UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Aspekte der Kommunikation und Datenintegration in semantischen Daten-Wikis

Diplomarbeit

Leipzig, Oktober 2009

vorgelegt von

Philipp Frischmuth
geb. am: 09.10.1984

Studiengang Informatik

Zusammenfassung

Das Semantic Web, eine Erweiterung des ursprünglichen World Wide Web um eine semantische Schicht, kann die Integration von Informationen aus verschiedenen Datenquellen stark vereinfachen. Mit RDF und der SPARQL-Anfragesprache wurden Standards etabliert, die eine einheitliche Darstellung von strukturierten Informationen ermöglichen und diese abfragbar machen. Mit Linked Data werden diese Informationen über ein einheitliches Protokoll verfügbar gemacht und es entsteht ein Netz aus Daten, anstelle von Dokumenten. In der vorliegenden Arbeit werden Aspekte einer auf solchen semantischen Technologien basierenden Datenintegration betrachtet und analysiert. Darauf aufbauend wird ein System spezifiziert und implementiert, das die Ergebnisse dieser Untersuchungen in einer konkreten Anwendung realisiert. Als Basis für die Implementierung dient OntoWiki, ein semantisches Daten-Wiki.

Stichworte Semantic Web, Linked Data, OntoWiki

Inhaltsverzeichnis

1. Einleitung

In der vorliegenden Arbeit sollen Aspekte der Kommunikation zwischen semantischen Daten-Wikis und der Datenintegration auf unterschiedlichen Ebenen untersucht werden, um darauf aufbauend ein System zu entwickeln, das die gewonnenen Ergebnisse umsetzt. Der Bearbeitung des Themas vorangestellt, sollen zunächst Motivation und Ziele der Arbeit beschrieben werden. Zudem wird der Aufbau der übrigen Kapitel erläutert.

1.1. Motivation

Das heutige Web hält eine enorme Menge an strukturierten Informationen bereit. Ein großer Teil dieser Daten wird von den jeweiligen Diensteanbietern in proprietären Formaten gehalten und den Nutzern nur innerhalb des eigenen Dienstes zur Verfügung gestellt. Die so entstehenden „Daten-Silos“ sind in den meisten Fällen gegeneinander abgeschottet, so dass die Informationen aus einem Dienst, nicht mit denen eines anderen Dienstes verknüpft werden können. Ein einfaches Beispiel dafür lässt sich mit Hilfe der diversen sozialen Netzwerke, wie z. B. Facebook[1] oder MySpace[2], konstruieren. Unabhängig davon, welchen Zweck ein beliebiger Anbieter eines Dienstes im Web verfolgt, wird in den meisten Fällen eine Funktion integriert, die es erlaubt, soziale Kontakte aufzubauen und zu pflegen. Es wird nicht deutlich, wie diese Daten intern dargestellt werden, obwohl die sozialen Netze der meisten Anbieter im Kern die gleichen Eigenschaften besitzen. Dadurch besteht der soziale Graph eines jeden Nutzers aus diversen Teilgraphen, die untereinander nicht verbunden sind und ohne weitere Bemühungen nicht vereinbar sind. Um diesem Problem zu begegnen, stellen immer mehr Anbieter Programmierschnittstellen bereit. Diese Tatsache ändert nichts an der Inkompatibilität der Daten verschiedener Dienste, auf diese Weise wird es aber möglich, auf Daten in strukturerhaltener Weise zuzugreifen. Auf der anderen Seite schreiten auch die Bemühungen im Semantic Web Bereich stetig voran. In den letzten Jahren sind diverse

[1] http://www.facebook.com/
[2] http://www.myspace.com/

Datenquellen entstanden, die Informationen in einer einheitlichen Sprache, dem Resource Description Framework (RDF) über standardisierte Schnittstellen bereitstellen. Zusätzlich etablieren sich RDF-basierte Vokabulare, die Konzepte der sozialen Netzwerke modellieren und vereinheitlichen. Somit werden die Voraussetzungen für eine Ablösung proprietärer Datenformate bzw. deren Abbildung in geeigneter Art und Weise, geschaffen. Die Motivation dieser Arbeit liegt in der entstehenden Notwendigkeit, die diversen strukturierten Informationen zu integrieren, um einen echten Mehrwert für den Nutzer zu schaffen.

1.2. Ziele

Ziel dieser Arbeit soll die Beleuchtung der unterschiedlichen Facetten der Datenintegration auf verschiedenen Ebenen sein. Daraus ergeben sich Anforderungen an ein System, das die Integration von Informationen in geeigneter Weise ermöglicht. Um die Semantic Web Vision zu unterstützen, soll dieses System sowohl Daten konsumieren, als auch strukturierte Informationen produzieren. Das führt zu weiteren Anforderungen, die es in einem nächsten Schritt zu erfüllen gilt. Es soll ein System spezifiziert werden, das in ein semantisches Daten-Wiki integriert werden kann. Abschließendes Ziel dieser Arbeit, ist die prototypische Implementierung der ausgearbeiteten Komponenten auf Basis von OntoWiki.

1.3. Struktur der Arbeit

Der Aufbau der folgenden Kapitel ist eng an die Ziele dieser Arbeit angelehnt. Im zweiten Kapitel werden Grundlagen vermittelt, die für ein vollständiges Verständnis der weiteren Kapitel notwendig sind. Das dritte Kapitel untersucht die Anforderungen an ein zu entwickelndes System, die sich aus den verschiedenen Aspekten heraus ergeben. Diesbezüglich werden zunächst aussagekräftige Anwendungsfälle skizziert, aus denen sich die Anforderungen ableiten lassen. Anschließend werden die funktionalen und nichtfunktionalen Anforderungen erläutert. Das vierte Kapitel beleuchtet die heutige Situation in Bezug auf das Thema. Hierbei werden Forschungsergebnisse, Werkzeuge und Dienste vorgestellt, sowie auf ihre Funktionalität das Thema betreffend untersucht. In den darauf folgenden Kapiteln wird das zu entwickelnde System spezifiziert und implementiert. Schließlich werden die Ergebnisse zusammengefasst und das entstandene System evaluiert, bevor abschließend ein kurzer Ausblick auf mögliche zukünftige Entwicklungen gegeben wird.

2. Grundlagen

Im folgenden Kapitel werden die grundlegenden Konzepte und Technologien vorgestellt, auf denen die vorliegende Arbeit basiert und deren Verständnis deshalb in den übrigen Kapiteln vorausgesetzt wird. Der erste Abschnitt beschreibt die Vision des Semantic Web und beleuchtet die damit verbundenen Konzepte und Basistechnologien. Zwischen dieser Vision und dem ursprünglichen Web ist das heutige „Mitmach"-Internet angesiedelt, das für gewöhnlich mit dem Schlagwort Web 2.0 umschrieben wird. Mit fortschreitender Vereinheitlichung der Datenrepräsentation auf Basis von Semantic Web Technologien, wurde auch der Terminus des Social Semantic Web geprägt. Die mit diesen Begriffen in Zusammenhang stehenden Konzepte werden ebenfalls erläutert. Schließlich erfolgt eine Vorstellung der Idee des Wikis, sowie der zusätzlichen Eigenschaften von semantischen Wikis und Daten-Wikis.

2.1. Semantic Web / Data Web

Die Vision des Semantic Web oder Data Web, beschreibt kein neuartiges Web, sondern eine Erweiterung zum ursprünglichen Netz der Dokumente. Tim Berners-Lee beschreibt dies in [Berners-Lee et al., 2001] wie folgt:

> *„The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation."*

Das klassische Web, auch World Wide Web oder WWW genannt, wurde 1989 am Conseil Européen pour la Recherche Nucléaire (CERN) entwickelt, mit dem Zweck, wissenschaftliche Dokumente auf einfache Art und Weise austauschen zu können. Daraus entstand binnen weniger Jahre ein Netz, bestehend aus einer unüberschaubaren Menge an Dokumenten, die miteinander über Links verbunden sind. Diese Dokumente enthalten große Datenmengen, die allerdings größtenteils nur von Menschen verarbeitet werden können, da es sich um freien Text handelt, dessen Bedeutung eine Maschine nicht ohne Weiteres erfassen kann.

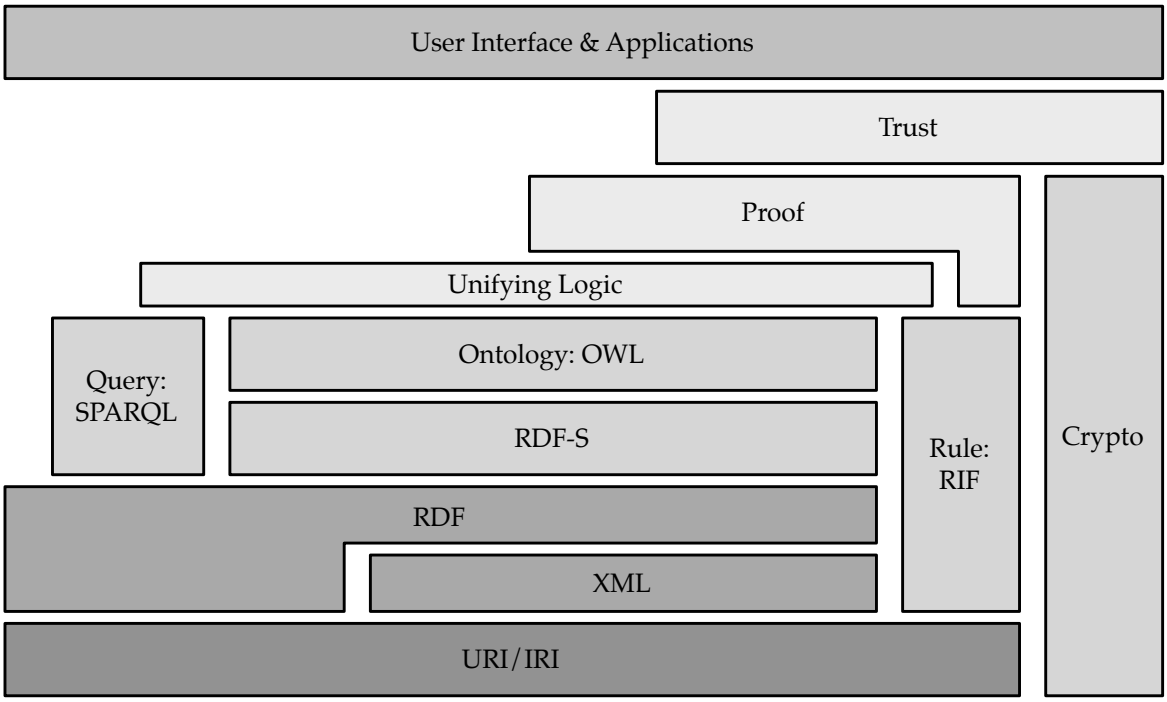

Abbildung 2.1.: Das Semantic Web Schichtenmodell

Die Idee des Semantic Web versucht sich dieses Problems anzunehmen und das bestehende Netz dahingehend zu erweitern, dass Informationen sowohl für Menschen, als auch für Maschinen interpretierbar werden. Auf diese Weise wird eine riesige Datenbasis geschaffen, die durch die Kombination vieler unabhängiger Datenquellen entsteht. Diese Datenquellen können beliebig kombiniert und wie Datenbanken abgefragt werden. In diesem Zusammenhang wird häufig der Begriff des Data Web verwendet. Abbildung 2.1 zeigt das Semantic Web Schichtenmodell, das ursprünglich in [Berners-Lee et al., 2006b] eingeführt und seitdem stetig weiterentwickelt wurde. Im Folgenden werden die für diese Arbeit relevanten Schichten erläutert.

Uniform Resource Identifier

Uniform Resource Identifier oder kurz URIs sind, wie es der Name andeutet, einheitliche Bezeichner, für sowohl physische, als auch abstrakte Ressourcen. URIs, deren generische Syntax in [Berners-Lee et al., 2005] definiert wird, werden insbesondere in Zusammenhang mit den wohl bekanntesten Internet-Diensten, dem WWW und Email eingesetzt. Im WWW

kommen URIs zum Einsatz, um Dokumente weltweit eindeutig zu referenzieren und werden in diesem Kontext auch als Uniform Resource Locator (URLs) bezeichnet. Beim Email-Dienst werden URIs, sog. `mailto`-URIs verwendet, um die Quelle und das Ziel von Nachrichten zu bestimmen. Eine gültige URI hat die Form `scheme:rest`, wobei `scheme` für ein Schema steht und `rest` durch das verwendete Schema bestimmt wird. Beispiele für gebräuchliche Schemata sind `http` und `ftp`. Ein Doppelpunkt trennt in allen URIs den Schema-Teil vom restlichen Teil der URI. Die URI

```
http://www.frischmuth24.de/diplomarbeit.pdf
```

bspw. macht Gebrauch vom `http`-Schema. Aus dieser Information lässt sich ableiten, dass das Hypertext Transfer Protocol (HTTP) genutzt wird, um diese Ressource anzusprechen. Des Weiteren enthält die URI die Information, wo sich die gewünschte Ressource befindet. Da URIs häufig eingesetzt werden, um Ressourcen eindeutig zu identifizieren, werden sie in diesem Zusammenhang als URI-Referenzen bezeichnet. Eine URI-Referenz kann absolut oder relativ sein und optional ein zusätzliches Fragment enthalten, das durch ein Trennzeichen (#) mit der URI-Referenz verbunden ist. Durch das Auflösen von relativen URI-Referenzen und das Entfernen eines evtl. vorhandenen Fragments, entsteht eine absolute URI, die bspw. genutzt werden kann, um ein Dokument abzurufen. Es sei darauf hingewiesen, dass im weiteren Verlauf der Arbeit nicht zwischen URIs und URI-Referenzen unterschieden wird, sondern einheitlich der Begriff URI verwendet wird.

