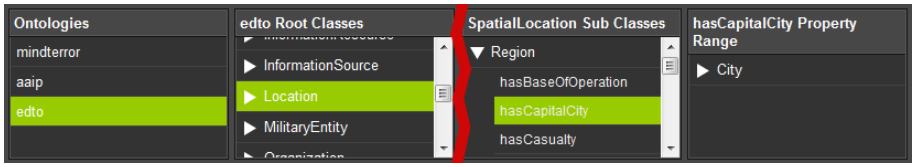

Abbildung 3.5: Der Nitelight Ontologie-Browser

NITELIGHT ist eine Webapplikation und wurde rein in JavaScript geschrieben. Das Userinterface besteht aus fünf Hauptkomponenten die von Smart et al. (2008) folgendermaßen beschrieben werden:

1. **Query Design Canvas:** Der Designbereich ist das Herzstück von NIGHTLIGHT. Hier werden die grafischen Repräsentationen der SPARQL-Queries mit den Konstrukten von vSPARQL erstellt und bearbeitet.

2. **Ontologie-Browser:** Der Ontologie-Browser gibt einen Überblick über die verwendeten bzw. in NIGHTLIGHT geladenen und damit verfügbaren Ontologien. Wie in Abbildung 3.5 ersichtlich ist, erlaubt der Browser eine Navigation über die einzelnen Hierarchiestufen der Ontologien. Angefangen bei den obersten Klassen der Ontologie kann sich der User von Subklasse zu Subklasse weiterklicken. Neben den Klassen bietet der Browser auch noch eine Übersicht über die mit den Klassen verbundenen Eigenschaften. Die Klassen oder Eigenschaften können aus dem Browser per Drag-and-Drop direkt in den Design-Canvas gezogen werden.

3. **Quick Toolbar:** Eine kleine Werkzeugleiste mit den wichtigsten Befehlen zur Manipulation der Ansicht des Anzeigebereichs und der grafischen Objekte wie Editierfunktionen, Zoom oder die Gruppierung von Elementen.

4. **Property Inspector Panel:** Dieser Bereich gibt einen Überblick über die wichtigsten Eigenschaften des gerade im Fokus stehenden Objektes.

5. **SPARQL Syntax Viewer/Query Results Viewer:** Ein wichtiger Bereich von NIGHTLIGHT. Hier wird die über den grafischen Editor erzeugte SPARQL-Query in Textform angezeigt. Eine Bearbeitung der Query über die textuelle Darstellung ist nicht möglich. Dieser Bereich dient auch zur Anzeige der Resultate der erzeugten Query im XML-Format.

Abgrenzung zu anderen Tools

Neben NIGHTLIGHT gibt es noch eine Reihe andere Tools für die visuelle Erzeugung von SPARQL-Abfragen wie „SPARQLviz“[2] oder „iSPARQL“ (siehe Kiefer et al., 2007). iSPARQL bietet einen ähnlichen Funktionalitätsumfang wie NIGHTLIGHT, weist aber nach Smart et al. (2008) einige Unterschiede auf:

- Der Ontologie-Browser von NIGHTLIGHT bietet mehr Informationen über die verwendeten Eigenschaften und eine hierarchische Übersicht über die Klassenstruktur, während iSPARQL die Ontologielemente in Konzepte und Eigenschaften gruppiert.
- Die grafische Darstellung der SPARQL-Elemente unterscheidet sich grundlegend in ihrem Formalismus.
- Die Querygenerierung in NIGHTLIGHT ist dynamischer als in iSPARQL. So wird beispielsweise die textuelle Repräsentation der erzeugten Query mit jeder Veränderung der grafischen Repräsentation sofort angepasst.
- NIGHTLIGHT ist eine reine Javascript-Applikation deren Look-and-Feel rein über Cascading Style Sheets (CSS) definiert wird, womit Anpassungen der Applikation sehr leicht möglich sind.

3.2.2 GoRelations

Neben den oben beschriebenen Systemen, bei denen eine SPARQL-Query mit Hilfe einer VQL generiert werden kann gibt es noch andere Möglichkeiten der Erstellung von Abfragen. Mit „GoRelations (Graph Of Relations)“ ist es nach Han et al. (2012) möglich, eine Abfragen über eine Frage oder eine Beschreibung eines Problems generieren zu lassen und so eine Antwort zu erhalten. Das Konzept ist als „Natural Language Interface“ (NLI) bekannt, und wird auch im Berereich des Semantic Web angewandt. Ein Beispiel ist das System „PowerAqua“ (siehe Lopez et al., 2011). Das Problem bei diesen Systemen liegt im Verständnis von komplexen Fragstrukturen in natürlicher Sprache. GoRelations verbindet das Konzept der NLIs mit einem grafischen Ansatz. Der User erstellt seine Frage über ein „Semantic Graph Interface“ (SGI), mit dem es auf einfachen und intuitiven Weg möglich ist, auch komplexe Fragestellungen zu modellieren. Ein Mappingprozess konvertiert den so entstandenen „Semantic Graph“ (SG) in eine SPARQL-Abfrage die an einen Endpoint weitergereicht wird.

[2] `http://sparqlviz.sourceforge.net/`

Das Semantic Graph Interface (SGI)

Han et al. (2012) beschreibt einen „Semantic Graph“ als eine grafische Repräsentation einer Frage oder einer Beschreibung. Er besteht aus Knoten und Kanten. Knoten repräsentieren Entitäten und die Kanten zwischen den Knoten stellen binäre Verknüpfungen der Entitäten dar. In Abbildung 3.6 ist ein SG mit den Entitäten „Place “, „Person“ und „Book“ sowie den Relationen „born in“ und „author“ dargestellt. Entitäten die im Resultat berücksichtigt werden sollen werden mit einem „?“ gekennzeichnet, andere mit einem „*“. Die Bezeichnungen der Entitäten und Verknüpfungen können vom User frei gewählt werden. Die einzige Einschränkung ist mit der erlaubten Anzahl der Wörter für Entitäten und Verknüpfungen gegeben. Entitäten werden in der Regel mit einzelnen oder mehreren Hauptwörtern beschrieben. Hier gilt eine maximale Anzahl von zwei Wörtern. Verknüpfungen werden mit Verben, Präpositionen, Hauptwörtern oder Phrasen beschrieben. Hier gilt eine Beschränkung von maximal drei Wörtern. GoRelations wurde speziell für DBpedia entwickelt und die genannten Einschränkungen beziehen sich auf die Eigenschaften der DBpedia-Ontonlogy. Außerdem wird der User durch die Einschränkung zu einer Vereinfachung seiner Fragestellung gezwungen.

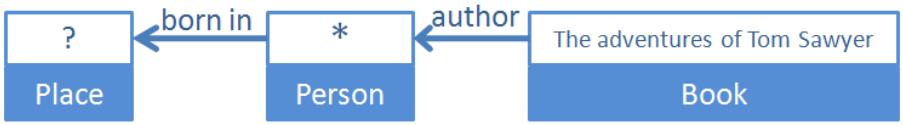

Abbildung 3.6: Beispiel eines Semantic Graph in *GoRelations*

Übersetzung des SG in eine SPARQL-Query

Die Übersetzung des SG in eine Abfrage erfolgt laut Han et al. (2012) in mehreren Schritten. Der erste Schritt ist der Mappingprozess, gefolgt von einer Disambiguation in Schritt 2 und einer Verfeinerung als dritten Schritt. In diesem Prozess wird versucht, die vom User gewählten Wörter und Phrasen für die Beschreibung der Entitäten und Verknüpfungen im SG auf die Klassen und Eigenschaften der DBpedia-Ontologie umzulegen.

1. **Schritt 1: Finde semantisch ähnliche Begriffe in der Ontologie.**

   Im ersten Schritt wird für jedem im SG beschriebene Konzept oder Beziehung eine Liste von k passenden Klassen oder Eigenschaften der Ontologie hinzugefügt, die die höchste semantische Ähnlichkeit aufweisen. Diese Klassen und Eigenschaften werden als „Kandidaten“ bezeichnet. Abbildung 3.7 zeigt den SG aus Abbildung 3.6 mit zugehörigen Kandidatenlisten.

   Die semantische Ähnlichkeit wird folgendermaßen bestimmt: Mit der Annahme, dass die Semantik einer Phrase aus der Zusammensetzung der Wörter der Phrase besteht, wird die semantische Ähnlichkeit zweier Phrasen über die Ähnlichkeit ihrer Wörter bestimmt. Dabei werden laut Han et al. (2012) Wortpaare aus Wörtern der Phrasen gebildet, sodass sich die maximale Summe der Wortähnlichkeit der einzelnen Paare ergibt. Diese maximale Summe wird in weiterer Folge über die Anzahl der Wörter der längeren Phrase normiert. Das Resultat ist die Ähnlichkeit der zwei Phrasen.

2. **Schritt 2: Disambiguation.**

   Aus jeder Kombination der für die Konzepte und Eigenschaften erhaltenen Kandidatenlisten aus Schritt 1 ergibt sich nun eine mögliche Interpretation der Userabfrage im Kontext der Ontologie. Der zweite Schritt dient dem Ausfiltern der sinnvollen Interpretationen. Ein intuitiver Ansatz ist ein Vergleich der Relationen aus dem SG mit den über die Kombination von Kandidaten enstehenden Relationen. Die Relation aus dem SG ist eine vom User generierte Assoziation zwischen zwei Konzepten. In Abbildung 3.7 werden die Entitäten „author“ und „book“ mit der Relation „wrote“ verknüpft. Der Grad der Sinnhaftigkeit dieser Verknüpfung spiegelt sich nun in der Ontologie wider, in der die Eigenschaft „wrote“ eng mit den beiden Konzepten „author“ und „book“ verbunden sein sollte. Für eine genaue Beschreibung des Vorgangs wird auf Han et al. (2012) verwiesen.

3. **Schritt 3: Verfeinerung**

   Über die oben genannten Schritte wird laut Han et al. (2012) meist eine gute Interpretation des SG über die Ontologie gefunden. Es kann allerdings vorkommen, dass Eigenschaften nicht gemapped werden können, oder dass eine falsche Eigenschaft im Kontext der Query gemapped wurde, weil eventuell eine andere, nicht so prominente Eigenschaft als die vom System gewählte, gemeint war. Da die Klassen richtig interpretiert wurden ist der Kontext der Query bekannt und es stehen alle möglichen Eigenschaften zur Verfügung die die beiden Klassen verbinden könnten. Falls eine Relation nicht eindeutig zugeordnet werden konnte wird aus den bekannten Relationen diejenige mit der größten semantischen Ähnlichkeit ausgewählt.

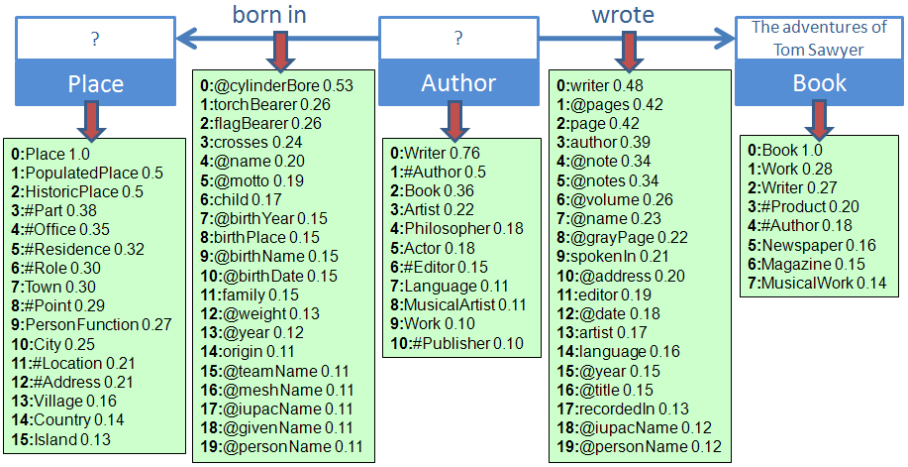

Abbildung 3.7: Mögliche Terme aus der Ontologie

Nachdem die vom User im SG definierten Entitäten und Verknüpfungen auf Klassen und Eigenschaften aus der Ontologie gemapped wurden, wird der SG in eine SPARQL-Query übersetzt, womit der Vorgang abgeschlossen ist.

3.2.3 Fazit

Systeme wie GoRelations gehen einen Schritt weiter als die in Abschnitt 3.1 und 3.2 beschriebenen Abfragesysteme in Form von Suchmaschinen und Querybuildern mit einer visuellen Abfragesprache (VQL). Sie wollen eine Unabhängigkeit des Users von der genauen Kenntnis der abzufragenden Wissensbasis und der darunterliegenden Datenstruktur erreichen. Der User stellt eine Frage, ohne über den Aufbau der Wissensbasis bescheid zu wissen. Die Umlegung erfolgt durch das System. Im Fall von GoRelations wird die Frage über den so genannten Semantic Graph ausgedrückt, was eine Umlegung auf die Graphstruktur von RDF erleichtert, aber vom User eine Konvertierung seiner ursprünglichen Frage in natürlicher Sprache in eine graphische Form verlangt.