Im Semantic Web Kontext dienen URIs vor allem der weltweit eindeutigen Bezeichnung von abstrakten Konzepten. Beispiele für solche abstrakten Ressourcen sind Personen, Institutionen oder Publikationen, aber auch Beziehungen, die zwischen solchen Ressourcen bestehen, wie z. B. eine Autoreneigenschaft oder eine Arbeitgeber-Arbeitnehmer-Beziehung.

Resource Description Framework

RDF gehört zu den zentralen Standards im Semantic Web und die Anordnung von RDF im Schichtenmodell direkt über URI, unterstreicht dessen Relevanz. RDF ist eine Sprache zur Darstellung von Informationen über Ressourcen [Manola and Miller, 2004]. Im Jahre 1999 wurde eine erste W3C-Empfehlung für RDF veröffentlicht [Lassilla and Swick, 1999], wobei in dieser Version der Fokus auf der Beschreibung von Web-Dokumenten lag. Im Zuge der Bemühungen des World Wide Web Konsortiums (W3C) zum Thema Semantic Web, wurde

die RDF-Spezifikation überarbeitet und 2004 eine Reihe neuer Spezifikationen, sog. W3C-Recommendations veröffentlicht [Manola and Miller, 2004; Klyne and Carroll, 2004; Beckett, 2004; Hayes, 2004]. Es existieren heute sowohl zahlreiche Systeme, die RDF-Daten verwalten können, sog. Triple Stores, als auch Bibliotheken zur Verarbeitung von RDF-Daten für alle gängigen Programmiersprachen.

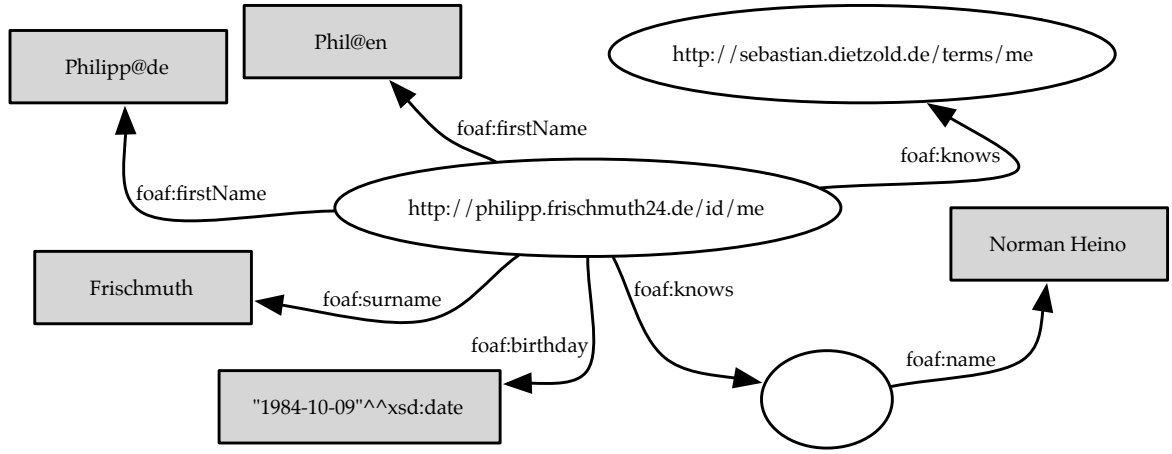

Abbildung 2.2.: Ein einfacher RDF-Graph mit URIs (Ovale), Literalen (Rechtecke) und einem Blank Node (leeres Oval)

Im Gegensatz zur Extensible Markup Language (XML), die Daten hierarchisch anordnet, stellt RDF Daten als Graph dar. Zum Datenaustausch kann ein solcher Graph in ein auf XML basierendes Format überführt werden. So sieht es zumindest die Spezifikation vor, wenngleich eine Reihe alternativer Darstellungsformen existieren, wie z. B. Notation3 (N3, Berners-Lee and Connolly [2008]) oder die Terse RDF Triple Language (Turtle, Beckett and Berners-Lee [2008]), die eine für Menschen lesbarere Syntax einsetzen. Zusätzlich zu den erwähnten Konzepten werden in RDF Literale verwendet, die mit einem Datentyp oder einer Sprache belegt werden können. Des Weiteren unterstützt RDF einfache Schlussfolgerungen. Das Kernkonzept, das RDF verwendet, ist das der Aussage, auch als Statement oder Tripel bezeichnet. Im weiteren Verlauf der Arbeit, werden die Begriffe Statement und Tripel ausschließlich für das Konzept der RDF-Aussage verwendet. Ein Statement besteht aus drei Teilen, einem Subjekt, einem Prädikat und einem Objekt. Ein Subjekt wird stets aus einer Ressource oder einem sog. Blank Node gebildet, wobei Ressourcen durch URIs benannt werden und somit weltweit eindeutig referenzierbar sind. Blank Nodes hingegen, besitzen lediglich eine lokale Gültigkeit. Sie werden eingesetzt, da für die Repräsentation bestimmter

Sachverhalte, z. B. Listen, in RDF Hilfskonstrukte benötigt werden und die Verwendung von URIs für sämtliche solcher Hilfsknoten unpraktisch erscheint. Im weiteren Verlauf werden Ressourcen und Blank Nodes auch als benannte bzw. unbenannte Ressourcen bezeichnet. Ein Prädikat wird stets aus einer URI gebildet, d. h. Blank Nodes sind hier nicht erlaubt. Das Prädikat einer Aussage bestimmt dessen Eigenschaft und solche Ressourcen werden deshalb häufig mit dem Begriff Property bezeichnet. Ein Objekt kann sowohl durch eine Ressource oder einen Blank Node, als auch durch ein Literal gebildet werden. Literale sind einfache Unicode-kodierte Zeichenketten. Es werden zwei Arten unterschieden, einfache Literale und typisierte Literale.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">

  <rdf:Description rdf:about="http://philipp.frischmuth24.de/id/me">
    <foaf:firstName xml:lang="de">Philipp</foaf:firstName>
    <foaf:firstName xml:lang="en">Phil</foaf:firstName>
    <foaf:surname>Frischmuth</foaf:surname>
    <foaf:birthday rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
      1984-10-09
    </foaf:birthday>
    <foaf:knows rdf:resource="http://sebastian.dietzold.de/terms/me" />
    <foaf:knows>
      <rdf:Description rdf:nodeID="bnode1">
        <foaf:name>Norman Heino</foaf:name>
      </rdf:Description>
    </foaf:knows>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.1: RDF/XML-Serialisierung des Beispielgraphen aus Abbildung 2.2 (Seite 6)

Jedes Statement setzt in einem RDF-Graphen einen Knoten (Subjekt) über eine gerichtete Kante (Prädikat) mit einem anderen Knoten (Objekt) in Beziehung. Der Ziel-Knoten kann, insofern es sich um eine Ressource oder einen Blank Node handelt, in weiteren Aussagen in einer anderen Rolle auftreten, bspw. als Subjekt. Gleiches gilt für den Ausgangs-Knoten und die Eigenschafts-Kante. Lediglich Literale stellen „Endpunkte“ dar, in der Weise, dass sie ausschließlich eingehende Kanten besitzen. Eigenschaften, die eine Ressource mit einer anderen Ressource verbinden, werden auch als Relation bezeichnet. Im weiteren Verlauf

dieser Arbeit wird der Begriff Relation genau für solche Eigenschaften verwendet.

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .

<http://philipp.frischmuth24.de/id/me>
  foaf:firstName "Philipp"@de ;
  foaf:firstName "Phil"@en ;
  foaf:surname "Frischmuth" ;
  foaf:birthday "1984-10-09"^^<http://www.w3.org/2001/XMLSchema#date> ;
  foaf:knows <http://sebastian.dietzold.de/terms/me> ;
  foaf:knows [
    foaf:name "Norman Heino"
  ] .
```

Listing 2.2: Turtle-Serialisierung des Beispielgraphen aus Abbildung 2.2 (Seite 6)

Abbildung 2.2 visualisiert einen einfachen RDF-Graphen. In diesem Beispiel sind einer benannten Ressource Vornamen in unterschiedlichen Sprachen, ein Nachname und ein Geburtsdatum mit einem speziellen Datentyp zugeordnet. Diese Aussagen enden allesamt in einem Literal. Zusätzlich wird modelliert, dass die Entität zwei weitere Ressourcen „kennt“, wobei eine davon durch eine URI benannt ist, während die andere über einen Blank Node identifiziert wird. Listing 2.1 stellt den beschriebenen Graphen serialisiert in der RDF/XML-Syntax dar, Listing 2.2 zeigt die Serialisierung in der Turtle-Syntax. Aufgrund der besseren Lesbarkeit gelangt im weiteren Verlauf der Arbeit die Turtle-Syntax zum Einsatz. Um den Leser der Arbeit mit den wesentlichen Informationen zu versorgen, werden die in Listing 2.2 verwendeten Präfixe verwendet, ohne diese erneut zu definieren.

RDF Schema

RDF Schema (RDFS) ist eine auf RDF basierende Beschreibungssprache für Vokabulare [Brickley and Guha, 2004]. Aus diesem Grund kann RDFS selbst als Vokabular betrachtet werden. RDF ermöglicht es, Ressourcen über Eigenschaften mit Werten zu belegen oder über Relationen mit anderen Ressourcen zu verknüpfen. Es existiert jedoch keine Möglichkeit, diese Eigenschaften zu beschreiben und zu anderen Eigenschaften in Beziehung zu setzen.

Zu diesem Zweck wurde RDFS eingeführt. Sie definiert ein allgemeingültiges Vokabular, um domänenspezifische Vokabulare erstellen zu können, deren Termini eine definierte Bedeutung besitzen. RDFS ist somit eine Sprache, die eine Repräsentation von Wissen in einer formalen Art und Weise ermöglicht. Die Ausdrucksmächtigkeit ist jedoch beschränkt und deshalb wird RDFS auch als leichtgewichtige Ontologiesprache bezeichnet [Hitzler et al., 2008]. Im Folgenden sollen die primären Bestandteile des RDFS-Vokabulars kurz beschrieben und zueinander in Beziehung gesetzt werden.

Zunächst definiert RDFS den Term `rdfs:Resource` als die Gesamtheit aller mit RDF beschreibbaren Ressourcen. Solche Gesamtheiten können als Instanz der Klasse `rdfs:Class` deklariert werden, wobei `rdfs:Class` als Instanz von sich selbst definiert ist. Die Typisierung einer Ressource ist bereits in RDF mit der `rdf:type` Relation vorgesehen und von dieser Möglichkeit wird in RDFS Gebrauch gemacht. Da auch `rdfs:Class` eine Ressource ist, ergibt sich, dass diese wiederum als Instanz der Klasse `rdfs:Ressource` angesehen werden muss. Des Weiteren definiert RDFS u. a. die Klasse aller Literale (`rdfs:Literal`) und die Klasse aller Eigenschaften (`rdf:Property`).

Auf Grundlage dieser Klassendefinitionen führt RDFS eine Reihe nützlicher Eigenschaften und Relationen ein, die allesamt Instanzen von `rdf:Property` sind und von denen einige ausgewählte an dieser Stelle kurz beschrieben werden sollen.

- `rdfs:subClassOf` und `rdfs:subPropertyOf` erlauben es, Hierarchien von Klassen bzw. Eigenschaften zu erzeugen. Dadurch wird es bspw. möglich, mit RDFS Taxonomien abzubilden.
- `rdf:type` wird explizit als `rdf:Property` definiert.
- `rdfs:domain` und `rdfs:range` ermöglichen es, Definitions- und Wertebereiche für Eigenschaften und Relationen festzulegen.
- Mit `rdfs:label` und `rdfs:comment` lassen sich Bezeichner und Beschreibungen für Ressourcen erzeugen, die für den menschlichen Betrachter bestimmt sind.
- `rdfs:seeAlso` und `rdfs:isDefinedBy` können verwendet werden, um auf weitere Beschreibungen und Definitionen von Ressourcen zu verweisen.

Web Ontology Language

RDFS lässt sich für die Modellierung einfacher Sachverhalte verwenden. In engen Grenzen kann auch implizites Wissen aus solchen Wissensbasen gewonnen werden. Für die Model-

lierung komplexerer Sachverhalte ist die Ausdrucksmächtigkeit von RDFS jedoch zu gering. Aus diesem Grund wurde eine ausdrucksstarke und formale Ontologiesprache benötigt. Der Begriff Ontologie bezeichnet im Kontext des Semantic Web die explizite und formale Spezifikation einer Konzeptualisierung [Gruber, 1993]. Aus diesen Anforderungen ist die Web Ontology Language (OWL) hervorgegangen, die im Jahre 2004 als W3C-Empfehlung veröffentlicht wurde [McGuinness and van Harmelen, 2004; Patel-Schneider et al., 2004]. Sie basiert auf RDFS und somit auch auf RDF, bietet eine formale Semantik und ist in drei Untersprachen aufgeteilt, die jeweils eine unterschiedliche Ausdrucksmächtigkeit besitzen.

OWL Full umfasst den gesamten Sprachumfang und ist deshalb sehr ausdrucksstark. Sie ist jedoch unentscheidbar.

OWL DL ist eine echte Teilsprache von OWL Full. Sie ist entscheidbar und nutzt das vollständige OWL-Vokabular. Um die Entscheidbarkeit zu wahren, wurden jedoch einige Einschränkungen definiert. Bspw. kann eine Klasse nicht gleichzeitig als Individuum oder Eigenschaft auftreten.

OWL Lite ist eine echte Teilsprache von OWL DL. Sie ist ebenfalls entscheidbar, greift dabei aber nur auf einen Teil der Möglichkeiten zurück, die OWL bietet.

In OWL können wie in RDFS Klassen definiert werden. Für die explizite Benennung einer Klasse führt OWL den Term `owl:Class` ein. Es sei jedoch angemerkt, dass in OWL im Gegensatz zu RDFS explizit zwischen Instanzen, hier Individuen genannt, und Klassen unterschieden wird. Auch lassen sich Klassenhierarchien mittels der bekannten `rdfs:subClassOf`-Relation erzeugen. Zusätzlich können in OWL aber auch komplexe Klassen, Eigenschaften und Relationen konstruiert werden. Für die Definition einer komplexen Klasse lassen sich z. B. die von OWL eingeführten Konzepte `owl:unionOf` oder `owl:intersectionOf` nutzen. In OWL können dedizierte Klassen für Eigenschaften (`owl:DatatypeProperty`) und Relationen (`owl:ObjectProperty`) verwendet und darüber hinaus komplexe Beziehungen definiert werden. So lässt sich bspw. ausdrücken, dass eine Relation in beide Richtungen gilt (`owl:SymmetricProperty`) oder dass der Wert einer Eigenschaft für eine Instanz (`owl:FunctionalProperty`) oder alle Instanzen (`owl:InverseFunctionalProperty`) eindeutig ist. Eine weitere Eigenschaft von OWL, die auch für diese Arbeit relevant sein wird, ist die Möglichkeit verschiedene, auf OWL basierende Wissensbasen zu vereinigen und somit einen kombinierten RDF-Graphen zu erzeugen. Zu diesem Zweck kommt die Relation `owl:imports` zum Einsatz.

SPARQL Protokoll und Anfragesprache

Der Begriff SPARQL, ein rekursives Akronym, steht für SPARQL Protocol and RDF Query Language und bezeichnet sowohl eine Anfragesprache für RDF-Daten, als auch ein abstraktes Protokoll, um entfernte SPARQL-Endpunkte über ein zu bestimmendes Protokoll anzufragen. SPARQL besteht aus drei konkreten Spezifikationen, die allesamt den Status einer W3C-Empfehlung besitzen und somit als Web-Standards gelten.

Die **SPARQL Anfragesprache** [Prud'hommeaux and Seaborne, 2008] basiert auf einfachen Graphmustern (Basic Graph Pattern), die zu komplexen Graphmustern (Group Graph Pattern) zusammengefügt werden können. Ein Basic Graph Pattern besteht aus einem oder mehreren sog. Triple Pattern, die zusätzlich um Filterausdrücke ergänzt werden können. Ein einzelnes Triple Pattern besteht wie ein Statement aus Subjekt, Prädikat und Objekt, wobei alle drei Teile durch eine Variable ersetzt werden können. Variablen wiederum werden in SPARQL durch ein `?` oder `$`, gefolgt von einem Variablennamen, gekennzeichnet. Group Graph Pattern gruppieren einzelne Basic Graph Pattern durch den Einsatz von geschweiften Klammern (`{}`). Mehrere solcher Muster lassen sich z. B. durch den Einsatz des Schlüsselwortes `UNION` kombinieren.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name ?nickname
WHERE {
    <http://philipp.frischmuth24.de/id/me> foaf:knows ?x .
    ?x foaf:name ?name
    OPTIONAL { ?x foaf:nick ?nickname }
}
```