Die in diesem Abschnitt beschriebenen graphischen Abfragesysteme verlangen vom User ein tieferes Verständnis über RDF und die Abfragesprache SPARQL und sind für User ohne Vorwissen nur sehr schwer nutzbar. Sie sind dafür ausgelegt eine kontrollierte Abfrage über grafische Komponenten zu generieren. Die generierten Abfragen sind in manchen Systemen speicherbar und können zu einem späteren Zeitpunkt weiterbearbeitet werden. Die Verknüpfung mehrerer verschiedener Repositories ist ebenfalls nur in manchen Systemen möglich.

3.3 Ansätze mit maschinellem Lernen

In Abschnitt 3.2.2 wurde mit GoRelations ein System vorgestellt, das mit vom User definierten Fragestellungen arbeitet, und seine Abfragen nicht über das Zusammenstellen von Eigenschaften und Resourcen definiert. Allerdings sind diese Fragen nicht in natürlicher Sprache zu stellen, sondern über den Umweg einer grafischen Repräsentation. Der Einsatz von KI-Techniken wie „Natural Language Processing“ (NLP) in „Question Answering“ (QA) Systemen, erlaubt es dem User laut Lehmann and Bühmann (2011), Anfragen in natürlicher Sprache an ein System zu stellen. Beispiele für solche Systeme im Kontext des Semantic Web sind „Ginseng“ (siehe Bernstein et al., 2006), „NLP-Reduce“ (siehe Kaufmann et al., 2007) oder „PowerAqua“ (siehe Lopez et al., 2011).

3.3.1 AutoSPARQL

AutoSPARQL benutzt laut Lehmann and Bühmann (2011) den Ansatz des „supervised machine learning“, also Überwachtes Lernen, um SPARQL-Abfragen auf der Basis von positiven und negativen Beispielen zu generieren.

AutoSPARQL Workflow

Nach Lehmann and Bühmann (2011) beginnt der User seine Abfrage entweder über eine Frage in natürlicher Sprache, oder über die direkte Suche einer bestimmten Resource wie beispielsweise „Berlin “. Er erhält daraufhin erste mögliche Resultate. Hier wählt der User ein zu seiner Anfrage passendes Resultat als erstes positives Feedback aus. Der nächste Schritt besteht in der Beantwortung einiger Entscheidungsfragen über einige Resourcen die der User entweder mit „Ja“ oder mit “Nein“ beantwortet, um anzuzeigen, dass die vorgeschlagenen Resourcen zu der erwarteten Resultatsmenge gehören oder nicht. Mit Hilfe

dieser Feedbackinformationen ist das System in der Lage schrittweise zu bestimmen, an welcher Abfrage der User interessiert sein könnte.

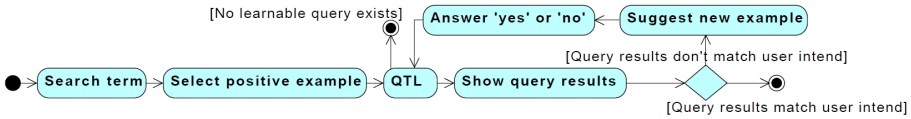

Abbildung 3.8: Der Workflow von AutoSPARQL (siehe Lehmann and Bühmann, 2011)

In Abbildung 3.8 wird der oben beschriebene Vorgang noch einmal dargestellt. Zentraler Bestandteil der Abfragegenerierung ist, wie in der Abbildung ersichtlich, der Knoten QTL. Der Knoten QTL steht für den „Query Tree Language“-Algorithmus und wird im nachfolgenden Abschnitt 3.3.1 erläutert.

Query Tree Language (QTL)

Als „Query Tree“ wird eine interne Datenstruktur des bei AutoSPARQL verwendeten QTL-Algorithmus bezeichnet. Jeder Query-Tree repräsentiert eine SPARQL-Abfrage. (Es können aber nicht alle SPARQL-Abfragen über Query-Trees dargestellt werden.) Das Resultat eines Query-Trees ist immer eine einzelne Spalte die über den Root-Knoten des QT als Variable „?“ definiert wird. Jede Kante des QT repräsentiert ein SPARQL Tripel-Pattern der Abfrage (siehe auch Abschnitt 2.5.2). Ein QT kann aus weiteren Subbäumen bestehen die wiederum Variablen der Query repräsentieren. In Abbildung 3.9 wird ein beispielhafter QT dargestellt und Listing 3.1 zeigt die mit ihm korrespondierende SPARQL-Abfrage (siehe Lehmann and Bühmann, 2011).

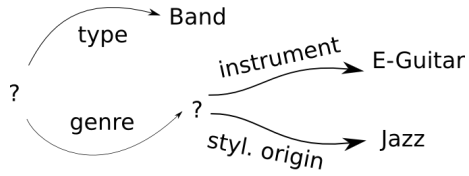

Abbildung 3.9: AutoSPARQL Query Tree (siehe Lehmann and Bühmann, 2011)

Gesucht werden Bands deren Stilrichtung eine Mischung aus Jazz und elektrische Gitarren darstellt.

```
SELECT ?x0 WHERE {
?x0 rdf:type dbo:Band.
?x0 dbo:genre ?x1.
?x1 dbo:instrument dbp:Electric_guitar.
?x1 dbo:stylisticOrigin dbp:Jazz.
}
```

Listing 3.1: SPARQL Query zu Query Tree aus Abbildung 3.9

Query Trees agieren nach Lehmann and Bühmann (2011) also als Bindeglied zwischen der Beschreibung einer Resource in einem RDF Grapen und der SPARQL-Abfrage die

diese Resource in ihrem Resultat enthält. Mit der Hilfe von Query Trees wurde mit AutoSPARQL ein sehr effizienter Lernalgorithmus für SPARQL-Queries entwickelt, der QTL-Algorithmus.

QTL-Algorithmus

Laut Lehmann and Bühmann (2011) nimmt der Algorithmus die positiven und negativen Feedbacks (in Form von RDF Resourcen) des Users als Input. In einem ersten Schritt werden alle Feedback-Beispiele in Query Trees umgewandelt. Für diesen Mappingvorgang wird noch eine maximale Rekursionstiefe als Parameter benötigt. Die Rekursionstiefe beeinflusst die Größe des generierten Baumes, da sie die maximale Tiefe des Baumes und damit auch die maximale Tiefe der im Anschluss generierten SPARQL-Query definiert. Im zweiten Schritt wird der kleinste gemeinsame Teilaspekt der positiven Beispiele über ihre Query-Trees ermittelt und in einem neuen QT gespeichert. Dies geschieht durch den Vergleich je zwei positiver QT. Es werden die Root-Labels der beiden Bäume verglichen. Sind sie ident bleibt ihr Label als Variable erhalten, ansonsten wird eine neue Variable *?* eingeführt. Danach werden die aus den jeweiligen Wuzelknoten der Bäume ausgehenden Kanten, also die gesuchten Eigenschaften verglichen. Übereinstimmende Kanten (Eigenschaften) werden übernommen. Danach wird für jede Kombination der Kanten der übernommenen Knoten der Vorgang rekursiv wiederholt. Wenn der so erhaltene QT negative Resourcen enthält wird die Query verworfen und der Vorgang abgebrochen da keine lernbare Query existiert. Ist dies nicht der Fall wird die so gelernte Query als Basis für neue Beispiele herangezogen, die vom User bewertet, und als neues Feedback für eine weitere Verfeinerung verwendet werden können. (Siehe auch Abbildung 3.8.)

3.3.2 Fazit

AutoSPARQL bietet nach einer erfolgten Question-Answer Session einige Möglichkeiten zur Verfeinerung der resultierenden Abfrage. So können die gewünschten Eigenschaften sowie Sortierungen oder die gewünschte Sprache des Resultats gewählt werden. Die Abfragen sind speicherbar und es wird ein URL bereitgestellt, bei dessen Aufruf die Ergebnisse der Query zurückgeliefert werden um sie beispielsweise in eine eigene Homepage einbinden zu können. Hierfür bietet AutoSPARQL einen Caching-Mechanismus mit einem definierbaren Timeout um immer relativ aktuelle Ergebnisse zu erhalten (siehe Lehmann and Bühmann, 2011).

3.4 Andere Ansätze

Neben den browserorientierten Ansätzen, den grafischen Repräsentationen von Abfragen und lernbasierten Ansätzen zur Abfrage und Erzeugung von SPARQL-Queries gibt es noch andere Möglichkeiten um den Benutzern den Umgang mit SPARQL zu erleichtern. Als weiteres Beispiel ist das System „SPARQL Assist“ (siehe McCarthy et al., 2012) anzuführen.

3.4.1 SPARQL Assist

„SPARQL Assist“ ist eine Webapplikation die einen Assistenten zur Erstellung von SPARQL-Abfragen mit Hilfe einer kontextsensitiver Autovervollständigung anbietet. Die Vervoll-

ständigung erfolgt auf Basis der Informationen aus der Ontologie eines Repositories. Im Speziellen werden nach McCarthy et al. (2012) die Labelinformationen der Ontologie extrahiert, und für die Autovervollständigung verwendet. Über das *xml:lang*-Attribut werden überdies verschiedene Sprachen unterstützt.

3.4.2 Fazit

Mit SPARQL Assist erfolgt die Unterstützung des Users über eine andere Philosophie als bei den vorher genannten Systemen. Die bisher beschriebenen Systeme verfolgen das Ziel dem User die Last der Abfragegenerierung komplett abzunehmen. Im Idealfall bekommt der User die Query überhaupt nicht zu Gesicht. SPARQL Assist versucht im Gegensatz dazu, den User bei der händischen Erstellung der Abfrage zu assistieren. Hintergrund dabei ist die Tatsache, dass für die händische Erzeugung einer SPARQL-Query das darunterliegende Vokabular bekannt sein muss, was bei umfangreichen und komplexen Vokabularen eine Hürde darstellen kann. Ein weiteres Problem ist nach McCarthy et al. (2012) dadurch gegeben, dass viele Vokabulare darauf übergegangen sind semantikfreie URIs für Klassen und Eigenschaften zu verwenden, was die Konstruktion einer händischen Query extrem erschwert. So ist beispielsweise die Bedeutung der Eigenschaften „rdf:type“ oder „dc:title“ recht klar, während der Sinn der Eigenschaft „sio:SIO_010302“ nicht sofort erkennbar ist.

3.5 Zusammenfassung

Die in diesem Abschnitt beschriebenen unterstützenden Systeme zur Erstellung von SPARQL-Abfragen vereinen einige wichige Eigenschaften die der User von einem Abfrageassistenten erwarten kann.

Das System muss den User beim Auffinden der benötigten Informationen der dem Repository zugrundeliegenden Datenstruktur unterstützen. Dies geschieht entweder durch grafische Repräsentationen der verwendeten Ontologien und Vokabularen, Benutzerelemente die ein Browsen durch die wählbaren Elemente ermöglichen, oder durch indizierte Datenstände und eine mit diesen Daten realisierte Autovervollständigung.

Die Generierung der Abfrage selbst erfolgt bei den vorgestellten Systemen entweder über eine grafische Abfragesprache, mit der der User die gewünschte Query selbstständig erzeugt, oder über einen geführten Suchvorgang bei dem die Abfrage über das System nach den Angaben des Benutzers und dessen Feedback, zum Beispiel über Frage-Antwort Systeme, über einen iterativen Vorgang generiert und verfeinert wird. Letzterer Ansatz hat den Vorteil, dass der User mit der eigentlichen Abfrage nicht in Berührung kommen muss und auch ohne Kenntnisse der Abfragesprache SPARQL gezielte Informationen aus Repositories extrahieren kann.

Kapitel 4

Design

In diesem Kapitel werden die für SPARQL-Wizard erforderlichen Anforderungen definiert und beschrieben. In weiterer Folge werden die für die Umsetzung des SPARQL-Wizard verwendeten Technologien erläutert. Den Abschluss dieses Kapitels bildet eine Abhandlung der für die Umsetzung des Wizards erdachten Lösungsansätze.

4.1 Anforderungen

Durch die Analyse der in Abschnitt 3 besprochenenen vorhandenen Systeme haben sich einige Anforderungen an einen Abfrageassistenten, sowohl auf Ebene des Benutzers als auch auf Systemebene, herauskristallisiert.

4.1.1 Benutzeranforderungen

Den Benutzeranforderungen kommt ein hoher Stellenwert zu. Sie sind maßgeblich für die Umsetzung des in dieser Arbeit beschriebenen Abfrageassistenten.

Funktionale Anforderungen

1. **Entitity Search:** Der User soll nach einer einzelnen Entität suchen können. Der User gibt ein oder mehrere Schlüsselwörter in ein Suchfeld ein und erhält eine Liste mit möglichen Entitäten. Er wählt nun die gewünschte Entität aus der Liste aus.

2. **Entities über Kategorie:** Der User soll über eine Entität eine der Entität zugeordneten Kategorie auswählen können. Der User gibt ein oder mehrere Schlüsselwörter in ein Suchfeld ein, und erhält eine Liste mit möglichen Entitäten. Er wählt die gewünschte Entität und erhält eine Liste mit Kategorien zu denen diese Entität zugeordnet sind. Er wählt die gewünschte Kategorie.