Listing 2.3: Einfache SPARQL-Anfrage mit eindeutigen Ergebnistupeln und einem optionalen Bestandteil

SPARQL unterstützt eine Reihe von Funktionen, die für eine Filterung eingesetzt werden können. So ist es bspw. möglich, reguläre Ausdrücke auf Ressourcen und Literale anzuwenden (`regex`) oder zu prüfen, ob eine bestimmte Variable an einen Wert gebunden ist (`bound`). Des Weiteren können über das `OPTIONAL`-Schlüsselwort Muster angegeben werden, die optional von der Anfrage erfüllt werden können. Sollte ein Ergebnis den als optional deklarierten Teil nicht erfüllen, führt dies nicht zum Ausschluss des Tupels aus der Ergebnismenge. Eine weitere Eigenschaft von SPARQL ist die Unterstützung von unterschiedlichen Ergebnisformaten, deren Verwendung durch den Einsatz spezieller Schlüsselwörter zu Be-

ginn der Anfrage markiert wird. Mit SELECT werden, wie in der Structured Query Language (SQL), bestimmte Variablen selektiert und die einzelnen Ergebnistupel in einer Liste angeordnet. Über das ASK-Schlüsselwort lässt sich prüfen, ob mindestens ein Statement existiert, das auf die spezifizierte Anfrage zutrifft. Mit CONSTRUCT lassen sich beliebige RDF-Graphen konstruieren und DESCRIBE liefert die Beschreibung einer Ressource, wobei nicht definiert ist, was als Beschreibung einer Ressource gilt. Weitere wichtige Funktionen von SPARQL sind die Nutzung unterschiedlicher Graphen für Anfragen (FROM und FROM NAMED), die Sortierung der Ergebnisse (ORDER BY), die Auswahl von Ergebnisbereichen mit LIMIT und OFFSET und die Eliminierung von doppelten Ergebnissen (DISTINCT). Listing 2.3 zeigt eine einfache SPARQL-Anfrage, Listing 2.4 ein mögliches Ergebnis dieser Anfrage.

```
name                  nickname
-----------------------------------
Jens Lehmann
Norman Heino
Sebastian Dietzold    Seebi
Soeren Auer
Thomas Riechert
```

Listing 2.4: Mögliches Ergebnis der SPARQL-Anfrage aus Listing 2.3

Das **SPARQL Protokoll** [Clark et al., 2008] definiert auf abstrakter Ebene eine Möglichkeit, um in standardisierter Art und Weise entfernte SPARQL-Prozessoren anfragbar zu machen. Das Protokoll spezifiziert diesbezüglich die query-Operation, die aus zwei Fehlertypen (MalformedQuery und QueryRequestRefused) und zwei Nachrichtenarten besteht, je eine für Anfrage und Antwort. Für Anfragen wird ein Parameter eingeführt, der die SPARQL-Anfrage enthält. Des Weiteren werden zwei optionale Parameter definiert, um die zu verwendenden Graphen zu bestimmen. Antworten bestehen je nach Beschaffenheit der SPARQL-Anfrage aus einem RDF-Dokument (DESCRIBE und CONSTRUCT) oder einem speziellen XML-Dokument für SPARQL-Ergebnisse, das im folgenden Abschnitt kurz vorgestellt wird. Das SPARQL-Protokoll spezifiziert nicht nur ein abstraktes Protokoll für die Anfrage von entfernten SPARQL-Endpunkten, sondern auch konkrete Umsetzungen dieses Protokolls für HTTP und das Simple Object Access Protocol (SOAP).

In Verbindung mit dem Protokoll wurde das **SPARQL Results XML Format** [Beckett and Broekstra, 2008] eingeführt. Es definiert ein XML-basiertes Format für SELECT- und ASK-Anfragen. Jedes so formulierte Ergebnis besteht aus einem Kopf (<head>), der die im Ergebnis verwendeten Variablen angibt und optional eine Referenz auf ein Dokument mit Metadaten

zur gestellten Anfrage enthält. Daneben besteht es aus einem Ergebnisteil (<results>), das für jedes Ergebenistupel (<result>) die Variablenbindungen (<binding>) angibt. Für ASK-Anfragen besteht der Ergebnisteil lediglich aus dem boolschen Ergebnis der Anfrage (<boolean>).

Linked Data

Der Begriff Linked Data stellt eine alternative Benennung des Semantic Web dar und macht gleichzeitig deutlich, dass es nicht genügt, Daten in irgendeiner Form verfügbar zu machen. Vielmehr ist es notwendig, solche Daten mit anderen verfügbaren Daten zu verknüpfen [Berners-Lee, 2006]. Tim Berners-Lee stellt in diesem Zusammenhang vier grundlegende Schritte für die Bereitstellung von Daten vor, die er als Regeln bezeichnet.

1. Es sollten URIs verwendet werden, wenn die Benennung von Ressourcen erforderlich ist.
2. Es sollte Gebrauch von HTTP-URIs gemacht werden, um Namen über ein einheitliches Protokoll abrufbar zu machen.
3. Beim Aufruf einer URI sollten nützliche Informationen über die Ressource bereitgestellt werden, basierend auf Semantic Web Standards.
4. Wenn möglich, sollten Referenzen auf andere URIs in die Daten integriert werden, damit diese ebenfalls abgerufen werden können.

In den letzten Jahren hat der Begriff Linked Data stark an Bedeutung gewonnen, u. a. durch das Linking Open Data Projekt[3], das die Verlinkung umfangreicher Datenquellen mit Hilfe der Linked Data Prinzipien anstrebt. Im Mai 2009 bestand die sog. LOD data cloud bereits aus über 4,7 Milliarden RDF-Tripeln und ca. 142 Millionen Links zwischen verschiedenen Datenquellen. Dies wird dadurch erreicht, dass große Daten-Provider, wie z. B. DBpedia[4] oder Geonames[5] ihre Daten als Linked Data bereitstellen und zusätzlich auf URIs aus anderen Datenquellen verweisen. Im weiteren Verlauf dieser Arbeit wird Linked Data ebenfalls eine wichtige Rolle einnehmen.

[3] http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[4] http://dbpedia.org/
[5] http://www.geonames.org/

2.2. Web 2.0 / Social Web

Das ursprüngliche Web wurde häufig als eine riesige Ansammlung von statischen, untereinander verlinkten Hypertext-Dokumenten betrachtet. Diese Charakterisierung mag für die frühen Jahre des Internets korrekt gewesen sein, heute ist das Web jedoch weitaus dynamischer. Viele der gegenwärtig existierenden Webseiten sind vielmehr interaktive Plattformen, die es den Nutzern ermöglichen, soziale Kontakte zu pflegen, Informationen auszutauschen und Inhalte eigenständig zu generieren. In diesem Zusammenhang wird sehr häufig von einem sozialen Netz, einem Social Web gesprochen, da bei den meisten Diensten die menschliche Kommunikation im Vordergrund steht oder zumindest gefördert wird. Sehr viel häufiger wird aber der Begriff Web 2.0 verwendet, ein Schlagwort, das in den letzten Jahren einen sehr hohen Bekanntheitsgrad erlangt hat. Es wird darüber debattiert, wann der Begriff das erste Mal erwähnt wurde, jedoch steht fest, dass er mit der Veröffentlichung des Artikels „What is Web 2.0“ [O'Reilly, 2005] von Tim O'Reilly einer breiten Öffentlichkeit zugänglich gemacht wurde. Das Schlagwort Web 2.0 ist an die Versionierung von Softwaresystemen angelehnt und steht somit für eine vollständig neue oder zumindest generalüberholte Version. Aus diesem Grund ist der Begriff nicht unumstritten. In einem Interview mit IBM[6] äußert sich Tim Berners-Lee bspw. wie folgt über den Begriff:

> *„[...] I think Web 2.0 is of course a piece of jargon, nobody even knows what it means. If Web 2.0 for you is blogs and wikis, then that is people to people. But that was what the Web was supposed to be all along.“*

Auch das Web in seiner ursprünglichen Form war dazu gedacht, Menschen miteinander zu verbinden. Die heute so populären Anwendungen basieren auf den gleichen Standards, die für das Web geschaffen wurden. Unumstritten ist jedoch, dass das moderne Web eine neue Philosophie mit sich gebracht hat, die den Nutzer in den Mittelpunkt rückt und eine aktive Beteiligung von diesem fordert. Zusätzlich sind durch die meist kostenlosen Dienste neue, lukrative Geschäftsmodelle entstanden, die oft auf der Platzierung von Werbeinhalten basieren. Im Folgenden sollen die wichtigsten technologischen Änderungen kurz vorgestellt werden.

[6] `http://www.ibm.com/developerworks/podcast/dwi/cm-int082206.txt`

Dynamische Web-Anwendungen mit AJAX

Der Begriff AJAX steht für Asynchronous JavaScript and XML und ist im eigentlichen Sinne eine Kombination aus bestehenden Technologien. Zum Einen ist das JavaScript, eine browserseitige Skriptsprache, die von allen modernen Webbrowsern unterstützt wird. Zum Anderen ist es das `XMLHttpRequest`-Objekt, das von JavaScript zur Verfügung gestellt wird und asynchrone HTTP-Anfragen ermöglicht. So wird es möglich, dass Anfragen an einen Webserver im Hintergrund gestellt werden können, ohne die Applikation dabei zu blockieren. Als Datenaustausch-Format zwischen Browser und Server kommt XML zum Einsatz, da diese sowohl auf der Seite des Servers, als auch innerhalb von JavaScript verarbeitet werden kann. Mit diesen Technologien ist es möglich, sehr dynamische Web-Anwendungen zu realisieren, die sich in ihrer Handhabbarkeit wie moderne Desktop-Anwendungen verhalten.

JSON statt XML

Da die Verarbeitung von XML-Daten mitunter umständlich ist und die Behandlung dieser Daten in JavaScript erfolgt, wird zum Datenaustausch häufig JSON verwendet. JSON steht für JavaScript Object Notation und ist ein leichtgewichtiges, textbasiertes Daten-Austauschformat [Crockford, 2006]. Obwohl der Name es suggeriert, ist JSON nicht auf JavaScript beschränkt, sondern generell sprachunabhängig. Es existieren für die meisten Programmiersprachen JSON-Schnittstellen, u. a. für PHP, Java und C.

REST-konforme Programmierschnittstellen

Viele bekannte Dienste im Web 2.0 stellen Programmierschnittstellen (Application Programming Interfaces, APIs) auf Basis von REST bereit. REST bezeichnet einen Architekturstil für auf Hypertext basierende Systeme und steht für Representational State Transfer [Fielding, 2000]. In einer REST-Architektur werden alle Daten, die zugänglich gemacht werden sollen, über HTTP-URIs identifiziert. REST-basierte Dienste zeichnen sich durch Zustandslosigkeit aus, d. h. alle für eine Anfrage notwendigen Informationen sind in der URI kodiert.

Im folgenden Abschnitt soll der Begriff des Social Semantic Web eingeführt werden, das seinerseits auf den Errungenschaften des Web 2.0 aufbaut und zugleich semantische Technologien einsetzt.

2.3. Social Semantic Web

Das Web 2.0 hat eine stetig wachsende Menge strukturierter Informationen hervorgebracht. Teile dieser Informationen sind über APIs abrufbar, meistens jedoch mit Hilfe von proprietären Schnittstellen, die untereinander inkompatibel sind. Mit der wachsenden Adaption von semantischen Technologien, zeichnet sich nun ein Trend dahingehend ab, dass versucht wird, diese Web 2.0 „Daten-Inseln“ strukturiert zu erschließen [Blumauer and Pellegrini, 2009]. Dies wird einerseits durch die Entwicklung geeigneter Vokabulare ermöglicht, die in der Lage sind, die vielfältigen sozialen Informationen in einer homogenen Art und Weise darzustellen. Andererseits werden Werkzeuge geschaffen, die Gebrauch von diesen Vokabularen machen. Mit der fortschreitenden Annäherung der beiden Welten Web 2.0 und Semantic Web wurde auch der Begriff des Social Semantic Web geprägt. Zwei repräsentative Vokabulare für die Repräsentation von Daten in diesem Kontext sollen im Folgenden kurz vorgestellt werden.

Friend of a Friend

Das Friend of a Friend (FOAF) Vokabular [Brickley and Miller, 2007] wurde entwickelt, um Personen zu beschreiben und diese mit ihrer Umgebung in Relation zu setzen. Das FOAF-Vokabular wird bereits in vielen Bereichen eingesetzt und auch in dieser Arbeit wird es von zentraler Bedeutung sein. Es ist eine gängige Praxis, für die Öffentlichkeit bestimmte personenbezogene Daten und Informationen über Beziehungen zu anderen Personen in einer FOAF-Datei auf einem Server zu platzieren und so der Allgemeinheit verfügbar zu machen. Zusätzlich setzen bereits einige bekannte Web-Dienste, wie z. B. LiveJournal[7] FOAF ein, um Nutzerinformationen zu veröffentlichen. Das Beispiel aus Abbildung 2.2 (Seite 6) verwendet ebenfalls das FOAF-Vokabular, um eine Person zu beschreiben und dessen Verbindung zu anderen Personen darzustellen.

Semantically-Interlinked Online Communities

Eine zweite wichtige Initiative in dieser Richtung ist das SIOC-Projekt. SIOC steht für Semantically-Interlinked Online Communities und bietet ein RDF-Vokabular, um diverse Konzepte, die in Online-Gemeinschaften auftreten, in RDF formulieren zu können [Breslin

[7] http://www.livejournal.com/

```
<http://philipp.frischmuth24.de/id/me> a sioc:User ;
  foaf:knows <http://sebastian.dietzold.de/terms/me> .

:blogPost1 a sioc:BlogPost ;
  sioc:has_creator <http://sebastian.dietzold.de/terms/me> ;
  sioc:content "Ich blogge..." ;
  sioc:has_reply :comment1 .

:comment1 a sioc:Comment ;
  sioc:has_creator <http://philipp.frischmuth24.de/id/me> ;
  sioc:content "Wahnsinn!"
  sioc:reply_of :blogPost1 .
```

Listing 2.5: Beispiel für einen mit FOAF und SIOC modellierten Sachverhalt

et al., 2005]. Es nutzt bestehende Konzepte aus anderen verbreiteten Vokabularen, wie bspw. FOAF und erweitert diese. Beispiele für SIOC-Konzepte sind `sioc:User`, eine Subklasse von `foaf:Person` und `sioc:Post` eine Subklasse von `foaf:Document`. Daneben werden Relationen zwischen den definierten Konzepten eingeführt. Ein `sioc:Post` lässt sich so durch die Verwendung der Relation `sioc:has_creator` mit einem `sioc:User` in Beziehung setzen. Die SIOC-Ontologie lässt sich zudem über Module erweitern. So existiert z. B. das SIOC Type Module, das zusätzliche Klassen für Online-Gemeinschaften bereitstellt. Listing 2.5 zeigt einen einfachen, mit Hilfe von FOAF und SIOC modellierten Sachverhalt, der so oder in ähnlicher Form im Web auftreten könnte.

2.4. Wikis, semantische Wikis und Daten-Wikis

Der Begriff Wiki stammt aus Hawaii und bedeutet „schnell“. Wikis, ursprünglich auch WikiWikiWeb genannt [Leuf and Cunningham, 2001], gehören zur Kategorie der Content-Management-Systeme. Sie zeichnen sich durch die Möglichkeit aus, Inhalte schnell und einfach editieren zu können. Als Inhaltsbausteine dienen hierfür die Wiki-Seiten. Diese können durch die Nutzer, meistens mit Hilfe einer speziellen Syntax, jederzeit bearbeitet werden. Existiert eine aufgerufene Wiki-Seite nicht, wird sie erzeugt. Um dieses offene Konzept praktisch nutzbar zu machen, kommt in den meisten Wiki-Systemen zusätzlich eine Versions- und Rechteverwaltung zum Einsatz. Außerdem bieten alle gängigen Wikis die Möglichkeit der Diskussion auf Basis von Wiki-Seiten. Einzelne Wiki-Seiten können andere Wiki-Seiten

referenzieren, wodurch eine einfache Navigation innerhalb von Wikis ermöglicht wird. Zusätzlich wird in den meisten Fällen eine einfache Freitext-Suche bereitgestellt. Das wohl populärste Wiki ist die freie Enzyklopädie Wikipedia, die auf dem Wiki-System MediaWiki basiert. Obwohl Wikis häufig verwendet werden, um Informationen zu strukturieren, besitzen die Inhalte für Maschinen keine Bedeutung. Aus diesem Grund existieren zahlreiche Ansätze, Wikis mit semantischen Technologien zu erweitern.

Semantische Wikis

Semantische Wikis erweitern das Wiki-Konzept derart, dass es möglich wird, bestimmte Informationen semantisch zu annotieren, so dass diesen eine formale Bedeutung zugewiesen werden kann. So können bspw. einfache Links zwischen Wiki-Seiten zu konkreten Relationen zwischen Konzepten erweitert werden. Die Annotation erfolgt über eine spezielle Wiki-Syntax oder zusätzliche Eingabemasken. Durch die semantische Annotation ist u. a. eine verbesserte Navigation innerhalb von Wikis möglich, z. B. eine Navigation nach Facetten. Daneben kann eine Suche realisiert werden, die mehr Informationen, als nur Titel und Inhalt mit einbezieht und es ist eine höhere Konsistenz der Inhalte möglich. Soll in einem normalen Wiki eine Liste aller deutschen Städte mit mehr als 100.000 Einwohnern erzeugt werden, so muss diese Liste manuell erstellt und gewartet werden. In einem semantischen Wiki könnte diese Liste automatisch generiert werden, vorausgesetzt alle Datensätze zu Städten enthalten Aussagen über das entsprechende Land, die Einwohnerzahl und die Zugehörigkeit zum Konzept Stadt. Semantic MediaWiki [Krötzsch et al., 2006] ist ein semantisches Wiki mit den beschriebenen Eigenschaften. Es wurde als Erweiterung zum weit verbreiteten MediaWiki realisiert. Trotz der Erweiterung um semantische Technologien, steht auch bei den meisten semantischen Wikis der textuelle Inhalt im Zentrum der Betrachtung. Aus diesem Grund soll an dieser Stelle ein weiterer, auf dem Wiki-Konzept basierender Applikationstyp vorgestellt werden.

Semantische Daten-Wikis

Semantische Daten-Wikis unterscheiden sich zu anderen Wikis in einem wesentlichen Punkt. Anstelle der Verwendung von Wiki-Seiten als zentrale „Informationsbausteine“, deren Inhalte zu einem großen Teil aus Text in prosaischer Form bestehen, werden Ressourcen betrachtet, denen über Eigenschaften und Relationen in strukturierter Weise Informationen zugeordnet werden können. Daher können Daten-Wikis eher als Werkzeuge zur Verwaltung

von strukturiertem Wissen angesehen werden, die um wiki-typische Funktionen erweitert werden, wobei die einzelnen Subsysteme ebenfalls auf der Basis von Ressourcen arbeiten. Eine konkrete Realisierung eines Daten-Wikis ist OntoWiki, das in den Kapiteln 4 und 5 detaillierter vorgestellt wird, da Teile der Arbeit auf OntoWiki basieren.

3. Anforderungen

Im folgenden Kapitel werden die Anforderungen an ein System erarbeitet, das eine Datenintegration auf verschiedenen Ebenen in semantischen Daten-Wikis ermöglichen soll und zu diesem Zweck auch Funktionen zur Kommunikation zwischen Wiki-Instanzen bereitstellen muss. Diesbezüglich werden zunächst Anwendungsfälle definiert, die so gewählt werden, dass sie ein möglichst breites Spektrum der erforderlichen Funktionalität abbilden. Anschließend werden Ebenen der Datenintegration eingeführt, die für diese Arbeit relevant sind. Schließlich lassen sich die Anforderungen aus den Anwendungsfällen ableiten, wobei sowohl die funktionalen, als auch nichtfunktionale Anforderungen an ein zu entwickelndes System betrachtet werden. Die funktionalen Anforderungen werden dabei den zuvor definierten Ebenen zugeordnet.

3.1. Anwendungsfälle

Im Folgenden werden Anwendungsfälle definiert, welche die erforderliche Funktionalität eines zu entwickelnden Systems aus Nutzersicht beschreiben. Bei der Auswahl der Anwendungsfälle wurde darauf geachtet, dass diese nach Möglichkeit alle relevanten Aspekte beleuchten und gleichzeitig den praktischen Nutzen für den Endnutzer aufzeigen.

3.1.1. Wiederverwendung von Ressourcen

Für die erfolgreiche Integration von Daten unterschiedlicher Herkunft, bedarf es zunächst der Klärung einiger Fragen, die in Abbildung 3.1 skizziert sind und im Folgenden näher erläutert werden:

1. Zunächst muss sichergestellt sein, dass die gewünschte Ressource betrachtet wird. Soll bspw. eine Aussage über Berlin getroffen werden, so muss in einem ersten Schritt

geklärt werden, ob damit die Stadt oder das Bundesland gemeint ist oder gar ein Ortsteil der Gemeinde Seedorf[8] in Schleswig-Holstein.

2. Nachdem ein Konzept bestimmt wurde, muss dieses eindeutig identifiziert werden. Ein Mensch kann verschiedene Entitäten unter Umständen aus dem Kontext heraus unterscheiden, obwohl sie evtl. den gleichen Wortlaut besitzen. Computerprogramme sind hierzu nicht in der Lage und benötigen deshalb einen Mechanismus, um Ressourcen eindeutig identifizierbar zu machen. In RDF wird zu diesem Zweck vom Konzept der URIs Gebrauch gemacht.
3. Ockhams Skalpell, ein Prinzip, das implizit von Wilhelm von Ockham geprägt und deshalb nach ihm benannt wurde, besagt, dass Entitäten nur soweit notwendig dupliziert werden sollten. Für die Zuordnung von URIs zu Entitäten, sollte Ähnliches gelten. Wann immer eine URI für eine Ressource wissentlich bereits existiert, sollte diese für die eigenen Zwecke wiederverwendet werden.
4. Um in einem letzten Schritt Daten, die mit einer Ressource verknüpft sind, abrufen zu können, werden URIs benötigt, die mit RDF-Daten hinterlegt sind. In Anlehnung an das heutige Hypertext-Web, in dem über URLs Dokumente in der Hypertext Markup Language (HTML) bereitgestellt werden, erfolgt im Semantic Web eine Bereitstellung von RDF-Dokumenten über URIs. Handelt es sich dabei um HTTP-URIs, spricht man auch von Linked Data, wie im Grundlagen-Kapitel bereits erläutert wurde.

In vielen Situationen sind Informationen relevant, die einer maßgeblichen Quelle zugeordnet sind. Im Linked Data Kontext stellt der Eigentümer einer URI solche Informationen bereit [Bizer et al., 2007]. Zusätzlich werden teilweise auch Daten benötigt, die explizit nicht von einer maßgeblichen Quelle stammen, bspw. bei der Recherche über ein Produkt, für das eine Kaufabsicht besteht. Oft existieren für Ressourcen auch mehrere maßgebliche Informationsquellen. Für das Konzept Stadt Leipzig könnte eine solche Quelle bspw. mit der URI `http://www.leipzig.de#city` verknüpft sein. Zusätzlich definiert aber auch DBpedia mit der URI `http://dbpedia.org/resource/Leipzig` diese Entität.