3. **Category Search:** Der User soll nach einer bestimmten Kategorie/Überbegriff suchen können. Der User gibt ein oder mehrere Schlüsselwörter in das Suchfeld ein und erhält eine Liste mit möglichen Kategorien die dem Schlüsselwort entsprechen. Er wählt die gewünschte Kategorie aus.

4. **Hinzufügen/Entfernen von Prädikaten:** Der User soll einzelne Prädikate zu seiner Ergebnistabelle hinzufügen oder entfernen können.
5. **Entfernen von Datensätzen:** Der User soll einzelne Datensätze aus seiner Ergebnistabelle entfernen können.
6. **Hinzufügen von Datensätzen:** Der User soll einzelne Datensätze zu seiner Ergebnistabelle hinzufügen können.
7. **Sortieren von Prädikaten:** Der User soll Reihenfolge der Prädikate in seiner Ergebnistabelle beliebig abändern können.
8. **Sortieren nach Prädikaten:** Der User soll seine Ergebnistabelle nach einzelnen Prädikaten sortieren können.
9. **Filterfunktion:** Der User soll seine Ergebnisse der Kategoriesuche über Filterkriterien der Prädikate eingrenzen können.
10. **Speicherfunktion:** Der User soll seine Ergebnisse für die spätere Verwendung speichern können. Dabei kann der User entweder einzelne Datensätze aus seiner Ergebnistabelle für die Sicherung auswählen, oder er kann die gesamte Ergebnistabelle speichern. Die Speicherung erfolgt in sogenannten „Collections“.
11. **Laden von Collections:** Der User soll einmal gespeicherte Collections zur Ansicht der Ergebnisse oder Weiterverarbeitung laden können. Hierfür wählt er in seinem Profil die gewünschte Collection.
12. **Löschen von Collections:** Der User soll einzelne Collections aus seinem Profil löschen können.

Nichtfunktionale Anforderungen

1. **Benutzbarkeit:** Der Wizard soll ohne jegliches Vorwissen über RDF und die SPARQL-Abfragesprache bedienbar sein. Der User soll in der Lage sein mit durchschnittlichen Erfahrungen mit dem Internet das System zu bedienen und Abfragen zu generieren.
2. **Plattformunabhängigkeit:** Das System soll auf allen gängigen Plattformen lauffähig sein.
3. **Browserkompatibilität:** Das System soll mit allen gängigen Browsern lauffähig sein.
4. **Erweiterbarkeit:** Das System soll im Bezug auf die unterstützten Repositories einfach erweiterbar sein. Im Idealfall über einen weitgehend automatisierten Mechanismus.

4.2 Technologien

Für die technische Umsetzung des Query-Wizards wurden auch im Hinblick auf die im vorigen Abschnitt besprochenen Anforderungen an das System Entscheidungen bezüglich der für die Implementierung verwendeten Technologien getroffen. Das System soll plattformunabhängig und webbasierend sein. Daher fällt die Wahl auf aktuelle Webtechnologien sowohl für die Client-, als auch die Serverseite.

Serverseitig wurde die Programmiersprache Python, dass sich laut Ernesti and Kaiser (2009) auf Grund seiner Flexibilität besonders für die Entwicklung als serverseitige Programmiersprache und „Rapid Prototyping“ eignet, und die Open-Source-Datenbank MySQL gewählt. Ein Grund für die Auswahl von Python als serverseitige Entwicklungssprache ist *Django*[1], ein Framework zur Erstellung von Web-Applikationen. Django bietet ein Grundgerüst und eine Reihe von Komponenten für die Entwicklung von Modulen auf Basis einer Model-View-Controller Architektur (MVC, siehe auch Abschnitt 4.2).

Mit Hilfe von Django wurde die Plattform „Linked Data For You“ (LD4U) aufgesetzt, und der in dieser Arbeit beschriebene SPARQL-Query Wizard als Django-Modul implementiert. Mehr Informationen über die Implementierung von Modulen im Allgemeinen und dem Query-Wizard Modul im Speziellen finden sich in Abschnitt 4.3. Durch die Modularchitektur von Django kann die Plattform mit beliebiger Funktionalität und um beliebige Features erweitert werden, da neue Funktionalitäten einfach als neues Modul in die Plattform eingehängt werden können.

Das Django-Framework bietet folgende Grundfunktionalitäten die bei der Erstellung eines Moduls angewandt werden können:

- **Objektrelationale Abbildung der Datenbank**
  Mit Django erfolgt der Zugriff auf Datenbanken über einen objektorientierten Ansatz. Datensätze werden als Objekte erzeugt und verarbeitet. Siehe Abschnitt 4.3.1.
- **Integriertes Administrationsinterface**
  Django enthält eine integrierte Administrationsoberfläche zur Verwaltung von Benutzern, Gruppen und verschiedenen Berechtigungen die auf der LD4U Plattform zur Userverwaltung eingesetzt[2] wird.
- **URL Dispatcher**
  Das in einer django Applikation verwendete URL-Schema ist über das integrierte Modul „URLconf“ frei definierbar und ermöglicht ein Mapping von URLs auf Callback-Funktionen der jeweiligen Module und deren Funktionalitäten. Siehe auch Abschnitt 4.3.4.
- **Integriertes Templating-System**
  Django enthält ein eigenes Templating-System das mit einer eigenen Template-Sprache

[1] `https://www.djangoproject.com/`
[2] `https://docs.djangoproject.com/en/dev/topics/auth/`

für die Trennung von Design und Inhalt verwendet werden kann. Siehe auch Abschnitt 4.3.2.

- **Integriertes Caching-System**
  Zur Leistungssteigerung bietet Django unterschiedliche Cachingvarianten. Siehe Abschnitt 4.3.5

- **Unterstützung von Mehrsprachigkeit**
  Django unterstützt von Haus aus Mehrsprachigkeit in Applikationen[3].

Auf Clientseite kommt JavaScript zum Einsatz. Große Teile des Userinterface werden über JQery UI[4] generiert. JQuery UI ist eine Erweiterung der JQuery JavaScript Bibliothek und bietet eine Vielzahl an Interface-Elementen, Widgets und Effekten.

Technologie	Version	Anwendung
Python	2.7.2+	Serverseitig, LD4U Query Wizard Modul
Django	1.3.0	Serverseitig, LD4U Plattform
JQuery	1.6.2	Clientseitig, UI
JQuery-UI	1.8.18	Clientseitig, UI
MySQL	5.1.62	Serverseitig, Datenbank

Tabelle 4.1: Technologien im Überblick

4.3 Architektur

Wie im Abschnitt 4.2 beschrieben, wurde der im Rahmen dieser Arbeit enwickelte SPARQL-Wizard als Applikation für das Python Webframework Django konzipiert und umgesetzt. Daher entspricht die Architektur der einer standard Django-Applikation und beinhaltet einige Basiskomponenten. Django befolgt prinzipiell das MVC-Prinzip (Model View Controller), allerdings mit leichten Abweichungen. So ist laut djangobook (2009), in Django eigentlich von einem MVT-Prinzip zu sprechen. MVT steht in diesem Fall für Model, View und Template.

Diese Begriffe stellen die drei Kernebenen von Django dar und werden folgend kurz beschrieben[5]:

4.3.1 Model

Das Model dient zur Definition eines Datenmodells dessen Daten typischerweise in einer relationalen Datenbank wie beispielsweise MySQL gespeichert werden. In Django wird ein Model durch eine Python Klasse realisiert, die Variablen, aber auch Methoden, für einen bestimmten Typ von Daten definieren. Diese Klasse bildet ein Mapping auf eine Datenbanktabelle. Jedes Model repräsentiert somit eine Tabelle in einer Datenbank und jedes Attribut der Klasse entspricht einer Spalte der Tabelle. Django stellt der Applikation

[3] `https://docs.djangoproject.com/en/1.4/topics/i18n/`
[4] `http://jqueryui.com/`
[5] `http://jeffcroft.com/blog/2007/jan/11/django-and-mtv/`

nun eine automatisch generierte API für den Zugriff auf die Datenbank zur Verfügung, über die ein objektorientierter Zugriff auf die gespeicherten Daten möglich ist. [6]

Im Falle des Sparql Wizard wurden Informationen über gespeicherten Queries eines Users über ein Model definiert (siehe auch Abschnitt 5.4).

4.3.2 View

Die Ebene der View hat im Wesentlichen die Aufgabe, die für eine bestimmte Ansicht benötigten Daten aus dem Datenmodell bzw. der Datenbank zu extrahieren und an das enstprechende Template weiterzureichen. Die View stellt also die Daten, die in weiterer Folge im Template zur Darstellung einer bestimmten Information benötigt werden bereit. Die View im Sinne von Django hat noch nichts mit der Darstellung der Information zu tun. Dies unterscheidet Django von anderen MVC-Frameworks. Die Darstellung der Daten erfolgt in weiterer Folge rein über das Template.

Eine View in Django wird über eine Funktion definiert die einen Webrequest als Parameter erhält und eine Webresponse zurückgibt.[7]

4.3.3 Template

Ein Template ist im Grunde ein einfaches Html File mit einigen extra Features. Das Template erhält seinen Kontext in Form von Variablen, Daten und Dergleichen über die View-Ebene. Auf diese über die View zur Verfügung gestellten Daten kann mit einfachen Programmierkonstrukten wie If-Else-Konstrukte oder Schleifen in einer Darstellungslogik für die Repräsentation zugegriffen werden. Es gilt eine Trennung der reinen Darstellungslogik von der Businesslogik, die in der View angesiedelt ist, einzuhalten![8]

Abbildung 4.1 dient zur Veranschaulichung des Zusammenspiels der drei beschriebenen Kernebenen Model, Template und View. Zusätzlich sind noch zwei weitere Punkte angeführt. Das Caching-Framework und der URL-Dispatcher.

4.3.4 URL-Dispatcher

Der URL Dispatcher dient dazu, ein Mapping zwischen einer angeforderten URL und einer bestimmten View herzustellen. Dies geschieht über die Definition von Pattern und regulären Ausdrücken, die bei einem Match die entsprechende View-Funktionalität aufrufen.[9]

4.3.5 Caching-Framework

Bei aktivierten Caching werden die Outputs angeforderter View-Funktionen für einen bestimmten Zeitraum zwischengespeichert und bei einem erneuten Aufruf wird anstatt eines neuerlichen Aufrufes der Funktion der gecachte Content herangezogen. Django unterstützt verschiedene Caching-Mechanismen wie beispielsweise „Memory caching“, „Database caching“ oder „Filesystem caching“[10].

[6] `https://docs.djangoproject.com/en/dev/topics/db/models/`
[7] `https://docs.djangoproject.com/en/dev/topics/http/views/`
[8] `https://docs.djangoproject.com/en/dev/ref/templates/`
[9] `https://docs.djangoproject.com/en/1.4/topics/http/urls/`
[10] `https://docs.djangoproject.com/en/dev/topics/cache/?from=olddocs`

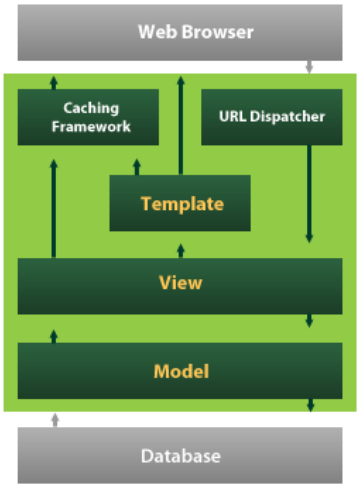

Abbildung 4.1: Django Architektur (`http://mytardis.readthedocs.org`)

4.4 Umsetzungsstrategien/Ansätze

Für die Umsetzung eines Prototypen des SPARQL-Wizard der den definierten Anwendungsfällen mit oben beschriebenen Technologien gerecht wird, wurden verschiedene Ideen bzw. Ansätze zur Umsetzung herangezogen und wieder verworfen, bis sich die entgültige, im weiteren Verlauf dieser Arbeit beschriebene Strategie entwickelt hat.

Der Ablauf der Generierung einer Anfrage durch den User erfolgt schrittweise in einem interativen Verfahren. Der User beginnt mit einer allgemeinen Definition seiner Anfrage und verfeinert diese in weiterer Folge bis die Ergebnismenge seinen Wünschen entspricht. Für die Umsetzung bedarf es daher einen geeigneten Einstiegspunkt für den Benutzer. Wie in den Anforderungen an den in dieser Arbeit beschriebenen SPARQL-Wizard in Abschnitt 4.1 beschrieben, soll der User die Möglichkeit einer entitätsbasierten Suche (Entity-Search) und die Möglichkeit einer entitätsmengenbasierten Suche (Suche über Kategorien), um die als Basis für die Abfrage benötigten Resourcen zu erhalten, bekommen

Hier haben sich, auch auf Grund der Erfahrungen mit den in Abschnitt 3 dargestellten Systemen zwei Ansätze angeboten, die sich schließlich zu einem dritten Ansatz weiterentwickelten.

4.4.1 Ansatz 1: Ontology

Der erste Ansatz arbeitet über die dem Repository zugrundeliegenden Ontologie und bietet dem User als Einstiegspunkt eine Darstellung der in der Ontologie enthaltenen Klassen, deren Subklassen und Eigenschaften und ermöglicht dem User somit eine Vorauswahl der gewünschten Kategorie, gefolgt von einer Auswahl der in dieser Kategorie möglichen Eigenschaften oder Prädikate. Dieser Ansatz entspricht im Wesentlichen der Herangehensweise

von visuellen Abfragetools wie NITELIGHT (siehe Abschnitt 3.2.1), die zwar Hilfestellung im Bezug auf die möglichen Klassen und Eigenschaften, aber keinerlei Unterstützung bei der Auswahl der eigentlichen Entitäten bzw. Resourcen geben. Der User muss sich von vorneherein genau im Klaren sein nach was er sucht und in welcher Kategorie er das gewünschte Ergebnis finden wird. Der Wizard begibt sich damit auch in eine Abhängigkeit im Bezug auf die Ontologie. Ist die für den Wizard als Basis dienende Ontologie unvollständig oder nicht aktuell können die Informationen nicht abgefragt werden. Ein Vorteil dieses Ansatzes ist der geringe Resourcenbedarf. Die einzigen Daten die benötigt werden ist das abzufragende Repository und dessen Ontologiedefinition. Dieser Vorteil ist aber auch gleichzeitig wieder ein Schwachpunkt dieses Ansatzes. Es gibt keinerlei lokale Daten und jede Zusatzinformation muss über den SPARQL-Endpoint des Repository abgefragt werden, was mitunter zu sehr langen Reaktionszeiten und somit für den User zu unangenehmen Wartezeiten führt. Dieser Ansatz ist darüber hinaus nur für Kategoriebezogene Abfragen geeignet. Einzelne, dem User eventuell unbekannte Entitäten oder Resourcen abzufragen, ist auf Grund der unzureichenden Information, und ohne den Endpoint des Zielrepositories nicht unnötig zu belasten, nicht möglich.

4.4.2 Ansatz 2: Lokale Kopie des Repositories

Um die Nachteile des ersten Ansatzes im Bezug auf Performance und Entitätensuche auszugleichen, basiert der zweite Ansatz auf einem lokalem Dump des Zielrepositories. Im Kontext dieser Arbeit handelt es sich dabei um eine Kopie des aktuellen Dbpedia Datensatzes (siehe auch Abschnitt 2.7.2). Der Vorteil einer lokalen Kopie des gesamten Repositories liegt in der höheren Verarbeitungsgeschwindigkeit da man keinen Limitierungen des jeweiligen SPARQL-Endpoints unterliegt. Es können beliebige Daten in für den User angemessener Zeit über Indexe abgefragt werden, was eine Unterstützung des Users mit einer Entitätensuche wie in Semantic-Web Suchmaschinen wie in Abschnitt 2.6.3 gestattet. Es bedarf aber einer guten Konfiguration des lokalen Systems, um die optimale Performance zu erreichen, was gerade bei einem sehr großen Dump wie der des Datenstandes von Dbpedia offensichtlich ist. Offensichtlich ist auch der Nachteil eines solchen Vorgehens. Einerseits benötigt man Resourcen für das lokale Repository. Andererseits widerspricht der Ansatz dem Grundgedanken von Linked Data (siehe auch Abschnitt 2.6.3). Es wird ein Snapshot eines schon vorhandenen Repositories erzeugt. Damit sind diese Daten erst mit der nächsten Aktualisierung des lokalen RDF-Repositories am neuesten Stand. Auch ist die Erweiterung des Systems auf mehrere Repositories problematisch weil von jedem unterstütztem Repository eine Kopie in das lokale RDf-Repository eingespielt werden muss.

Dieser Ansatz erweist sich also auch nicht als geeignet um alle Anforderungen an den Query-Wizard zu erfüllen. Benötigt wird eine Kombination aus den beiden Ansätzen, die einerseits einen lokalen Datensatz für eine schnelle Suche nach Entitäten ermöglicht und andererseits Informationen über die Datenstruktur des Repositories, bzw. dessen Klassen, Relationen und Eigenschaften für eine entitätsmengen basierte Abfrage bereitstellt.

4.4.3 Ansatz 3: Label-Extraktion

Dieser dritte Ansatz basiert ebenfalls auf einem Dump des oder der betreffenden Repositories. Allerdings wird dieser nicht für eine Kopie des gesamten Repositories genutzt. Aus dem Dump werden in einem Vorverarbeitungsschritt automatisiert Informationen extrahiert und in einer lokalen Datenbank gespeichert. Die extrahierten Informationen beinhalten

- **Repository-Information**
  Hier werden Informationen wie die URI des Repositories, ein Name, Beschreibungstext und das letzte Aktualisierungsdatum gespeichert. Alle Resourcen werden mit dem jeweiligen Repository verknüpft und sind so dem jeweiligen Datensatz zuzuordnen.

- **Resourcen**
  Von jeder Resource wird die URI gespeichert. Die URI verändert sich in der Regel nicht und identifiziert eine Resource eindeutig. Sie ist einer der wichtigsten Bestandteile für die Abfragegenerierung. Alle weiteren Daten sind mit der jeweiligen URI verknüpft.

- **Label-Informationen**
  Die Labelinformation einer URI sind die einzigen tatsächlichen Daten die aus einem Repository-Dump extrahiert werden. Sie werden für den Suchprozess indiziert und unterstützen den User bei der Erstellung seiner Abfrage.

- **Typ-Informationen**
  Es wird jeder Resource ein Typ zugeordnet. Auch hier werden nur die URIs der jeweiligen Typen gespeichert.

- **Prädikat-Information**
  Prädikate werden nach dem gleichen Schema wie Typen als URIs und mit Zuordnung zu einer Resource und Typ extrahiert und gespeichert.

- **Statistische Daten**
  Zusätzlich zu den URIs werden noch statistische Informationen über die extrahierten Daten erhoben. Es handelt sich dabei um Häufigkeitswerte bezüglich zugeordneter Typen und Prädikate, mit deren Hilfe der Suchprozess optimiert werden kann.

Eine genaue Beschreibung der Extraktion dieser Daten ist in Abschnitt 5.2 zu finden.

Die Informationen aus den Dumps dienen als Grundlage für die Erstellung der Abfrage nach den vom User gesuchten Entities. Über die Typ- und Prädikatinformationen lässt sich zu jeder Resource dessen Klasse, Elternklassen und Relationen ableiten bzw. umgekehrt die Resourcen die zu einer bestimmten Klasse gehören, über den lokalen Datenstand abfragen, ohne auch nur eine Abfrage an das Remote-Repository stellen zu müssen. Die Labelinformationen dienen zur Suche nach einzelnen Entitäten. Der User kann diese einfach über ein Suchfeld wie er es von einer Suchmaschine gewohnt ist über deren Labels suchen. Er benötigt auf diese Weise keinerlei Information über die Typzugehörigkeit der benötigten

Resource. Ebenfalls ermöglicht der indizierte lokale Datenstand eine Unterstützung des Users bei der Suche nach Entitäten über eine performante Autovervollständigung (siehe Abschnitt 5.3.1).

Eine Umsetzung mit diesem Ansatz bringt einige Vorteile. Man benötigt keine vollständige lokale Kopie der gewünschten Repositories und umgeht damit das Problem redundanter und nicht aktueller Daten. Da nur URIs gespeichert werden, die für die Abfragegenerierung benötigt werden und die Abfragen gegen den Endpoint des Originalrepositories gerichtet sind, sind die so generierten Daten immer auf dem neuesten Stand. Für den eigentlichen Wizard werden keine zusätzlichen Anfragen an das Repository benötigt, womit ein sehr reaktives System geschaffen wird.

Mit der konkrete Umsetzung des SPARQL-Wizard mit Hilfe dieses Ansatzes beschäftigt sich der auf dieses Kapitel folgende Abschnitt 5.

Kapitel 5

Implementierung

Dieses Kapitel befasst sich mit der Umsetzung eines Prototypen des in dieser Arbeit besprochenen SPARQL-Wizard. Es folgt ein Überblick, an dessen Anschluss sich eine detailierte Beschreibung der Umsetzung der einzelnen Anwendungsfälle einer Suche und der damit verbundenen Abfragegenerierung befindet.

5.1 Überblick

In Abschnitt 4.4.3 wurde der Ansatz der Labelextraktion als finaler Ansatz für die Umsetzung des SPARQL-Wizard vorgestellt. Diesem Ansatz ist zu entnehmen, dass ein Vorverarbeitungsschritt in Form einer Datenextraktion aus einem beliebigen RDF-Dump benötigt wird, um über die Daten des Repositories über den Wizard zugänglich zu machen. Dieser Vorgang wird in Abschnitt 5.2, „Pre-Processing “ beschrieben. Ausgehend von diesen nun lokal vorhandenen Daten kann ein User ein Suchinterface in Form eines einfachen Suchfeldes mit einer unterstützenden Autovervollständigung als Ausgangsbasis für die Generierung seiner Abfrage nutzen. Der Vorgang dieser initialen Suche wird in Abschnitt 5.3.1 besprochen. Je nachdem, was für eine Art von Ergebnis der User in seiner Abfrage benötigt kann er nun einen der zwei möglichen Modi einsetzen um seine Abfrage zu verfeinern. Die beiden Modi werden in den Abschnitten 5.3.2, „Entity-Search“ und 5.3.3, „Faceted-Search“ beschrieben. Einmal angefangene Anfragen können vom User zu einem späteren Zeitpunkt wiederaufgenommen werden. Dieser Vorgang wird in Abschnitt 5.4, „Speichern und Laden von Abfragen“ behandelt.

5.2 Pre-Processing

Als Basis für den in dieser Arbeit beschriebenen SPARQL-Wizard dient eine MySQL Datenbank in der Informationen zu den zu durchsuchenden Repositories gespeichert sind. Diese Datenbank muss in einem vorgelagerten Schritt befüllt werden. Es wurde ein Extraktionsskript entwickelt, das aus bestehenden RDF-Dumps der gewünschten Repositories Informationen die für den Wizard benötigt werden in die Datenbank integriert.

5.2.1 Datenbank

Die Datenbank enthält Informationen über die unterstützten Repositories in Form von Resourcen und Labels. Es werden Resourcen (URIs) der Repositories, Subjekte und der zugehörigen Prädikate gespeichert und in Relation zueinander gebracht, womit eine eindeutige Zuordnung der Resource ermöglicht wird. Neben den Resourcen werden Labelinformationen gespeichert die für den Suchprozess des Wizards unumgänglich sind. Die Struktur der verwendeten MySQL Datenbank ist in Abbildung 5.1 ersichtlich.

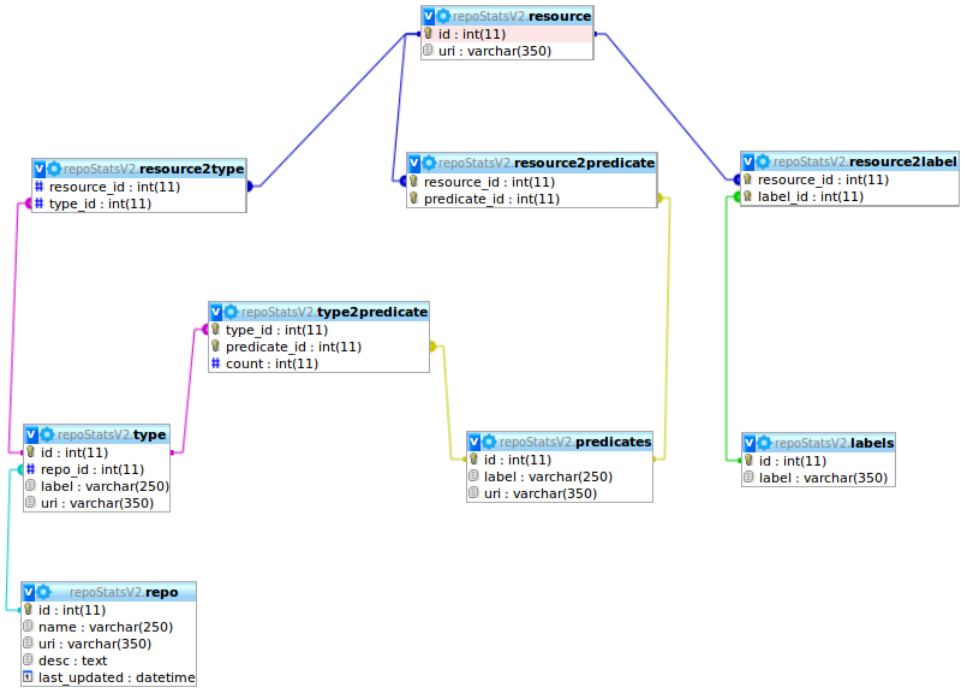

Abbildung 5.1: Datenbankstruktur

5.2.2 Repo-Crawler

Um die in Abbildung 5.1 dargestellte Datenbankstruktur zu befüllen bedarf es eines automatisierten Mechanismus um beliebige RDF-Repositories in den Wizard einfügen zu können, und deren Inhalte somit über das Suchinterface erreichbar zu machen.