In den meisten Fällen sind mit URIs heutzutage keine RDF-Daten direkt verknüpft. Das Beispiel der Stadt Leipzig demonstriert diesen Umstand. Über die URI `http://www.leipzig.de#city` lässt sich zum heutigen Zeitpunkt ausschließlich eine HTML-Repräsentation beziehen, die zwar u. a. Informationen über die Stadt Leipzig enthält, aber lediglich für den menschlichen Betrachter aufbereitet wurde. Für ein entsprechendes Com-

[8] `http://de.wikipedia.org/wiki/Seedorf_(Kreis_Segeberg)#Berlin`

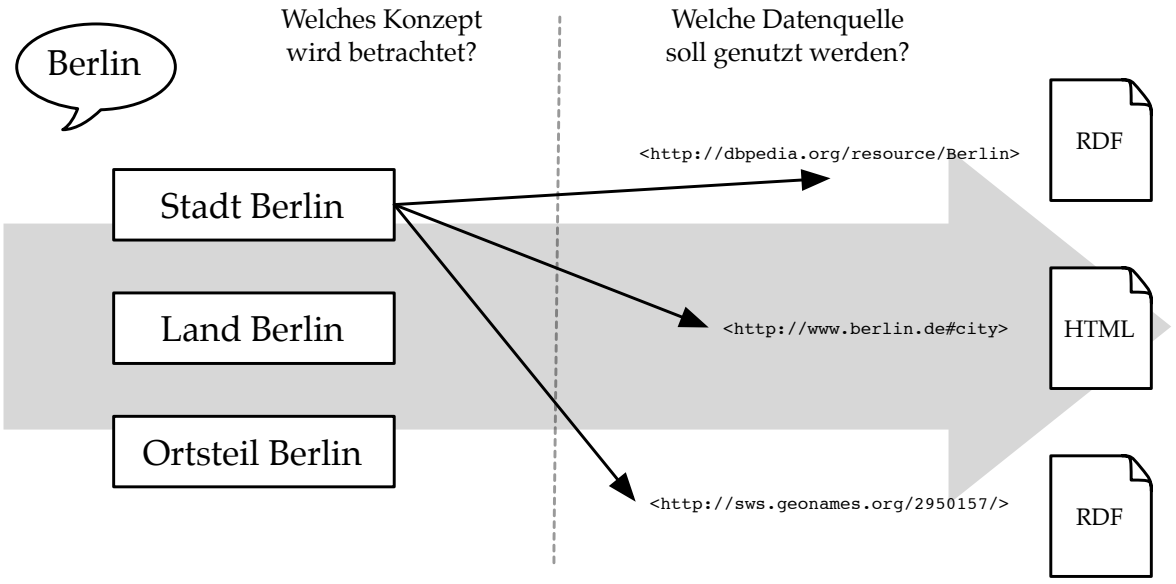

Abbildung 3.1.: Mögliche Fragen bei der Datenintegration

puterprogramm enthält ein solches Dokument, außer Hinweisen zur Darstellung auf einem Bildschirm, keine sinnvollen Informationen. In solchen Fällen wird auf URIs zurückgegriffen, die vertrauenswürdigen Quellen zugeordnet sind. Derzeit existieren bereits derartige Quellen, die über die von ihnen bereitgestellten URIs eine große Menge an strukturierten Informationen zur Verfügung stellen. Ein herausragendes Beispiel stellt DBpedia [Auer et al., 2007] dar. Beim DBpedia-Projekt werden RDF-Daten aus Wikipedia[9] extrahiert und als Linked Data verfügbar gemacht. Über die DBpedia-URI der Stadt Leipzig werden derzeit bereits eine Vielzahl an Daten bereitgestellt, u. a. Geokoordinaten, wie in Listing 3.1 dargestellt. Sind keine verlässlichen Quellen verfügbar, kann dennoch auf weniger vertrauenswürdige Daten zurückgegriffen werden. Diese können als Basis für eine manuelle Nachbearbeitung dienen.

Sollten keine RDF-Daten mit einer URI direkt verknüpft sein, kann diese dennoch von Bedeutung sein, da mit ihrer Hilfe evtl. Rückschlüsse auf Art und Ort der Daten gezogen werden können. Die Kerninformation liefert dabei der Hostname einer URL. Aus dieser Information lässt sich in vielen Situationen ableiten, wie die Daten angesprochen werden können und welche weitere Semantik in der URI steckt. Viele Internetdienste stellen heutzutage proprietäre Programmierschnittstellen zur Verfügung, über die strukturierte Informationen

[9] `http://wikipedia.org`

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
@prefix dbpprop: <http://dbpedia.org/property/> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
# ...
dbpedia:Leipzig dbpprop:latDeg "51"^^xsd:integer ;
                dbpprop:latMin "20"^^xsd:integer ;
                dbpprop:latSec "0"^^xsd:integer  ;
                dbpprop:lonDeg "12"^^xsd:integer ;
                dbpprop:lonMin "23"^^xsd:integer ;
                dbpprop:lonSec "0"^^xsd:integer  .
# ...
```

Listing 3.1: Auszug der RDF-Daten zur DBpedia-URI der Stadt Leizig

bezogen werden können. So lässt sich z. B. aus der URI `http://twitter.com/pfrischmuth` ableiten, dass

- es sich um eine URI aus dem Namensraum von Twitter handelt,
- die Twitter-API genutzt werden kann, um Daten abzurufen und
- sich die abrufbaren Daten auf den Benutzeraccount `pfrischmuth` beziehen.

Twitter[10] ist ein sog. Microblogging-Dienst, der derzeit eine sehr große Nutzergemeinde umfasst. Die Nutzer veröffentlichen über diesen Dienst kurze Nachrichten mit einer Länge von maximal 140 Zeichen.

Mit Hilfe der genannten Programmierschnittstelle, lassen sich eine Vielzahl an strukturierten Informationen abrufen und ggf. in RDF überführen. Demnach wäre es denkbar, Twitter-Nachrichten auf Instanzen der OWL-Klasse `sioc:MicroblogPost` abzubilden.

URIs und insbesondere HTTP-URIs, haben in der Regel die Eigenschaft, relativ lang zu sein und komplizierte Zeichenkombinationen zu enthalten, so dass sich solche Bezeichner für Ressourcen schlecht einprägen lassen. Dieses Problem existierte bereits im ursprünglichen Web, dem Web der Dokumente und bedingt mitunter die Popularität von Suchmaschinen. Anstelle einer klassischen Suchmaschine wie Google[11], die Links zu Dokumenten liefert, die für den menschlichen Betrachter bestimmt sind, werden Suchmaschinen benötigt, welche Links zu Dokumenten bereitstellen, die maschineninterpretierbare Daten enthalten. Eine solche Suchmaschine ist Sindice (s. Kaptiel 4). Sindice durchsucht das Internet nach RDF-

[10] `http://twitter.com/`
[11] `http://www.google.com/`

Dokumenten und HTML-Dokumenten, die Annotationen enthalten. Sie filtert sämtliche für eine Suche relevanten Aspekte heraus und indiziert diese. Das Ergebnis einer Sindice-Suche ist eine gewichtete Liste von Dokumenten mit relevanten Informationen zu einer bestimmten Entität. Unter der Voraussetzung, dass Daten über Linked Data zur Verfügung gestellt werden, können auf diese Weise relevante URIs für Ressourcen gewonnen werden, die für eine Wiederverwendung besonders gut geeignet sind.

Daneben enthält die lokale Datenbasis in vielen Fällen sehr exakte und domänenspezifische Daten, die daher ebenfalls mit einbezogen werden sollten. Relevante URIs könnten über eine SPARQL-Anfrage ermittelt werden.

Die Wiederverwendung von Ressourcen ist bei der Eingabe neuer Daten sinnvoll, aber auch dann, wenn zwei Ressourcen in Relation gesetzt werden sollen. Möchte ein Nutzer bspw. aussagen, dass er in der Nähe von Leipzig wohnhaft ist, sollte er ein Statement der Form

```
:me foaf:basedNear <http://dbpedia.org/resource/Leipzig>
```

erzeugen. Wie bereits erwähnt, stellt DBpedia u. a. Geokoordinaten bereit, was z. B. eine Darstellung des Wohnortes auf einer Karte ermöglicht. Verwendet der Nutzer stattdessen eine Aussage mit einem Literal der Form `"Leipzig"`, so wird ohne den Einsatz zusätzlicher Techniken wie Geokodierung, bestenfalls ein Mensch den Ort auf einer Karte darstellen können. In jedem Fall sind mit diesem Wert dann keine weiteren Informationen verknüpfbar.

3.1.2. Semantisches Kontaktdatenmanagement

Bei diesem Anwendungsfall speichert der Nutzer seine Kontaktinformationen in einer zentralen, persönlichen Wissensbasis, wobei u. a. FOAF eingesetzt wird, da es ein in diesem Kontext häufig eingesetztes Vokabular darstellt. Listing 3.2 zeigt einen typischen Kontakteintrag.

Ein möglichst großer Teil der Wissensbasis soll nun automatisch aktuell gehalten werden, wobei die folgenden Fälle unterschieden werden:

1. Bei der Datenquelle handelt es sich um ein FOAF-Dokument. In diesem Fall können die Daten oder Teile der Daten direkt in die lokale Datenbasis importiert werden, da die Informationen in RDF vorliegen. Solche FOAF-Dateien sind heutzutage weit verbreitet, u. a. weil Generatoren für bekannte soziale Dienste, wie bspw. Facebook[12]

[12] `http://www.facebook.com/apps/application.php?id=2626876931`

existieren. Für diesen Anwendungsfall sind solche FOAF-Daten deshalb von großer Bedeutung.

2. Die Datenquelle besteht aus einem HTML-Dokument mit eingebetteten RDFa-Statements oder Microformats[13]. Bei RDFa und Microformats handelt es sich um Verfahren, die eine Annotation von HTML-Dokumenten erlauben. Liegen mit RDFa annotierte Daten vor, müssen diese extrahiert und anschließend importiert werden. Werden Microformats, wie bspw. hCard verwendet, müssen die Daten extrahiert, auf ein RDF-Vokabular abgebildet und anschließend ebenfalls importiert werden.
3. Handelt es sich bei der Datenquelle um einen SPARQL-Endpunkt oder einen Wiki-Endpunkt, können die Daten über eine SPARQL-Anfrage direkt abgefragt werden und in das Modell importiert werden bzw. müssen nicht importiert werden, wenn eine Integration zur Laufzeit erfolgt. Ein solches Verfahren hat den Vorteil, dass stets aktuelle Daten zur Verfügung stehen.
4. Die Daten stammen aus einer der unzähligen Web 2.0 Quellen, wie z. B. Facebook, Delicious[14] oder Flickr[15], um nur einige wenige zu nennen. Hier müssen auf den jeweiligen Dienst angepasste Anfragen gestellt werden. Die Ergebnisse können anschließend ebenfalls auf ein RDF-Vokabular abgebildet und anschließend in die lokale Datenbasis importiert werden.

```
1 <http://philipp.frischmuth24.de/id/>  a foaf:Person ;
2   foaf:firstName "Philipp" ;
3   foaf:surname "Frischmuth" ;
4   foaf:mbox <mailto:philipp@frischmuth24.de> ;
5   foaf:birthday "1984-10-09" ;
6   foaf:homepage <http://www.frischmuth24.de> .
```

Listing 3.2: Ein FOAF-basierter Kontakteintrag

Der Nutzer möchte die so gewonnenen Daten nicht nur importieren, sondern auch aktuell halten. Er könnte für die Synchronisation mit den jeweiligen Datenquellen eine SPARQL-Anfrage definieren, die festlegt, welche Informationen für ihn relevant sind. Die so generierten Daten werden in der lokalen Wissensbasis abgelegt und müssen dort fortan aktuell gehalten werden. Der Synchronisationsprozess kann dann entweder durch den Nutzer

[13] http://microformats.org/
[14] http://delicious.com/
[15] http://www.flickr.com/

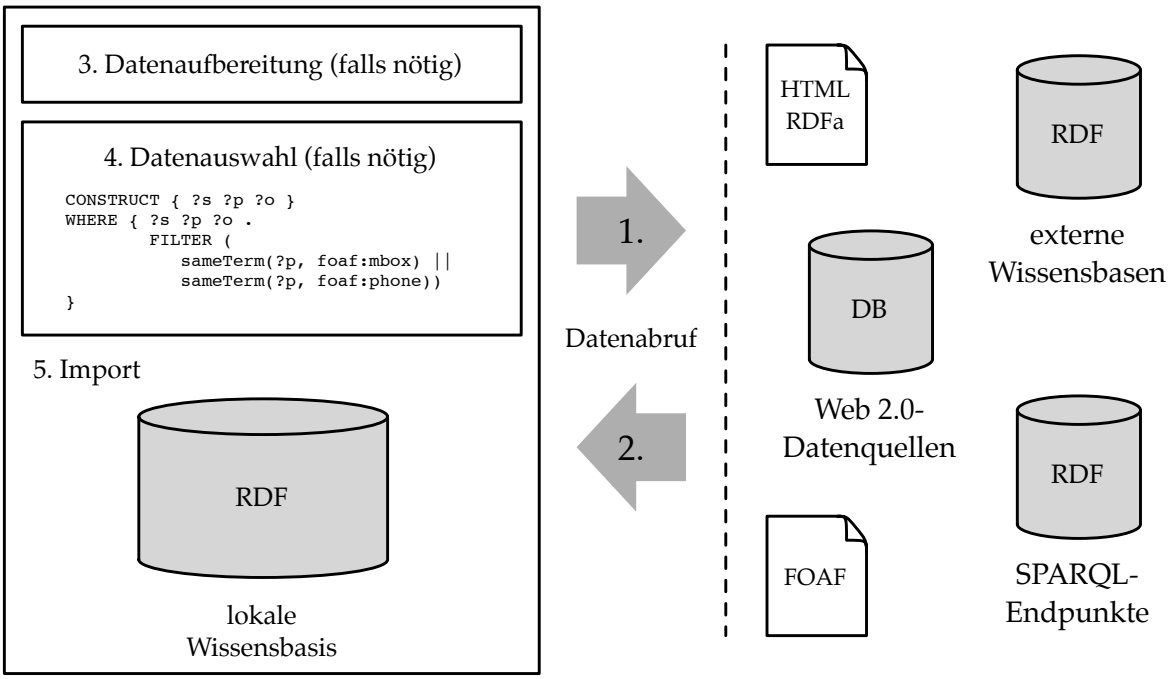

Abbildung 3.2.: Ablauf der Integration heterogener Kontaktdaten

selbst ausgelöst werden oder in gewissen Zeitintervallen automatisch stattfinden. Zusätzlich wäre es denkbar, dass sich entsprechende Werkzeuge untereinander über Änderungen informieren. Unterstützt eine Datenquelle SPARQL, so müssen lediglich die Informationen übertragen werden, die im Ergebnis der SPARQL-Anfrage auftauchen. Bei Datenquellen, die kein SPARQL unterstützen, müssen in einem ersten Schritt alle Daten abgerufen werden. Anschließend wird die Anfrage auf die gewonnenen Daten angewendet und das Ergebnis dieser Anfrage wird verwendet, um die lokalen Daten zu aktualisieren.

Diese Form der Kontaktdatenhaltung hat für den Nutzer den Vorteil, dass die Datenbasis stets aktueller ist, als sie es bei einer manuellen Pflege wäre. Sämtliche Informationen können direkt vom Eigentümer der Daten bezogen werden und müssen bei einer Aktualisierung nicht erneut eingepflegt werden. Zusätzlich liegen die Daten in einem offenen, standardisierten Format vor, wodurch mit relativ wenig Aufwand neue Benutzungsschnittstellen realisierbar sind oder bereits vorhandene Werkzeuge auf die Benutzung mit einer RDF-Datenbasis angepasst werden können. So wäre z. B. eine SyncML-Anwendung vorstellbar, welche die Kontaktdaten mit einem mobilen Endgerät synchron hält. SyncML[16] steht für

[16] http://www.openmobilealliance.org/syncml/

Synchronization Markup Language und ist ein Standard zur Datensynchronisation, der vor allem auf mobilen Endgeräten implementiert ist. Abbildung 3.2 zeigt eine schematische Darstellung der Vorgänge, die für eine Integration von Kontaktdaten aus heterogenen Datenquellen notwendig sind.

3.1.3. Erkunden von Linked Data

Im diesem Anwendungsfall möchte der Nutzer mit Hilfe eines semantischen Daten-Wikis Informationen erkunden, die über Linked Data bereitgestellt werden. Durch den Einsatz von Relationen innerhalb der Daten, werden weitere Ressourcen referenziert, die ebenfalls durchsucht werden können. Applikationen, die das Erkunden und Analysieren von RDF-Daten ermöglichen, bezeichnet man auch als Linked Data Browser. Ein solcher arbeitet ähnlich wie ein Webbrowser, visualisiert dabei aber keine HTML-Dokumente, sondern RDF-Dokumente. Ruft der Nutzer bspw. eine DBpedia-URI auf, sollte er in die Lage versetzt werden, sämtliche Daten zu betrachten und ggf. Links zu folgen, die auf weitere DBpedia-URIs verweisen oder DBpedia mit anderen Datenquellen verbinden. In diesem Anwendungsfall soll der Nutzer zusätzlich dazu befähigt werden, auch solche strukturierten Informationen zu explorieren, die nicht in RDF vorliegen. Zu diesem Zweck muss der Browser erweiterbar sein, um für solche Datenquellen ebenfalls RDF-Daten produzieren zu können. Im heutigen Web existieren eine Vielzahl von strukturierten Informationen. Diese Daten sind jedoch größtenteils nicht über ein generisches Verfahren abrufbar, sondern verborgen hinter proprietären Programmierschnittstellen und Datenmodellen. Erweiterungen des Browsers sind leichtgewichtig und auf bestimmte Dienste im Internet zugeschnitten. Sie rufen die jeweiligen Informationen ab und transformieren diese in das RDF-Datenmodel.

```
1 @prefix status: <http://twitter.com/coldplay/status/> .
2
3 <http://twitter.com/coldplay>
4   sioc:creator_of status:1973804049, status:1990253431, status:2005500325,
5                   status:2006462577, status:2021787248 ;
6   foaf:accountName "coldplay" .
```

Listing 3.3: Beispiel eines RDF-Generats für den Twitter-Account von Coldplay

Ein Beispiel für verlinkte Daten im erweiterten Sinne, lässt sich erneut mit Hilfe von Twitter skizzieren. Twitter-URLs lassen sich dafür verwenden, um einzelne Ressourcen zu identifizieren. Wird z. B. die Twitter-URL `http://twitter.com/coldplay` in einem Webbrowser

aufgerufen, wird ein HTML-Dokument erzeugt, das u. a. die letzten Einträge des Nutzers enthält. Eine Erweiterung könnte für solche Ressourcen eine Reihe von Anfragen an die Twitter-API stellen und die Ergebnisse auf ein RDF-Vokabular wie SIOC abbilden. Ein mögliches Resultat ist in Listing 3.3 dargestellt. Dem Nutzer werden die generierten RDF-Daten anschließend wie gewohnt präsentiert.

Darüber hinaus können sämtliche angezeigten Daten auch in die lokale Datenbasis importiert werden. Hierfür kann der Nutzer Teile der Daten auswählen oder sämtliche Informationen importieren.

Eine weitere Eigenschaft eines Browsers ist die Möglichkeit, den Einstiegspunkt selbständig zu definieren. Dem Nutzer soll es möglich sein, an einer beliebigen Stelle mit der Suche nach Informationen im semantischen Datennetz zu beginnen. Zu diesem Zweck gibt er die URI, welche er bspw. über Sindice (s. Kapitel 4) gefunden hat, über eine Adressleiste ein. Die Funktionsweise einer Adressleiste ist dem Nutzer bereits durch den Umgang mit Webbrowsern bekannt.

3.1.4. Reiseplanung

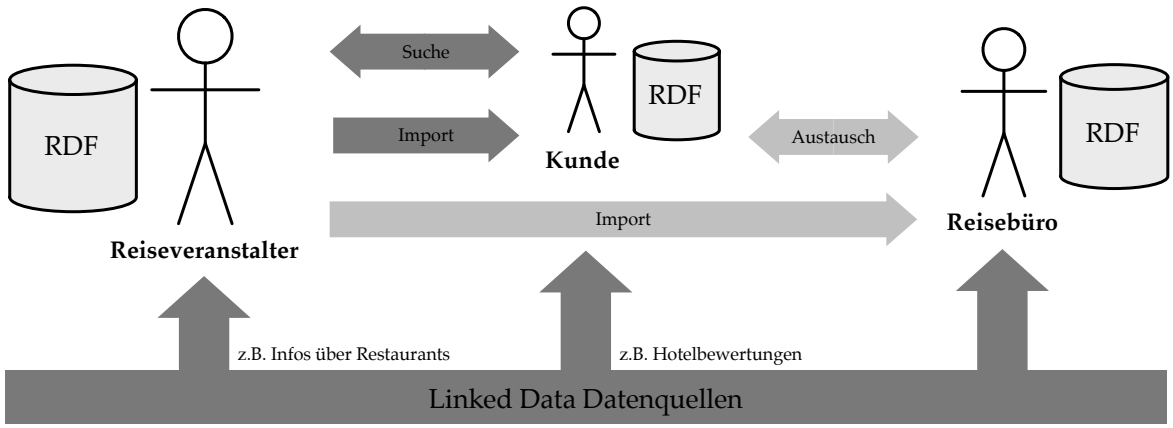

Abbildung 3.3.: Schematische Darstellung der Kommunikation zwischen den relevanten Akteuren

Im Folgenden wird ein Szenario beschrieben, in dem wichtige Teilaspekte dieser Arbeit im Kontext der Tourismus-Domäne betrachtet werden. Hierbei wird davon ausgegangen, dass ein Reiseanbieter seine Daten mit Hilfe von RDF beschrieben hat. In dieser Hinsicht ist das Szenario etwas visionär gehalten, da Reiseanbieter hiervon heute noch weit entfernt sind.

Momentan werden die Katalogdaten der Reiseveranstalter vielmehr in proprietären Datenmodellen monolithischer Softwaresysteme gehalten. Es existieren bisher wenig Gründe auf neue Technologien umzusteigen, aber wenn für Reiseanbieter der direkte Nutzen für den Endkunden erkennbar wäre und somit eine Chance auf ökonomische Vorteile gegenüber der Konkurrenz bestünde, würden Reiseanbieter ggf. einen Technologiewechsel in Betracht ziehen. Daher soll der vorliegende Anwendungsfall vor allem den Nutzen für den Endverbraucher aufzeigen. Abbildung 3.3 gibt einen Überblick über die für diesen Anwendungsfall relevanten Akteure und deren Kommunikation untereinander.

Ein Reiseveranstalter stellt Informationen zu angebotenen Produkten über einen Internetauftritt zur Verfügung. Diese Informationen enthalten eine Vielzahl strukturierter Daten und werden deshalb zusätzlich als Linked Data publiziert. Aus der Gesamtheit der beim Reiseveranstalter vorhandenen Produktdaten werden solche ausgewählt, die für das Unternehmen keinen schutzbedürftigen Charakter besitzen. Die öffentlich zugänglichen Daten enthalten u. a.

- eine Kurzbeschreibung des Angebots,
- eine detaillierte Beschreibung des Angebots,
- Produktbilder,
- Geokoordinaten,
- Hotelkategorien,
- Inklusivleistungen und
- Informationen zu optionalen Zusatzleistungen.

Des Weiteren werden Daten zur Verfügbarkeit des Angebotes in Verbindung mit den jeweils gültigen Preisen bereitgestellt. Zusätzlich sind die Angebote untereinander verknüpft, d. h. jedes Angebot enthält Links zu ähnlichen Angeboten und solchen der gleichen Preiskategorie, aber auch zu einer zufälligen Auswahl weiterer Angebote, die für den Kunden evtl. interessant sein könnten.

Der Reiseveranstalter hat sich ebenfalls dafür entschieden, Links zu DBpedia-Ressourcen über das jeweilige Reiseland in die Daten zu integrieren, um den Kunden mit Zusatzinformationen zu versorgen. Um potentielle Kunden auch über Attraktionen in der näheren Umgebung der Reiseziele zu informieren, integriert der Anbieter auch Daten weiterer Online-Dienste, die er von den jeweiligen Betreibern über Programmierschnittstellen bezieht und auf das verwendete RDF-Vokabular abbildet. Bei solchen Daten handelt es sich z. B. um

Informationen zu Restaurants in der unmittelbaren Umgebung der Hotels. Aus Angst vor negativen Einflüssen, verzichtet der Reiseanbieter darauf, Hotelbewertungen in seine Daten zu integrieren. Sämtliche Daten werden den potentiellen Kunden über eine Suchfunktion zur Verfügung gestellt, die über eine Programmierschnittstelle von beliebigen Anwendungen angesprochen werden kann. Die Suche kann einerseits genutzt werden, um Angebote für bestimmte Destinationen zu finden. Daneben lässt sie sich aber auch einsetzen, um den Reisezeitraum einzugrenzen oder Angebote für einen bestimmten Reisetyp zu suchen, unabhängig von einem bestimmten Zielort.

Ein interessierter Nutzer möchte sich nun über die diversen Angebote des Reiseveranstalters informieren. Zu diesem Zweck verwendet er die Suchfunktion seines persönlichen Reisemanagers. Bei letzterem handelt es sich um ein semantisches Daten-Wiki, das speziell auf die Bedürfnisse eines Endnutzers mit Reiseambitionen zugeschnitten wurde. Sämtliche technologischen Details, wie bspw. URIs, wurden durch benutzerfreundliche Oberflächenelemente ersetzt, so dass der Endnutzer sich auf die Reiseplanung konzentrieren kann. Die Suchfunktion der Applikation verwendet intern u. a. die Schnittstelle des Reiseveranstalters. Der Nutzer verwendet diese Suchfunktion, um Hotelangebote nach seinen Kriterien zu finden. Da die Applikation über Funktionalität zum Erkunden von Linked Data verfügt, kann er dabei zwischen den Angeboten navigieren. Sobald der Nutzer ein passendes Angebot gefunden hat, importiert er die Daten in seine lokale Datenbasis, um sie später mit weiteren Informationen vervollständigen zu können, z. B. mit Hotelbewertungen, die er von einer anderen Datenquelle bezieht. Sobald der potentielle Kunde eine Liste mit in Frage kommenden Angeboten generiert hat, kann er damit beginnen, diese zu vergleichen. Hierfür lässt er sich alle Hotels auf einer Karte darstellen. Er wählt drei Hotels aus, die gut gelegen sind und lässt sich diese in einer Liste anzeigen. Nun wählt er die für ihn maßgeblichen Kriterien, wie Preis, Hotelkategorie und Inklusivleistungen. Diese werden in zusätzlichen Spalten neben den Angeboten dargestellt. Auf diese Weise hat der Nutzer alle wichtigen Informationen auf einen Blick zur Verfügung und kann ein Angebot auswählen.

Nachdem sich der potenzielle Kunde für ein Angebot entschieden hat, kann er dieses mit Hilfe des Onlineangebots des Reiseveranstalters sofort buchen. Er entscheidet sich jedoch dafür, sich zusätzlich in einem Reisebüro beraten zu lassen. Hierfür exportiert der Nutzer die generierten Daten und sendet sie dem Reisebüro seines Vertrauens mit der Bitte, ein entsprechendes Angebot zu erstellen und evtl. weitere interessante Angebote mit einzubeziehen. Das Reisebüro kann anhand der Daten ermitteln, welche Kriterien für den Kunden relevant sind und auf dieser Basis weitere Angebote erstellen. Es hat auf die gleichen Da-

ten des Reiseveranstalters Zugriff wie der Kunde. Zusätzlich werden dem Reisebüro aber weitere Informationen zur Verfügung gestellt, bspw. über etwaige Sonderkonditionen. Der Mitarbeiter des Reisebüros kann die URIs der Reiseangebote also erneut dereferenzieren. Da das System des Reiseveranstalters den Nutzer als Mitarbeiter eines berechtigten Reisebüros authentifiziert, werden die Zusatzinformationen in die Daten integriert. Der Mitarbeiter erstellt dem Kunden auf dieser Basis ein neues Angebot und sendet es ihm zu. Dieser hat nun die Möglichkeit, das Angebot mit den aktualisierten Preisen in seinem Reisemanager erneut zu analysieren.

3.1.5. Kommunikation zwischen Wiki-Instanzen