RDF-Dumps

Um einen RDF-Datenstand in den Wizard einzufügen wird ein Dump der gewünschten Daten oder des gesamten Repositories im N-Triple Format benötigt. Dieses Format stellt eine Liste von RDF-Statements in der Form Subject, Prädikat, Objekt Zeile für Zeile in

Plaintext dar[1]. Das oder die Dump-Files werden nun einem Python-Skript übergeben. Das Skript arbeitet die Files zeilenweise ab und bearbeitet somit jeden Datensatz.

In Abbildung 5.2 wird ein Überblick über den Ablauf der Label- und Resourcen-Extraktion in Form eines Flussidagramms gegeben.

Funktionsweise

Ausgehend von der Annahme, dass der Dump eine Sortierung nach Resource bzw. Subjekt aufweist, erstellt der Crawler eine Datenstruktur in Form eines Dictionaries. Als Key wird der Reource-Url (Subjekt) des aktuellen Datensatzes verwendet. Dieser verweist auf ein weiteres Dictionary. Dieses Dictionary benutzt als Key die gefundenen Prädikate des aktuellen Datensatzes. Diese zeigen schließlich auf ein Array mit dem oder den Objekten die zu dem aktuellen Prädikat gefunden wurden. Somit sind alle Daten einer Resource in einer Art Baumstruktur abgelegt. Sind alle Daten der aktuellen Resource abgearbeitet wird die Datenstruktur in die Datenbank übertragen.

Die Datensätze sind beliebig erweiterbar. Bestehende Resourcen werden bei neuen Daten zu Prädikaten oder Objekten einfach erweitert. Somit ist eine Sortierung nach Resourcen bzw. Subjekten nicht zwingend notwendig. Dadurch ist es auch möglich die Bearbeitung eines Dumps auf mehrere Schritte aufzuteilen (zum Beispiel durch Splitten in mehrere Dateien) oder den Vorgang auf mehrere Prozesse zu verteilen.

Das Skript ist einfach in eine Serverumgebung zu integrieren. Für zukünftige Versionen ist ein webbasiertes Interface geplant um eine bestehende Wizard-Instanz über eine Administrationsoberfläche im Livebetrieb upzudaten oder neue Repositories zur Verfügung zu stellen. Näheres hierzu findet sich im Abschnitt 6.2.

5.3 Suche

5.3.1 Initialsuche

Der Initialsuche im Suchfeld kommt eine zentrale Bedeutung bei der Bedienung des Wizards zu. Es ist das Element, mit dem der User als erstes in Berührung kommt und über das der User die Interaktion mit dem Wizard beginnt. Der User gibt je nach Intention einen beliebigen Begriff in die Textbox ein und erhält daraufhin, unterstützt durch eine Autovervollständigung, Vorschläge zu seiner Eingabe.

Ausgehend von seinem Suchbegriff werden ihm Vorschläge in drei Kategorien angeboten. Abbildung 5.3 zeigt das Suchfeld mit Ergebnislisten für den Suchbegriff „mount“. Es werden dem User nun Vorschläge für Ergebnisse in drei verschiedenen Ergebniskategorien vorgegeben aus denen er das für ihn passende Item wählen kann.

Die Möglichen Ergebniskategorien sind:

[1] `http://www.w3.org/2001/sw/RDFCore/ntriples/`

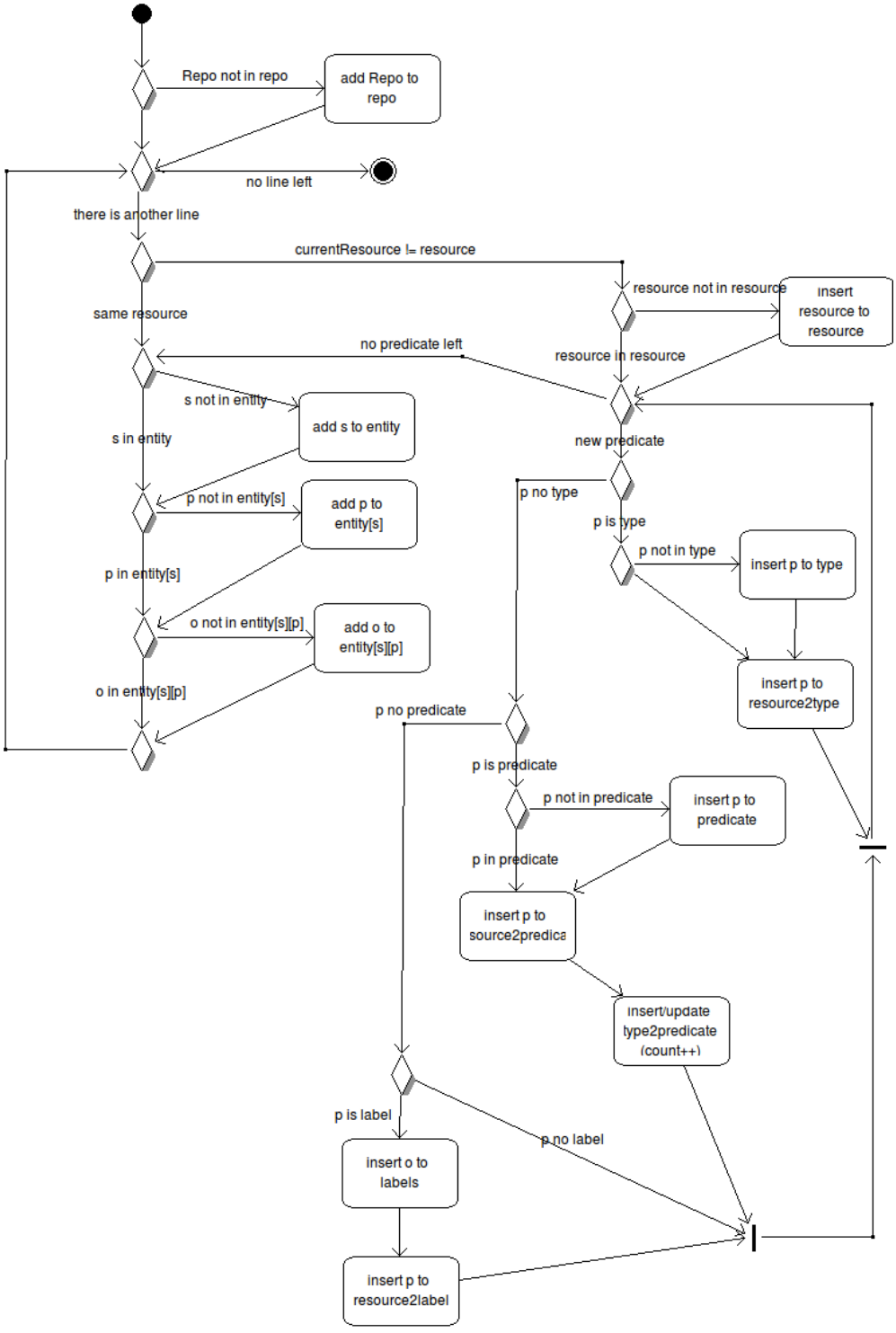

Abbildung 5.2: Repository-Crawler

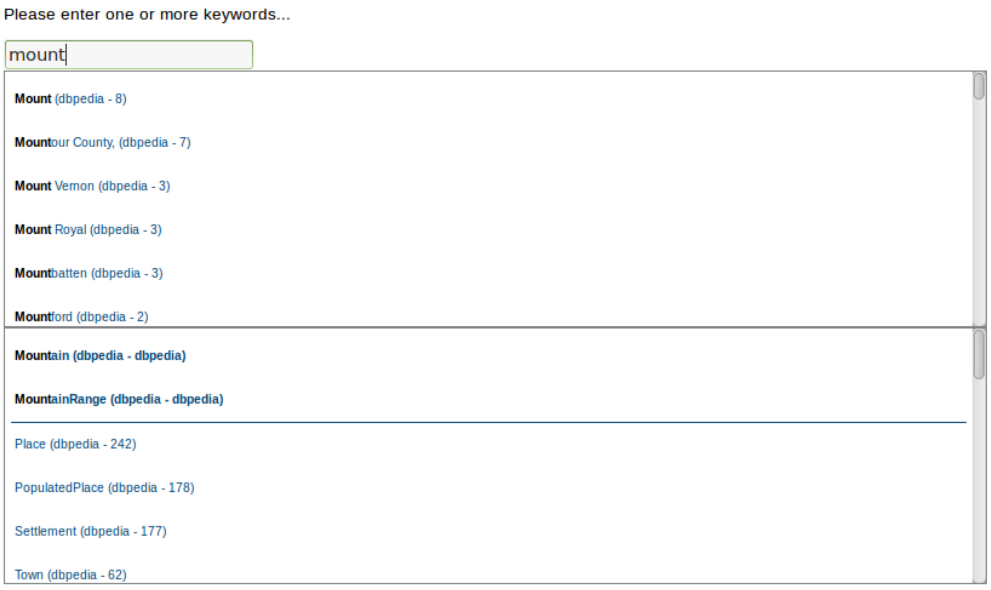

Abbildung 5.3: Suchfeld

Kategorie 1: Entitäten

Vorschläge dieser Kategorie entsprechen eindeutigen Resourcen aus den geladenen RDF-Repositories. Sie stellen einzelne für sich geschlossene Entitäten dar. Ein Beispiel für ein Suchergebnis der Kategorie 1 wäre zum Beispiel der Schriftsteller und Philosoph „Albert Camus“.

Je spezifischer der Suchbegriff, desto eingeschränkter gestaltet sich das Erebnis einer solchen Abfrage. Ein Suchbegriff kann mitunter Treffer in den unterschiedlichsten Domänen hervorbringen. Daher schränkt der User in einem weiteren Schritt seine Suchdomäne über die Auswahl einer bestimmten Kategorie ein. Damit erhält er im Anschluss nur noch Resultate aus der von ihm gewählten Domäne.

Kategorie 2: Explizite Kategorien

Der Suchbegriff entspricht einem Überbegriff der als Kategorie im Repository gelistet ist. Beispiele hierfür wären der Begriff „Person“ oder „Mountain“. Diese Form ist die allgemeinste Suche die der User durchführen kann. Hier erhält er alle Resultate aus der gewählten Überkategorie, die er in einem weiteren Schritt nach bestimmten Kriterien einschränken kann. Er erhält aber immer eine Teilmenge aus allen möglichen Resultaten dieser Kategorie. Ein mögliches Szenario für eine solche Abfrage ist zum Beispiel das Interesse an Informationen über Berge in Österreich.

Kategorie 3: Implizite Kategorien

Der Suchbegriff des Users deckt sich mit mehreren Resourcen bzw. Entitäten im Repository. Die Entitäten werden über ihre Kategorie/Typ zusammengefasst und über ihren Typ dem User präsentiert. Als Beispiel wird noch einmal der Suchbegriff aus dem Beispiel

der Kategorie 1 herangezogen. Sucht der User nach „Albert Camus“ werden ihm hier die Kategorien „Person“ und „Philosopher“ vorgeschlagen. Der User kann seine Abfrage nun dementsprechend spezifizieren indem er sich für eine dieser Kategorien entscheidet. Die Ergebnisse dieser Kategorie sind abhängig von der Auswahl in Kategorie 1. Wählt der User keinen speziellen Eintrag in Kategorie 1 werden alle möglichen Resultate die dem Suchwort entsprechen in deren Kategorien dargestellt. Entscheidet sich der User aber für eine spezielle Resource aktualisiert sich die Ansicht, und es werden nur noch die Domänen zur Auswahl angezeigt die mit der gewählten Resource in Einklang zu bringen sind.

Je nachdem, auf welche Art von Ergebnis die Entscheidung des Users fällt, wird eine der beiden Suchfunktionalitäten des Wizard angewandt. Entscheidet sich der User für ein Ergebnis der Kategorie 1 und 3 schaltet der Wizard in den Modus „Entity-Search“, bei Kategorie 2 in den Modus „Faceted-Search“.

In den folgenden Abschnitten werden diese beiden unterschiedlichen Funktionen zur Verfeinerung bzw. Erweiterung der Userabfrage beschrieben.

5.3.2 Entity-Search

In diesem Szenario ist der User an Informationen zu einer genau definierten Resource interessiert. Er sucht nach etwas Speziellem. In diesem Modus geht der Wizard davon aus, dass der User ein bestimmtes Ziel verfolgt und genau weiß welche Informationen er aquirieren möchte.

Die Funktionalität soll an Hand eines Beispiels erläutert werden. Gesucht wird nach der Entität „Mount Everest“. Der User verwendet hierfür das im Abschnitt 5.3.1 vorgestellte Suchfeld.

Interface

Wie in Abbildung 5.4 gibt der User den Suchbegriff in das Suchfeld ein und erhält über die Autovervollständigung Vorschläge die seiner Eingabe entsprechen. Er wählt den Eintrag „Mount Everest“ und die zugehörige Kategorie “Mountain“.

Abbildung 5.8 zeigt den auf die Suchmaske folgenden Screen mit der Darstellung der gefundenen Entitäten.

Der Screen teilt sich hierbei in zwei Hauptbereiche. Die Ergebnistabelle, in der die Resultate der Suchabfrage dargestellt werden, und den Kontrollbereich der zur nachträglichen Manipulation und Verfeinerung des Ergebnisses dient.

- **Der Kontrollbereich**

  Die Linke Spalte bietet einerseits Informationen über die Herkunft der gerade gewählten Resourcen wie das aktuelle RDF-Repository und die Kategorie, andererseits bietet sie die Möglichkeit der Auswahl der gewünschten Informationen über eine Resource in Form von Prädikaten. Es können Prädikate hinzugefügt oder entfernt werden. Des Weiteren kann die Reihenfolge der Prädikate im Ergebnis über Drag-And-Drop beliebig abgeändert werden.

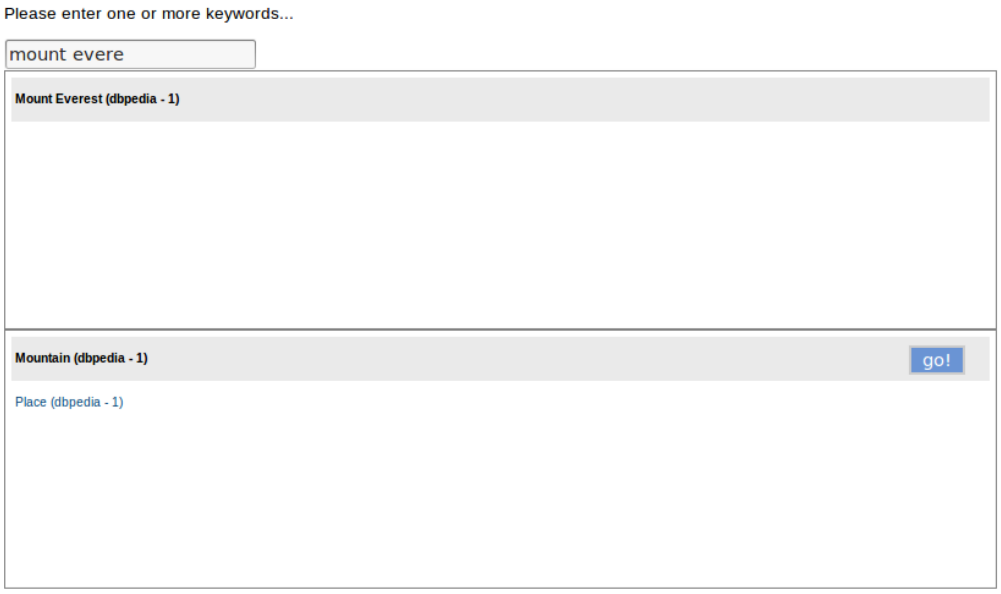

Abbildung 5.4: Suche nach Entität „Mount Everest“

Über das Dropdown-Menü auf der linken Seite können weitere Prädikate hinzugefügt werden. Hierfür wählt der User einfach das gewünschte Prädikat und bestätigt mit einem Klick auf das Pluszeichen. Die Ergebnistabelle wird daraufhin um das neue Prädikat ergänzt (Siehe Abbildung 5.5).

Der User hat die Möglichkeit zu den bestehenden Resourcen weitere Resourcen des gleichen Typs hinzuzufügen. Hierzu klickt er auf den Button „add another entity“. Er wird daraufhin auf zu einer Suchmaske geleitet die der ersten Suchmaske sehr ähnlich ist. Der Unterschied liegt im Fokus der Suche. Das Repository und die gewünschte Kategorie ist in diesem Fall schon ausgewählt und der Suchraum dementsprechend eingeschränkt. In Abbildung 5.6 fügt der User den Berg „Kilimanjaro“ hinzu. Abbildung 5.7 zeigt die um die Resource erweiterte Ergebnistabelle.

- **Die Ergebnistabelle**

  Die gefundenen Resourcen werden in einer Ergebnistabelle dargestellt die die fünf meistvorkommenden Prädikate der Kategorie derer die Resource angehört beinhaltet. Die Tabelle bietet eine Reihe von Funktionalitäten. Wie schon oben erwähnt können die Tabellenspalten beliebig über die Kontrollen auf der linken Seite des Screens verschoben werden. Über die Spaltenheader ist eine alphabetische Sortierung der Tabelle nach den Werten einer beliebigen Spalte möglich. Mit dem Löschen-Symbol in der letzten Spalte einer jeden Zeile kann eine einzelne Zeile, und damit eine einzelne Resource, aus dem Suchergebnis entfernt werden. Über die Checkboxen auf der linken Seite können eine, mehrere oder alle Zeilen bzw. Resourcen ausgewählt, und in einer „Collection“ gespeichert werden (siehe Abschnitt 5.4.

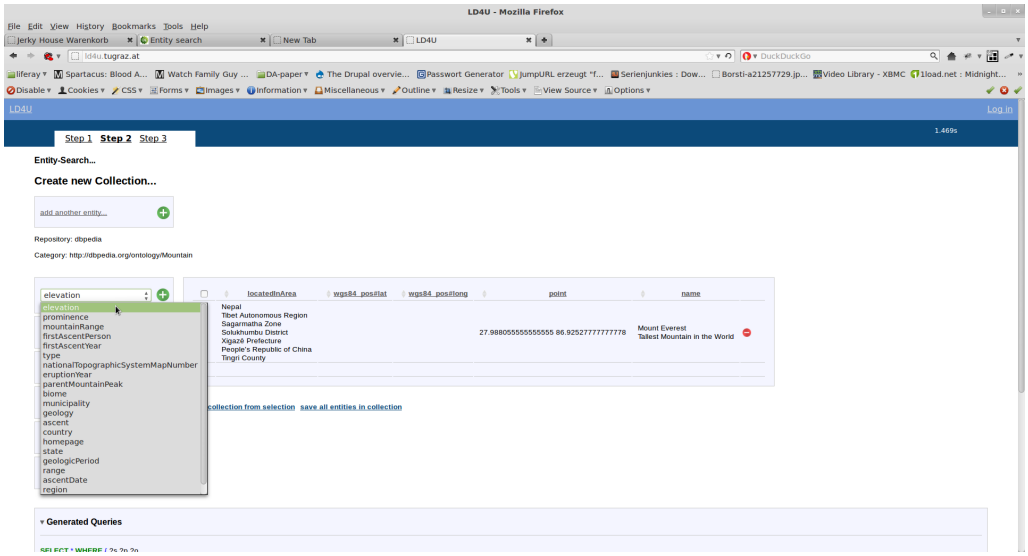

Abbildung 5.5: Prädikatauswahl

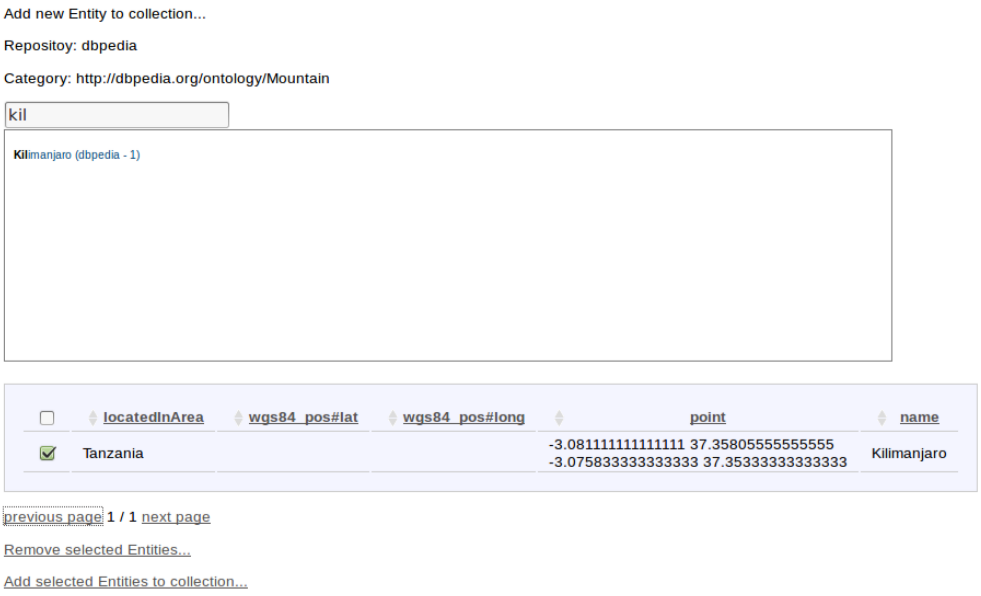

Abbildung 5.6: Hinzufügen weiterer Entitäten

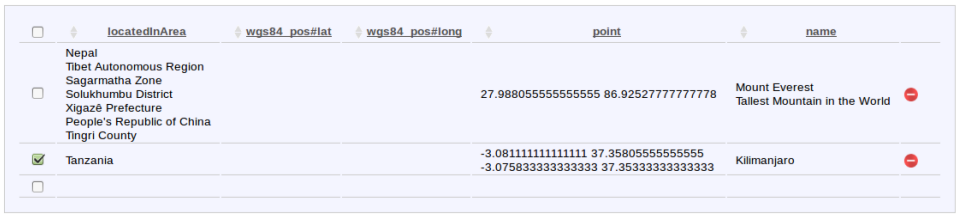

Abbildung 5.7: Erweiterte Ergebnistabelle

Abbildung 5.8: Ergebnis einer Entitätensuche

Abfragegenerierung

Wie erfolgt nun die Generierung einer Abfrage vom Typ Entity-Search? Über die vorhergehende Auswahl im Suchfeld ist das gewünschte Repository an das die Abfrage gerichtet wird dem System bekannt. Ebenso die entsprechenden Resourcen. Diese sind über den Preprocessing Schritt beim Import des entsprechenden Repositories in der Datenbank gespeichert. Es erfolgt nun eine Kombination des Repositories und der gewählten Resourcen mit den gewählten Prädikaten bzw. gewünschten Eigenschaften des Suchbegriffs. Die Resourcen der Prädikate werden aus der User-Auswahl extrahiert und mit diesen gesammelten Daten wird eine SPARQL-Query generiert. Abbildung 5.9 zeigt eine solche generierte Query wie sie auch dem User zur Ansicht angeboten wird. Wie in der Query zu sehen ist, erfolgt die erwartete Anwort vom Repository im klassischen Tripel Format mit der Form Subjekt, Prädikat, Objekt. Dies hat den Vorteil dass im Falle von mehrfachen Ergebnissen desselben Prädikats keine Verdoppelungen der übrigen Ergebnisse vorkommen wie es in einer Query der Fall wäre die versuchen würde die Ergebnistabelle direkt abzubilden und die so erhaltenen Daten für verschiedenste Darstellungen herangezogen werden können. Die Antwort des SPARQL-Endpoints des Repositories wird nun noch in einem Nachbearbeitungsschritt für die Applikation angepasst.

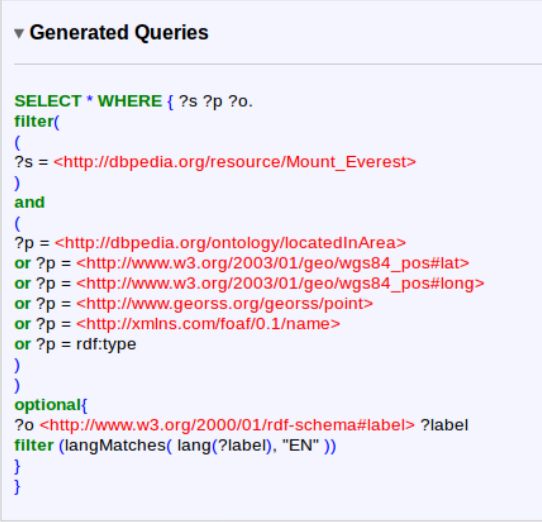

Abbildung 5.9: Generierte SPARQL-Query

Post-Processing

Die Ergebnistripel der Abfrage werden nach ihren Resourcen sortiert und mehrere Ergebnisse des gleichen Prädikates zusammengefasst. Die so erhaltene Struktur wird an das Userinterface weitergegeben und in einer Ergebnistabelle dargestellt. Die Zeilen der Tabelle repräsentieren dabei jeweils eine Resource, die Spalten stehen für je ein Prädikat.

5.3.3 Faceted-Search

Der zweite Modus des Wizard wird als „Faceted-Search“ bezeichnet. Hierbei geht der User von einem allgemeinen Startpunkt mit einer großen Ergebnismenge aus und verfeinert sein Ergebnis durch die Angabe von bestimmten Kriterien um die Ergebnismenge einzuschränken. Er gibt sozusagen Kritieren vor. Als erläuterndes Beispiel für einen Suchvorgang wird folgendes Szenario durchgespielt: Der User ist an Bergen mit einer Höhe zwischen 2.000 und 5.000 Metern interessiert.

Interface

Wie schon in Abschnitt 5.3.2 dargestellt beginnt der User mit der Eingabe eines Schlüsselwortes in der Suchmaske bekannt aus Abschnitt 5.3.1. In diesem Fall ist er an Bergen interessiert. Deswegen wählt er den Begriff „Mountain“. Das System schlägt dem User wiederum Begriffe zu Resourcen und Kategorien die den Suchbegriff enthalten vor. Der User kann hier eine aus den drei verschiedenen, aus Abschnitt 5.3.1 bekannten, Kategorien wählen. Da er in diesem Fall an allgemeinen Informationen und nicht an speziellen Resourcen interessiert ist wählt er den Überbegriff „Mountain“ aus der zweiten Kategorie (siehe Abbildung 5.10).