Trotz des offenen Wiki-Gedankens spielen gerade bei Daten-Wikis auch Zugriffskontrolle und Rechteverwaltung eine bedeutende Rolle. Durch eine Rechteverwaltung wird es ermöglicht, dass zusätzlich zu den öffentlich verfügbaren Daten, auch vertrauliche Informationen für gewisse Teilnehmer verfügbar gemacht werden können. Zum Einen wird eine Zugriffskontrolle auf User Interface (UI) Ebene benötigt, die es Nutzern ermöglicht, sich an einem System anzumelden. Zum Anderen bedarf es einer Zugriffskontrolle auf Ebene der APIs, damit Anwendungen untereinander schutzbedürftige Daten ohne eine direkte Beteiligung des Nutzers austauschen können. Im einfachsten Fall werden zu diesem Zweck bei jeder Anfrage Nutzername und Kennwort des Anwenders integriert. Dieses Verfahren setzt eine verschlüsselte Verbindung voraus, da ansonsten die sensiblen Nutzerdaten stets im Klartext übertragen werden würden. Ein Verfahren, das auf eine Übertragung des Nutzerpassworts verzichtet, wäre generell vorzuziehen, da sich ein potentielles Sicherheitsrisiko ergibt, wenn ein Nutzer sein Passwort der Applikation preisgibt und diese es im Klartext bereithalten muss. Ein typisches Szenario für eine Authentifizierung auf API-Ebene könnte sich wie folgt gestalten:

1. Ein Nutzer möchte ein geschütztes FOAF-Profil abrufen und teilt seiner Applikation A zu diesem Zweck die URI des FOAF-Profils mit. Der Nutzer selbst besitzt Zugriff auf die Daten, da er mit dem Eigentümer der Informationen direkt oder indirekt in einer Relation steht, die besagt, dass der Eigentümer dem Nutzer vertraut. Ein solches *Web of Trust* ließe sich z. B. mit Hilfe der `foaf:knows`-Relation konstruieren.
2. Die Applikation versucht das Profil abzurufen und scheitert, da eine Authentifizierung notwendig ist. Sie unternimmt einen erneuten Versuch, die Daten abzurufen, integriert aber zusätzlich die Information, dass die Anfrage im Namen des Nutzers geschieht.

3. Der Nutzer hat Zugang zur Applikation mit den geschützten Daten und Zugriff auf die gewünschte Ressource. Bisher hat er dieser Applikation jedoch nicht mitgeteilt, dass Applikation A in seinem Namen handeln darf. Aus diesem Grund wird er zu der Applikation mit der geschützten Ressource weitergeleitet, um sich dort am System anzumelden und eine entsprechende Berechtigung einzurichten.
4. Applikation A kann nun die geschützten Informationen direkt abrufen und muss den Nutzer zukünftig nicht mehr in diesen Vorgang einbeziehen.

Es sei angemerkt, dass die einzelnen Phasen notwendigerweise mit Hilfe kryptographischer Verfahren abgesichert werden müssen, damit sichergestellt werden kann, dass die ausgetauschten Nachrichten nicht durch einen Angreifer verändert werden können.

Es ist offensichtlich, dass in einigen Fällen trotzdem eine Zugriffskontrolle auf UI-Ebene erforderlich ist. Diesbezüglich kann im einfachsten Fall wieder eine Kombination aus Nutzername und Passwort zum Einsatz kommen. Hierfür ist es jedoch erforderlich, dass der Nutzer für jedes System, zu dem er Zugriff benötigt, ein separates Konto erstellt, jeweils mit Nutzername und Kennwort. Dieses Problem ist nicht neu, es resultiert vielmehr aus dem Web 2.0 mit seinen diversen Diensten und sozialen Netzwerken. Derzeit sind einige Techniken etabliert oder in der Entwicklung, um diesem Problem zu begegnen, z. B. OpenID (s. Kapitel 4), ein Protokoll, dass es ermöglicht, einen einzigen Benutzernamen, der die Form einer URI besitzt, in diversen Systemen einzusetzen. Eine Wiki-Applikation könnte es dem Nutzer nun ermöglichen, sich

- mit einer bestehenden OpenID zu registrieren und/oder
- eine neue OpenID zu erstellen.

Das System könnte sowohl als Produzent (Provider), als auch als Konsument (Consumer) fungieren, um lokalen Nutzern eine OpenID bereitzustellen und gleichzeitig die OpenIDs externer Nutzer zu akzeptieren. Der Nutzer besitzt im Gegenzug lediglich einen Benutzernamen und ein Passwort, das ausschließlich der Provider seiner OpenID kennt und nur von diesem überprüft wird.

Wie bereits erwähnt, ist es auf diese Weise möglich, schutzbedürftige Informationen kontrolliert zwischen Wiki-Instanzen zu übertragen. Außerdem können auf dieser Ebene auch Daten in die Quelle zurückgeschrieben werden, vorausgesetzt, die nötigen Rechte existieren und die Wiki-Instanzen kennen die Schnittstelle, um Daten in der jeweils anderen Applikation zu schreiben. Dieser Umstand ist gegeben, wenn die Wiki-Instanzen auf der gleichen Applikation basieren. Momentan fehlt ein Standard, um RDF-Daten zu aktualisieren,

aber mit SPARQL/Update [Seaborne and Manjunath, 2007] existiert ein Entwurf, der evtl. in dieser oder ähnlicher Form in den SPARQL-Standard integriert werden könnte. SPARQL/Update erweitert die SPARQL-Anfragesprache um Operationen zur Aktualisierung von RDF-Graphen. Sollte es zukünftig einen standardisierten Weg geben, um RDF-Daten zu schreiben, so kann dieser Anwendungsfall verallgemeinert und auf Endpunkte ausgedehnt werden, die ein Schreiben von RDF-Daten erlauben.

3.1.6. Multimediale Daten

Ein weiterer Anwendungsfall betrifft die in multimedialen Daten enthaltenen Metadaten. Die meisten digitalen Dokumente, wie z. B. Bilder, Videos, Musikdateien oder PDF-Dokumente verfügen gegenwärtig zusätzlich über eine Vielzahl an Informationen über das eigentliche Datum. Bei Bildern kommt dabei u. a. das Exchangeable Image File Format (Exif) zum Einsatz, ein von der Japan Electronic and Information Technology Industries Association (JEITA) eingeführter Standard, den die meisten Digitalkameras umsetzen. Betrachtet ein Nutzer nun eine Ressource, die ein Bild repräsentiert, so ist er zum Einen an der visuellen Darstellung des Bildes interessiert. Zum Anderen möchte er aber in vielen Fällen auch wissen, wann ein Bild entstanden ist, mit welcher Kamera es fotografiert oder gar an welchem Ort es aufgenommen wurde. Diese Informationen sind in den meisten Fällen vorhanden und müssen lediglich aus der Bilddatei extrahiert werden.

Ein Daten-Wiki könnte nun bei der Anzeige eines Bildes zusätzlich Exif-Daten aus diesem extrahieren und auf ein ensprechendes RDF-Vokabular abbilden. Es existiert bereits ein Ansatz, der für sämtliche Exif-Eigenschaften entsprechende RDF-Terme definiert[17]. Die generierten Daten könnten dann in Kombination mit dem Bild dargestellt und bei Bedarf in die lokale Datenbasis aufgenommen werden. Ähnliche Standards existieren auch für Videos oder Musikdateien. Ein Anwender könnte bspw. eine MP3-Datei verlinken und müsste Interpret und Titel nicht manuell hinzufügen, sondern lediglich die ohnehin vorhandenen Metadaten auslesen lassen.

3.2. Ebenen der Datenintegration

Die für diese Arbeit relevanten Anforderungen an die Datenintegration lassen sich, wie in Abbildung 3.4 dargestellt, verschiedenen Ebenen zuordnen. Diese Ebenen sollen im folgen-

[17] `http://www.w3.org/2003/12/exif/`

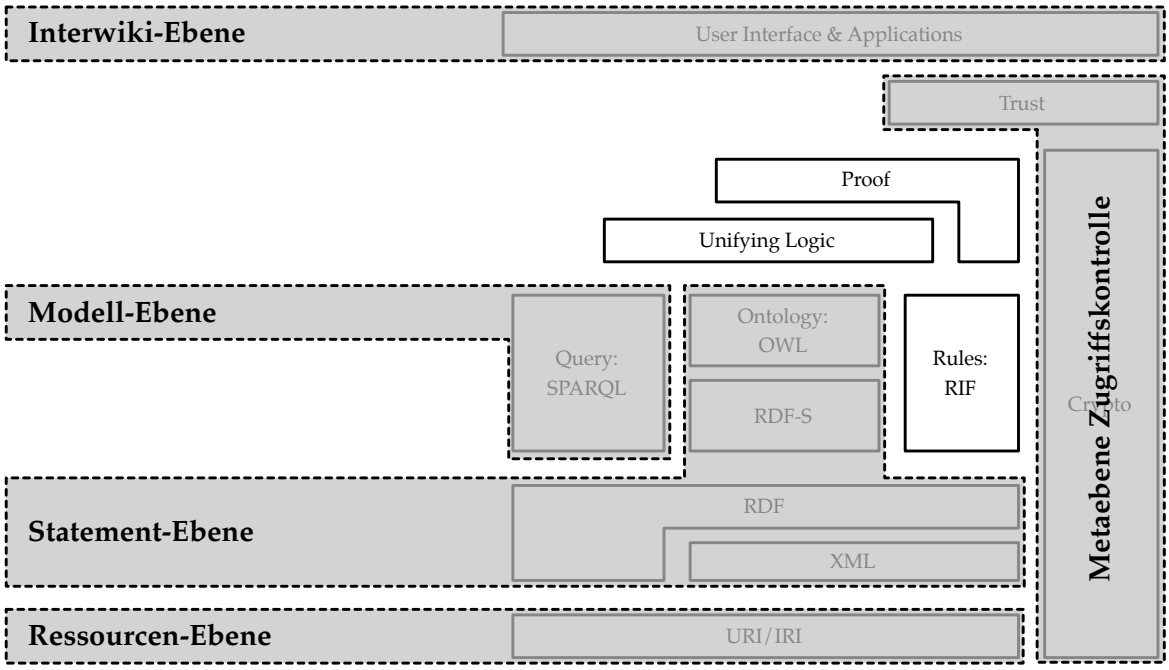

Abbildung 3.4.: Ebenen der Datenintegration mit Bezug auf das Semantic Web Schichtenmodell

den Abschnitt kurz erläutert werden, um im weiteren Verlauf der Arbeit auf sie verweisen zu können.

Ressourcen-Ebene

Auf der untersten Integrationsstufe werden die Voraussetzungen für eine Integration auf höheren Ebenen geschaffen. Die Ressourcen, über die später Aussagen getroffen werden sollen, werden identifiziert. Auf dieser Ebene findet darüber hinaus eine Zuordnung der Ressourcen zu URIs statt, die nach Möglichkeit so gewählt werden, dass Daten über diese URIs abgerufen oder generiert werden können. Um die Wiederverwendung von Ressourcen zu ermöglichen, lassen sich existierende Ressourcen außerdem über Suchfunktionen identifizieren.

Statement-Ebene

Ein einzelnes Datum wird in RDF als Statement bezeichnet und besteht aus Subjekt, Prädikat und Objekt. Auf dieser Ebene findet eine Integration genau dieser Tripel statt. Dies beinhaltet das Abrufen von Daten aus externen Datenquellen und das Importieren dieser in die lokale Datenbasis. Teilweise müssen solche Daten zuvor in das RDF-Datenmodell konvertiert werden, um Statements zu erzeugen. Zusätzlich wird auf dieser Ebene die Auswahl relevanter Statements nach Kriterien des Nutzers, sowie die Synchronisation von Daten mit der jeweiligen Datenquelle betrachtet.

Modell-Ebene

Auf dieser Ebene werden statt einzelner Statements RDF-Graphen (Modelle) betrachtet. Die Integration bezieht sich dabei nicht auf den Import einzelner Statements, sondern auf das Einbinden ganzer Modelle zur Laufzeit. Diese Integration vollzieht sich z. B. über externe SPARQL-Endpunkte, die beim Ausführen der Applikation direkt angesprochen werden. Eine explizite Synchronisation mit der Datenquelle ist auf dieser Ebene nicht notwendig, da durch die Abfrage zur Laufzeit alle Daten in der Regel aktuell sind. Zusätzlich wird auf dieser Ebene die Integration von Modellen unterschiedlicher Herkunft betrachtet.