Abbildung 5.10: Faceted Search: Suche nach Kategorie „Mountain“

Der User erhält daraufhin alle Ergebnisse der gewählten Kategorie „Mountain“. Die Ansicht unterscheidet sich auf den ersten Blick nicht wesentlich von einem Ergebnis einer Entity-Search, das Interface beinhaltet allerdings Zusatzelemente wie in Abbildung 5.11 ersichtlich. Die Ergebnistabelle verfügt über einen Pager und eine erweiterte Ansicht der Prädikate. Für jedes Prädikat kann ein Filterwerkzeug aufgeklappt werden, mit dem der User in der Lage ist die Ergebnismenge einzuschränken (siehe Abbildung 5.11).

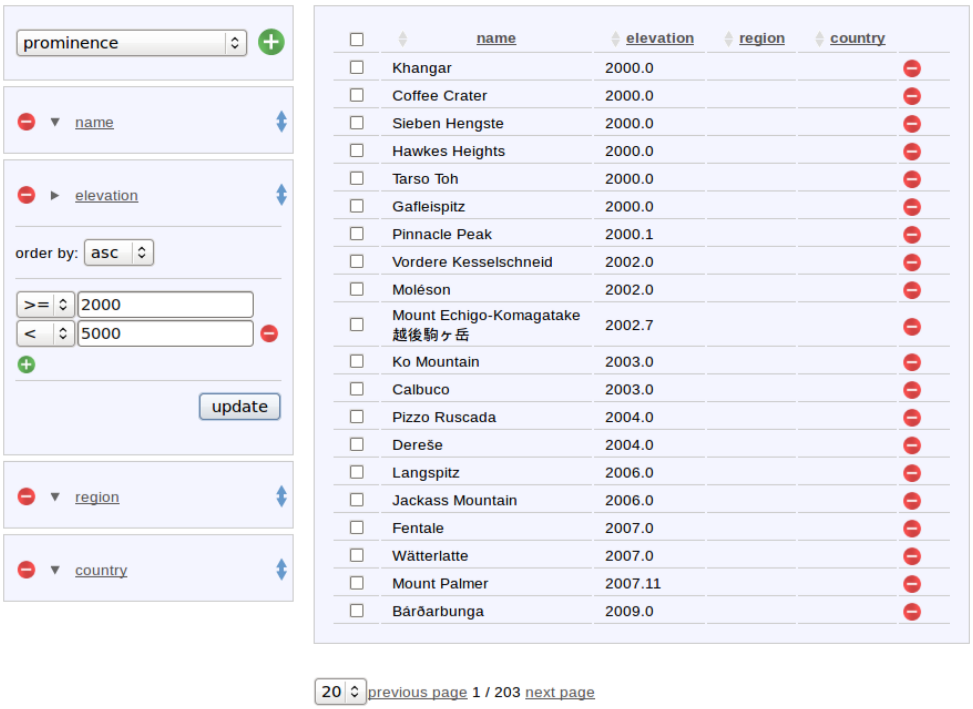

Abbildung 5.11: Faceted Search: Filterkriterien

Paging

Der Pagingmechanismus beschränkt das sichtbare Ergebnis auf eine definierbare Anzahl von Ergebnisse pro Seite. Der User hat die Möglichkeit die Ergebnisse seitenweise zu durchlaufen. Eine genauere Beschreibung des Pagers und dessen Funktionsweise siehe Abschnitt 5.3.3.

Filter

Je nach Datentyp des gewählten Prädikates können Kriterien zur Einschränkung festgelegt werden. In der ersten Version werden die Datentypen „String“, „Integer“, „Float“ und „Date“ unterstützt. Es können beliebig viele Filter pro Prädikat gewählt werden. Mehrere Filter entsprechen im Falle eines Zahl und Datumsformts einer logischen Und-Verknüpfung in der Abfrage. Falls es sich bei dem Wert des Prädikates um einen String handelt werden mehrere Filterangaben als Oder-Verknüpfung gehandhabt. Es müssen alle Filterkriterien zutreffen, damit eine Resource in der Ergebnismenge aufscheint.

Abfragegenerierung

Die Generierung der SPARQL-Query der Faceted-Search des Wizards erfolgt anders als im Falle einer Entity-Search, bei der die gesuchten Resourcen des entsprechenden Repositories durch den Preprocessing-Schritt schon bekannt sind (siehe hierzu Abschnitt 5.2). In diesem Modus erfolgt die Erstellung der Abfrage in zwei Schritten. Im ersten Schritt werden die

erforderlichen Resourcen für die aktuelle Abfrage ermittelt, mit denen in einem zweiten Schritt dann die eigentliche Abfrage nach Schema der Methode 1 (siehe Abbildung 5.9) zur Aquirierung der gewünschten Daten durchgeführt wird.

Die Teilung in zwei Schritte ist notwendig, da es sich bei der Faceted-Search um eine Auswahl über eine Oberkategorie handelt. Es müssten also alle Daten von allen Resourcen der gewählten Kategorie abgefragt werden. Durch die Trennung kann die zeitintensivere Abfrage der Prädikate auf geziehlte Resourcen beschränkt werden was für ein reaktives System unumgänglich ist.

- **Schritt 1: Ermittlung der erforderlichen Resourcen**

  Abbildung 5.12 zeigt eine Abfrage zur Ermittlung der Resourcen für die Kategorie «Mountain» aus dem oben genannten Userbeispiel. Die vom User gemachten Einschränkungen in Form von Kriterien werden ebenfalls in dieser Abfrage berücksichtigt. Die auf diesem Weg vom gewählten Repository ermittelten Resourcen werden vom System gespeichert und für die weitere Abfrage der eigentlichen Prädikatwerte verwendet.

```
SELECT distinct(?s) WHERE
{
?s rdf:type <http://dbpedia.org/ontology/Mountain>.
?s <http://dbpedia.org/ontology/elevation> ?pred0.
FILTER (?pred0 >= 2000)
FILTER (?pred0 < 5000)
}
order by
ASC(?pred0 )
```

Abbildung 5.12: Faceted Search: Ermittlung der benötigten Resourcen

- **Schritt 2: Abfrage der Daten**

  Die Abfrage der eigentlichen Daten erfolgt wie oben erwähnt über die in vorhergehenden Schritt ermittelten Resourcen. Die Entscheidung welche Resourcen in der Abfragequery verwendet werden, wird über den Pager bestimmt (siehe Abbildung 5.10). Dieser gibt einerseits die Anzahl der anzuzeigenden Resourcen pro Seite an, und andererseits auch den zu verwendenden Offset über alle Resourcen. Es werden also nur die Daten der Resourcen abgefragt die auch angezeigt werden. Einmal angezeigte Daten bleiben für eine spätere Ansicht erhalten. Es werden nur noch nicht vorhandene Daten vom Repository abgefragt.

Für einen eventuellen Export der gewünschten Datensätze müssen alle Datensätze geladen werden. Dies erfolgt in einem iterativen Verfahren, da die Länge der SPARQL-Query vom SPARQL-Endpoint des jeweiligen Repositories beschränkt ist. Es kann also nur eine gewisse Anzahl von Resourcen pro Abfrage verarbeitet werden.

Post-Processing

Die Nachbearbeitung der vom SPARQL-Endpoint empfangenen Tripel erfolgt auf die gleiche Weise wie Abschnitt 5.3.2 beschrieben, da die in diesem Kapitel in Abschnitt 5.3.3 beschriebene Abfrage der Daten, wie bei der Abfrage einer Entity-Search, über die Resource-URLs erfolgt.

5.4 Speichern und Laden von Abfragen

Ein Feature des SPARQL-Wizard ist eine einfache Benutzerverwaltung. Es können User über das Backend angelegt werden, die nach dem Einloggen in das System die Möglichkeit haben einzelne Abfragen bzw. Konfigurationen des Wizards zu speichern. Dadurch ist der User in der Lage begonnene Abfragen zu einem späteren Zeitpunkt fortzuführen oder auch wieder zu verwerfen. Ein wichtiger Aspekt bei der Speicherung der Abfragen ist die Tatsache, dass nicht die Ergebnisse der Queries, sondern der Zustand des Wizards selbst, und damit die eigentliche Query, gespeichert wird. Das bedeutet, dass die eigentlichen Daten mit jedem Aufruf neu aus dem Repository geladen werden und somit immer aktuell sind.

Der User speichert seine Abfragen in so genannten „Collections“. Eine Collection lässt sich als Sammlung von Resourcen verstehen und ergibt sich durch eine Abfragesession mit dem Wizard. Es wird zwischen einer einfachen Collection und einer „Smart-Collection“ unterschieden.

Collection

Eine Collection ist ein Sammlung von Resourcen. Der User hat die Möglichkeit einzelne oder alle Resourcen aus der Ergebnistabelle seiner Abfrage zu markieren und in einer Collection abzuspeichern die, versehen mit einem Titel und einer optionalen Beschreibung, in seinem Profil zur späteren Ansicht oder Bearbeitung bereitsteht (siehe Abbildung 5.13).

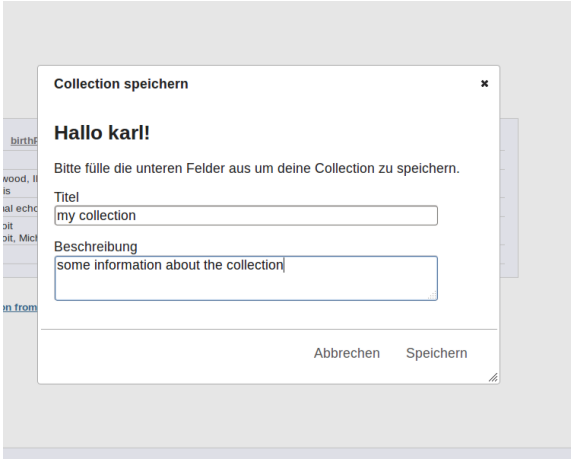

Abbildung 5.13: Usereingaben bei der Speicherung von Collections

Neben den Resourcen werden auch alle weiteren Einstellungen der aktuellen Wizardinstanz gespeichert. Es wird also ein Abbild des aktuellen Status des Wizard erstellt und in der Datenbank abgelegt. Beim Laden einer gespeicherten Collection wird der gespeicherte Zustand des Wizard mit den gewählten Resourcen wiederhergestellt und für ein aktuelles Ergebnis die daraus resultierende Query abgesetzt.

Smart-Collection

Eine Smart-Collection unterscheidet sich von einer einfachen Collection darin, dass in ihr keine Resourcen an sich gespeichert werden. Sie wird im Kontext der Faceted-Search (siehe Abschnitt 5.3.3) als Option für den User angeboten. Eine Smart-Collection sichert vielmehr die aktuellen Filtereinstellungen der verwendeten Prädikate und die für die Faceted-Search gewählte Kategorie.

Abbildung 5.14: Datenbanktabelle zur Speicherung von Collections

Beim Laden einer Smart-Collection wird wie bei der einfachen Collection der Zustand des Wizard wiederhergestellt und die aus den Filtereinstellungen generierte Query der Faceted-Search für ein aktuelles Ergebnis abgesetzt.

User-Management

Wie schon erwähnt verfügt der Wizard über eine simple Benutzerverwaltung. Hierfür wurde das djangoeigene Authentifizierungs- und Verwaltungssystem[2] benutzt und angepasst. Die Anpassungen beziehen sich in erster Linie auf die Datenbank und das Usermodell. Es wurde eine neue Tabelle und zugehöriges Model (siehe Abschnitt 4.3.1 für mehr Informationen über Models) eingeführt (siehe Abbildung 5.14). Sie dient zur Speicherung der von den Benutzern erstellten Collections und enthält neben einer Verknüpfung zum jeweiligen User Einträge über den Typ der Collection, deren Titel und Beschreibung, das Datum der Erstellung, die zuletzt erzeugten Abfragen und den aktuellen Zustand des Wizard als serialisiertes Javascript-Objekt.

Jeder User besitzt eine persönliche Seite „My Collections“ zur Verwaltung seiner in Collections gespeicherten Abfragen. In dieser frühen Version des Wizard sind die Möglichkeiten des Users auf einfache Aktionen wie Laden und Löschen der gelisteten Collections beschränkt.

[2] `https://docs.djangoproject.com/en/dev/topics/auth/`

Kapitel 6

Schlussbemerkung und Ausblick

6.1 Zusammenfassung

In dieser Arbeit wurde auf die Entwicklung und die Hintergründe des Semantic Web eingegangen und ausgehend von den Fähigkeiten und Limitierungen des Internets auf die Möglichkeiten eines mit semantischen Daten angereicherten und untereinander verknüpften „Web of Data“ hingewiesen, und die benötigten Technologien erläutert, die als Voraussetzung für die Bildung einer „Linked Open Data Cloud“ gesehen werden können. Das auf XML basierende „Resource Description Framework“, das vom W3C zum Standard für Metadaten erhoben wurde, dient der Beschreibung von Resourcen, die mittels der Abfragesprache SPARQL über SPARQL-Endpoints der verschiedenen Repositories abgefragt werden können. Es wurden Anforderungen an einen Wizard zur Erstellung solcher, für unerfahrene Nutzer sehr schwer zu formulierenden, SPARQL-Queries, auch auf Grund der in dieser Arbeit besprochenen, bereits existierenden Systeme zur Generierung von Abfragen gebildet. Auf Grund dieser Anforderungen wurde eine Plattform entwickelt, mit der auf einfachen und intuitivem Weg Abfragen an vorher integrierte Wissensbasen gestellt werden können. Im Rahmen der Arbeit lag der Fokus auf der Wissensbasis DBpedia, die als Beispiel herangezogen und auch theoretisch behandelt wurde.

Zur Umsetzung des Query-Wizards standen mehrere Ansätze zur Wahl. Die Erfahrungen, die bei der Umsetzung gemacht wurden haben gezeigt, dass bei der Umsetzung eines RDF-basierenden Systems einige Dinge beachtet werden müssen. Zum einen sollte man versuchen die Anzahl der Abfragen an ein Repository möglichst gering zu halten und bei der Abfrage selbst nach Möglichkeit rechenintensive Abfragekonstrukte wie beispielsweise Pattern in Kombination mit regulären Ausdrücken vermeiden. Das Betreiben eines gespiegelten Repositories ist bei relativ kleinen externen Datenständen eine durchaus gute Lösung, wenn es um die Schaffung eines performanten Systems geht, und die Aktualität der Daten nicht so wichtig ist. Bei Wissensbasen in der Größenordnung von DBpedia ist eine Kopie des Repositories auf Grund des enormen Resourcenverbrauchs nicht sinnvoll. Durch die Nutzung der Originalrepositories unterliegt man natürlich, neben allen Vorteilen der Resourcenschonung und der Aktualität der generierten Daten, dessen Beschränkungen. Dies macht sich besonders beim verwendeteten SPARQL-Endpoint bemerkbar. Die Länge einer Abfrage ist auf eine bestimmte Anzahl von Zeichen beschränkt. Auch die

Anzahl der maximal zurückgegebenen Ergebnisse ist eingeschränkt. Dadurch steht man bei automatisch generierten Queries vor dem Problem, den Generierungsprozess an diese Gegebenheiten anzupassen. Auch unterliegt man in Fragen der Performance weitgehend der Leistungsfähigkeit der angesteuerten Endpoints. Timeouts bei der Abfrage größerer Datenmengen sind beispielsweise bei DBpedia keine Seltenheit.

6.2 Zukünftige Arbeiten

Im Hinblick auf die in dieser Arbeit definierten Anforderungen an einen SPARQL-Wizard gibt es noch einige Verbesserungen die an dem bestehenden Prototypen vorgenommen werden können um seine bestehenden Funktionalitäten zu erweitern oder zu verbessern.

Erweiterung der SPARQL-Query Abdeckung

Der Wizard deckt in seiner jetzigen Version Basis-Sprachkonstrukte der SELECT-Anweisung von SPARQL ab. Für zukünfitge Version ist eine Erweiterung der Funktionalität um weitere SPARQL-Sprachkonstrukte denkbar. Zum Beispiel wäre eine Erweiterung der Filter-Funktionalitäten um weitere Datentypen oder die Unterstützung von Regulären Ausdrücken.

Export-Funktionalität

Derzeit können die generierten Daten nicht aus dem Wizard extrahiert werden. Hier wäre eine Export-Funktionalität, die den Export der Daten in unterschiedlichen Formaten wie CSV, RDF, XLS und dergleichen unterstützt denkbar.

Webinterface zur Repositoryerweiterung

Für die Erweiterung des Wizard um weitere Datensätze bzw. weitere Repositories könnte es eine Backendfunktionalität für den Administrator geben. Hier kann der Admin entweder weitere Sourcefiles direkt uploaden, oder einen Link zu einem weiteren Dump angeben. Der Dump wird dann vom System automatisiert geladen und über das Import-Skript wird die aktuelle Datenbank um diese RDF-Resource erweitert.

Ranking von Suchergebnissen

Im aktuellen Wizard werden die vorgeschlagenen Entitäten oder Prädikate auf Grund von Häufigkeitsinformationen gereiht und dem User vorgeschlagen. Zusätzlich könnte eine Art Page-Rank Algorithmus wie in Dali et al. (2012) beschrieben implementiert werden, um die Autovervollständigung zu optimieren.

Caching

Um die Performance bei häufigen Abfragen zu verbessern könnte ein weiterer Caching-Mechanismus, zum Beispiel nach Vorbild von Yang and Wu (2012) zwischengeschalten werden.

Mehrsprachigkeit

Bei der Labelexraktion wird bisher keine Unterscheidung zwischen unterschiedlichen Sprachen getroffen. Hier kann das System um die Unterstützung von Mehrsprachigkeit sowohl bei der Aufbereitung der Daten, als auch im Userinterface hinzugefügt werden. Der User ist dann in der Lage Ergebnisse einer gewünschten Zielsprache anzuzeigen bzw. Sprache des Interfaces nach seinen Wünschen anzupassen.

Literaturverzeichnis

Antoniou, G., Franconi, E., and van Harmelen, F. (2005). Introduction to semantic web ontology languages. In Eisinger, N. and Maluszynski, J., editors, *Reasoning Web*, volume 3564 of *Lecture Notes in Computer Science*, pages 96–96. Springer Berlin / Heidelberg.

Berners-Lee, T. (1998). What the semantic web can represent. `http://www.w3.org/DesignIssues/RDFnot.html`.

Berners-Lee, T. (2006). Design issues: Linked data note. `http://www.w3.org/DesignIssues/LinkedData.html`.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

Bernstein, A., Kaufmann, E., Kaiser, C., and Kiefer, C. (2006). Ginseng: A guided input natural language search engine for querying ontologies. In *2006 Jena User Conference, Bristol, UK*.

Bizer, C. (2011). *DBpedia 3.7 released, including 15 localized Editions*. `http://blog.dbpedia.org/2011/09/11/dbpedia-37-released-including-15-localized-editions/`.

Bizer, C., Heath, T., and Berners-Lee, T. (2009a). Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009b). Dbpedia - a crystallization point for the web of data. *Web Semant.*, 7(3):154–165.

Brickley, D. and Guha, R. V., editors (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. World Wide Web Consortium.

Catarci, T., Costabile, M. F., Levialdi, S., and Batini, C. (1997). Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8:215–260.

Chris Bizer, A. J. and Cyganiak, R. (2012). *State of the LOD Cloud*. `http://www4.wiwiss.fu-berlin.de/lodcloud/state/`.

Dali, L., Fortuna, B., Duc, T., and Mladenić, D. (2012). Query-independent learning to rank for rdf entity search. In Simperl, E., Cimiano, P., Polleres, A., Corcho, O., and Presutti, V., editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 484–498. Springer Berlin / Heidelberg.

Dengel, A. (2012). Linked open data, semantic web datensätze. In Dengel, A., editor, *Semantische Technologien*, pages 183–203. Spektrum Akademischer Verlag.

djangobook (2009). The django book. Website. Available online at `http://www.djangobook.com/en/2.0/`; visited on April 15th 2012.

Ernesti, J. and Kaiser, P. (2009). *Python 3, Das umfassende Handbuch*. Galileo Press, Bonn.

Hahn, R., Bizer, C., Sahnwaldt, C., Herta, C., Robinson, S., Bürgle, M., Düwiger, H., and Scheel, U. (2010). Faceted wikipedia search. In Abramowicz, W. and Tolksdorf, R., editors, *BIS*, volume 47 of *Lecture Notes in Business Information Processing*, pages 1–11. Springer.

Halevy, A. (2012). Bringing (web) databases to the masses. In Simperl, E., Cimiano, P., Polleres, A., Corcho, O., and Presutti, V., editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 3–3. Springer Berlin / Heidelberg.

Han, L., Finin, T., and Joshi, A. (2012). Gorelations: An intuitive query system for dbpedia. In Pan, J., Chen, H., Kim, H.-G., Li, J., Wu, Z., Horrocks, I., Mizoguchi, R., and Wu, Z., editors, *The Semantic Web*, volume 7185 of *Lecture Notes in Computer Science*, pages 334–341. Springer Berlin / Heidelberg.

Hartig, O. (2012). Sparql for a web of linked data: Semantics and computability. In Simperl, E., Cimiano, P., Polleres, A., Corcho, O., and Presutti, V., editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 8–23. Springer Berlin / Heidelberg.

Heath, T. and Bizer, C. (2011). *Linked Data: Evolving the Web Into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool.

Heflin, J. (2004). OWL Web Ontology Language Use Cases and Requirements. Technical report, W3C.

Horrocks, I., Parsia, B., Patel-Schneider, P., and Hendler, J. (2005). Semantic web architecture: Stack or two towers? In Fages, F. and Soliman, S., editors, *Principles and Practice of Semantic Web Reasoning*, volume 3703 of *Lecture Notes in Computer Science*, pages 37–41. Springer Berlin / Heidelberg.

Jakob, M. (2011). *The DBpedia Ontology*. `http://wiki.dbpedia.org/Ontology?v=181z`.

Kaufmann, E., Bernstein, A., and Fischer, L. (2007). NLP-Reduce: A „naive“ but Domain-independent Natural Language Interface for Querying Ontologies.

Kiefer, C., Bernstein, A., and Stocker, M. (2007). The fundamentals of iSPARQL: a virtual triple approach for Similarity-Based semantic web tasks. *Lecture Notes in Computer Science*, 4825:295.

Lacasta, J., Iso, J. N., Soria, F. J. Z., Lacasta, J., Nogueras-Iso, J., and Zarazaga-Soria, F. J. (2010). *Terminological Ontologies: Design, Management and Practical Applications*, volume 9 of *Semantic Web and Beyond*. Springer US. 10.1007/978-1-4419-6981-1_1.

Lehmann, J. and Bühmann, L. (2011). Autosparql: Let users query your knowledge base. In Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., and Pan, J., editors, *The Semantic Web: Research and Applications*, volume 6643 of *Lecture Notes in Computer Science*, pages 63–79. Springer Berlin / Heidelberg.

Lopez, V., Fernández, M., Motta, E., and Stieler, N. (2011). Poweraqua: supporting users in querying and exploring the semantic web content. In *Semantik Web Journal*, page 17. IOS Press.

Manola, F. and Miller, E., editors (2004). *RDF Primer*. W3C Recommendation. World Wide Web Consortium.

McCarthy, L., Vandervalk, B., and Wilkinson, M. (2012). Sparql assist language-neutral query composer. *BMC Bioinformatics*, 13:1–9.

Morsey, M., Lehmann, J., Auer, S., and Ngonga Ngomo, A.-C. (2011). Dbpedia sparql benchmark – performance assessment with real queries on real data. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., and Blomqvist, E., editors, *The Semantic Web – ISWC 2011*, volume 7031 of *Lecture Notes in Computer Science*, pages 454–469. Springer Berlin / Heidelberg.

Pellegrini, T. (2006). *Semantic Web, Wege zur vernetzten Wissenschaft*. Springer-Verlag Berlin Heidelberg.

Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and complexity of sparql. In Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., and Aroyo, L., editors, *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 30–43. Springer Berlin / Heidelberg.

Richard Cyganiak, Jun Zhao, K. A. and Hausenblas, M. (2012). *Vocabulary of Interlinked Datasets (VoID)*. `http://vocab.deri.ie/void/`.

Schmitz-Esser, W. and Sigel, A. (2006). Introducing terminology-based ontologies. 9th International Conference of the International Society for Knowledge Organization (ISKO), Vienna, Austria, July 2006, 5-7th.

Shadbolt, N., Berners-Lee, T., and Hall, W. (2006). The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101.

Smart, P., Russell, A., Braines, D., Kalfoglou, Y., Bao, J., and Shadbolt, N. (2008). A visual approach to semantic query design using a web-based graphical query designer. In Gangemi, A. and Euzenat, J., editors, *Knowledge Engineering: Practice and Patterns*, volume 5268 of *Lecture Notes in Computer Science*, pages 275–291. Springer Berlin / Heidelberg.

Sowa, J. F. (2000). Ontology, metadata, and semiotics. In *Proceedings of the Linguistic on Conceptual Structures: Logical Linguistic, and Computational Issues*, ICCS '00, pages 55–81, London, UK, UK. Springer-Verlag.

Weiss, M. (2009). Resource description framework. In LIU, L. and ÖZSU, M. T., editors, *Encyclopedia of Database Systems*, pages 2423–2425. Springer US.

Wittgenstein, L. (1984). *Philosophische Untersuchungen*. Suhrkamp, Frankfurt.

Yang, M. and Wu, G. (2012). Semantic caching for semantic web applications. In Pan, J., Chen, H., Kim, H.-G., Li, J., Wu, Z., Horrocks, I., Mizoguchi, R., and Wu, Z., editors, *The Semantic Web*, volume 7185 of *Lecture Notes in Computer Science*, pages 192–209. Springer Berlin / Heidelberg.

Yu, L. (2011). *A Developer's Guide to the Semantic Web*. Springer-Verlag Berlin Heidelberg.