Interwiki-Ebene

Diese Integrationsstufe bezieht Aspekte eines schreibenden Zugriffs auf entfernte Endpunkte in die Betrachtungen mit ein. Da für die Aktualisierung von RDF-Graphen bisher keine standardisierten Lösungen existieren, muss diese auf Interwiki-Ebene gesondert behandelt werden.

Metaebene Zugriffskontrolle

Auf der Metaebene Zugriffskontrolle werden Aspekte der Zugangs- und Zugriffskontrolle betrachtet. Solche Betrachtungen können auf allen Ebenen der Datenintegration relevant sein und werden deshalb übergreifend behandelt.

Mit Hilfe dieser Ebenen können im folgenden Abschnitt die Anforderungen entsprechend aufgeteilt werden. So entsteht ein modularer Anforderungskatalog, der sich in Teilpakete

gliedern lässt. Diese lassen sich dann relativ unabhängig voneinander betrachten, wodurch das Gesamtsystem an Komplexität verliert.

3.3. Funktionale Anforderungen

Aus den o. g. Anwendungsfällen lassen sich im Folgenden die funktionalen Anforderungen an das zu entwickelnde System ableiten. Diese Anforderungen werden erläutert und den Ebenen der Datenintegration zugeordnet.

3.3.1. Suche nach relevanten Ressourcen

Die vorgestellten Anwendungsfälle implizieren, dass das zu entwickelnde System den Nutzer bei der Suche nach Ressourcen unterstützen sollte. Dabei müssen zwei Stufen der Unterstützung unterschieden werden:

1. Disambiguierung von Ressourcen,
2. Zuordnung von Ressourcen zu URIs.

Die Disambiguierung betreffend, sollte das System den Nutzer bei der Auswahl des gewünschten Konzepts unterstützen. Wie bereits im Anwendungsfall am Beispiel **Berlin** geschildert, beinhaltet dies die Unterscheidung von Ressourcen mit gleichem Wortlaut und unterschiedlicher Bedeutung. Der Nutzer sollte anhand der URI oder einer textuellen Repräsentation der Ressource erkennen können, um welche Entität es sich handelt. Im zweiten Schritt soll der Benutzer dazu befähigt werden, der Ressource eine URI zuzuordnen. Zu diesem Zweck muss dem Nutzer eine möglichst exakte Liste von verfügbaren, relevanten URIs präsentiert werden, zu denen nach Möglichkeit bereits Daten hinterlegt sind. Es ist wichtig, dass dabei sowohl lokale, als auch externe Informationen mit einbezogen werden, um die Liste zu generieren. Für weit verbreitete Entitäten ist es daher sinnvoll, eine externe Suchmaschine zu bemühen, da hier mit hoher Wahrscheinlichkeit relevante URIs gefunden werden, mit denen bereits Daten verknüpft sind. Für domänenspezifische Konzepte sind möglicherweise nur in der lokalen Datenbasis bereits URIs vorhanden, die wiederverwendet werden können. Daher ist eine Kombination sowohl externer, als auch interner Informationen von großer Bedeutung. Des Weiteren ist es wichtig, dass der Nutzer mit der Wahl der Stichworte die Suche signifikant beeinflussen kann. Wie Abbildung 3.5 zeigt, liefert bspw. eine Sindice-Suche nach dem Begriff `"leipzig"` überwiegend DBpedia-URIs.

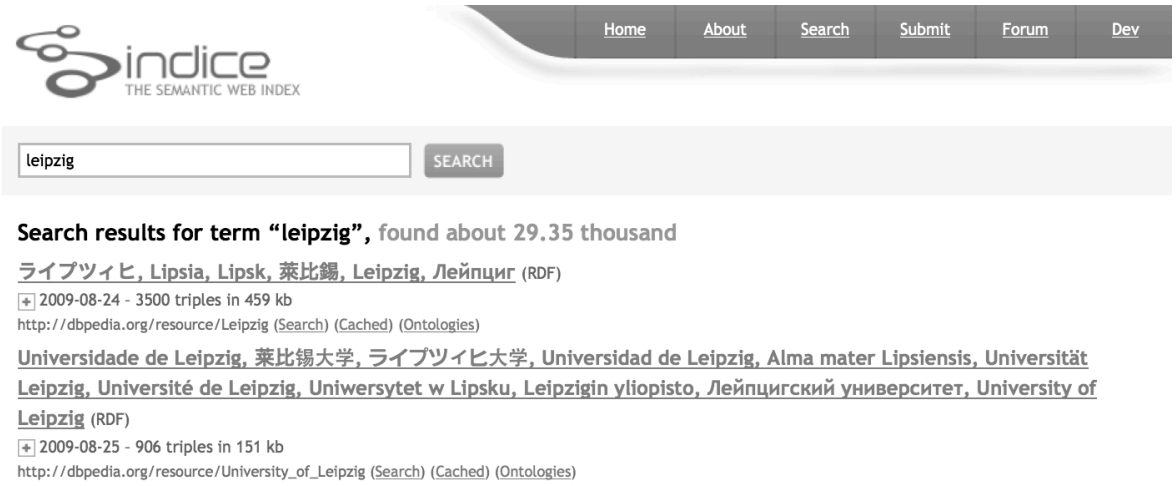

Abbildung 3.5.: Sindice-Suche nach "`leipzig`"

Weiß ein Nutzer aber bereits, dass er vor allem an Geokoordinaten interessiert ist, so bevorzugt er evtl. eine URI aus dem Geonames[18] Namensraum. Zwar enthalten auch die DBpedia-Daten mit hoher Wahrscheinlichkeit Geokoordinaten, dafür bestehen sie neben diesen aber aus einer Vielzahl weiterer Informationen, an denen der Nutzer evtl. nicht interessiert ist. In diesem Fall kann er seine Suchanfrage verfeinern und nach "`leipzig geonames`" suchen. Abbildung 3.6 zeigt das Ergebnis für diese Anfrage. Die Liste enthält nun ausschließlich Geonames-URIs.

Die Suchfunktion sollte erweiterbar sein, damit zukünftig weitere Suchmaschinen und Informationsquellen in das System integriert werden können. Derzeit liefert die Suchmaschine Sindice sehr gute Ergebnisse. Für die Suche nach Prädikaten ist die Suchmaschine aber nur eingeschränkt nutzbar. Für solche Suchanfragen ist eine lokale Suche besser geeignet. Es ist aber durchaus denkbar, dass zukünftig eine darauf spezialisierte Suchmaschine entwickelt wird. Dann sollte es möglich sein, mit möglichst geringem Aufwand auch diese in das System zu integrieren.

Eine weitere Anforderung an diese Funktionalität, ist die für den Nutzer transparente Integration. Für den Endnutzer sollte diese Funktion nach Möglichkeit lediglich aus einem einfachen Eingabefeld und einer beim Tippen generierten Auswahlliste bestehen. Hierfür müssen die Ergebnisse aller Informationsquellen in einem Ergebnis zusammengeführt werden. Dies wiederum erfordert eine nachträgliche Sortierung der teilweise vorsortierten Er-

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

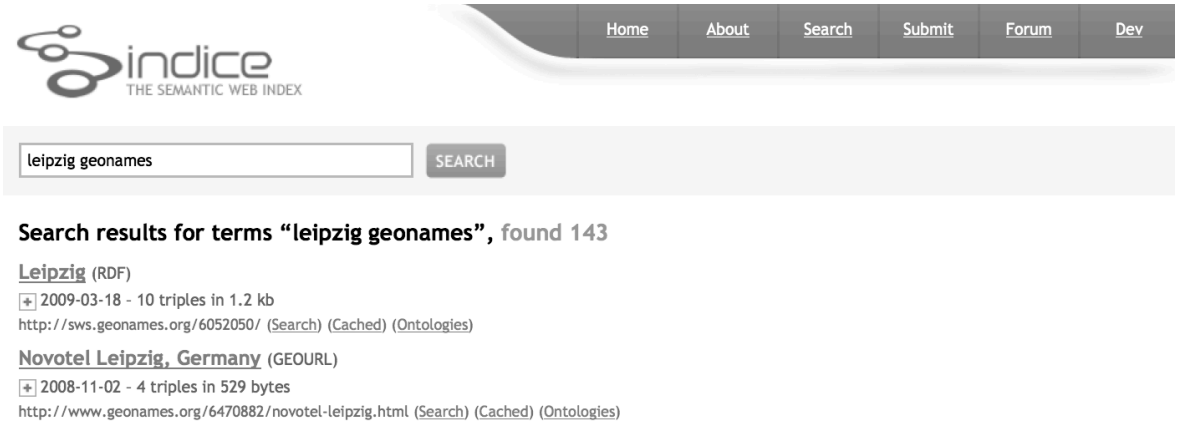

Abbildung 3.6.: Sindice-Suche nach "`leipzig geonames`"

gebnismengen. Eine weitere Möglichkeit besteht darin, die Ergebnisse der einzelnen Quellen voneinander getrennt zu halten und dem Nutzer zusätzlich die Herkunft der Informationen mitzuteilen, z. B. mit Hilfe von Abschnitten innerhalb der erzeugten Liste. Es muss dem Endnutzer aber in jedem Fall deutlich gemacht werden, woher die jeweiligen Ergebnisse stammen.

Das zu entwickelnde System sollte den Nutzer auch dann unterstützen, wenn keine Suche erforderlich ist oder keine Suchergebnisse gefunden werden. Dies ist der Fall, wenn

- der Nutzer eine URI manuell eingibt,
- er ein neues Konzept im lokalen Namensraum einführen möchte oder
- zu den gegebenen Stichworten keine Ergebnisse vorliegen und eine neue URI im lokalen Namensraum erstellt werden muss.

Bei der Generierung von URIs sollte die URI des lokalen Namensraumes mit den gegebenen Schlüsselwörtern zu einer neuen, im lokalen System eindeutigen URI zusammengefügt werden. Für die Schlüsselwörter `stadt leipzig` und den lokalen Namensraum `http://example.org/` könnte bspw. die URI `http://example.org/StadtLeipzig` generiert werden. Solch eine URI ist einer URI der Form `http://example.org/Resource815` o. ä. vorzuziehen, da sich Menschen solche URIs zum Einen besser merken können und zum Anderen die genutzten Stichworte in der URI enthalten sind. Somit wird diese URI bei einer erneuten Suche gefunden, auch wenn keine weiteren Informationen, wie z. B. ein `rdfs:label` zur Verfügung stehen. Generell ist die Wiederverwendung von URIs jedoch zu

favorisieren und deshalb sollte der Nutzer durch das System dazu animiert werden, bereits existierende URIs zu verwenden. Gibt ein Nutzer eine vollständige URI manuell ein, muss eine Suche unterbunden werden, damit, wie sich aus den Anwendungsfällen ebenfalls ableiten lässt, beliebige URIs genutzt werden können, auch solche, die evtl. nicht über eine Suche adressierbar sind.

Ebene	Ressourcen-Ebene
Eingabe	Schlagwörter oder URI
Ziel	Auswahlliste mit relevanten URIs
Subanforderungen	Erweiterbarkeit der Suchfunktion, Integration der Teilergebnisse in einem Gesamtergebnis, Generierung neuer URIs, Verwendung menschenlesbarer Bezeichner

Tabelle 3.1.: Zusammenfassung der Suchfunktion

Eine Integration dieser Funktion ist überall dort notwendig, wo

- Ressourcen zueinander in Relation gesetzt werden sollen,
- die Einführung neuer Entitäten beabsichtigt wird,
- beliebige Ressourcen dargestellt werden sollen oder
- URIs für Eigenschaften oder Relationen gesucht werden.

Tabelle 3.1 fasst die Anforderungen zusammen und ordnet sie den weiter oben definierten Ebenen der Datenintegration zu.

3.3.2. Import von Statements über eine URI

Die Anwendungsfälle zum semantischen Kontaktdatenmanagement und zur Reiseplanung führen u. a. zu der Anforderung, dass es mit dem zu entwickelnden System möglich sein muss, Daten abzurufen, die mit einer URI verknüpft sind und diese Daten in die lokale Datenbasis zu importieren. Dabei muss die Art der abzurufenden Daten unterschieden werden. Es kann sich bei den Daten um

- reine RDF-Daten,
- eingebettete RDF-Daten,

- eingebettete Daten, die leicht in RDF konvertiert werden können oder
- sonstige strukturierte Daten

handeln. Je nach Art, müssen vor dem Import der Daten weitere Schritte unternommen werden, um RDF-Daten zu generieren oder RDF-Daten zu extrahieren. Abbildung 3.7 stellt die verschiedenen Arten der abzurufenden Daten schematisch gegenüber. Es sei anzumerken, dass mit dem Begriff URI in diesem Abschnitt ausschließlich HTTP-URIs gemeint sind, da diese sich sowohl für den Endnutzer, als auch für eine Applikation direkt dereferenzieren lassen.

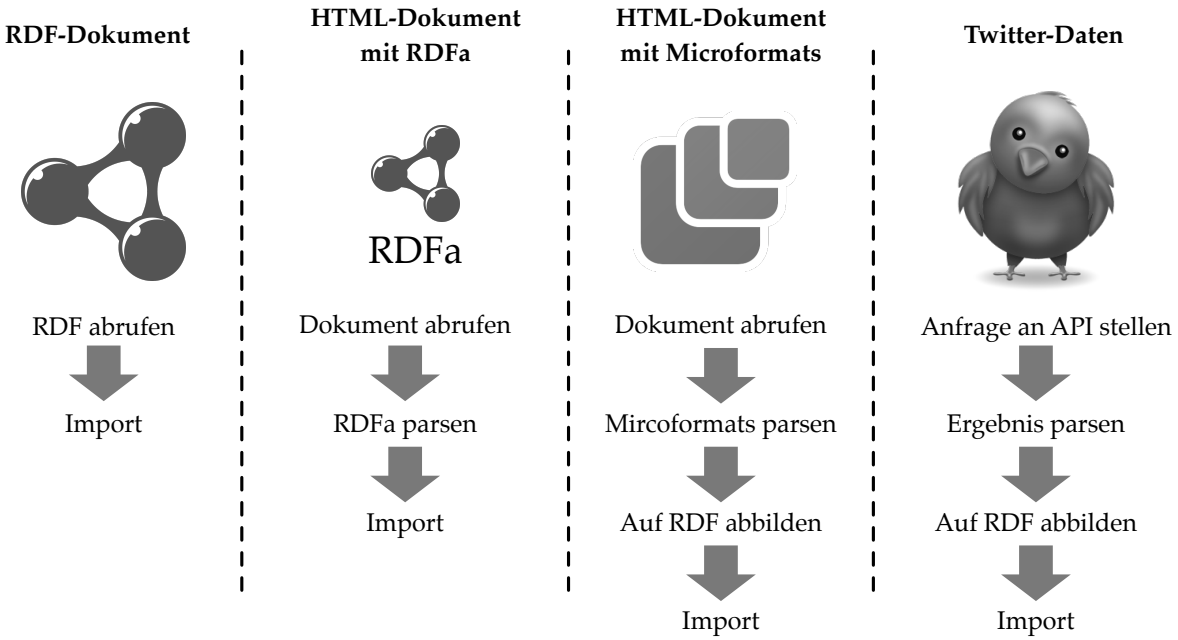

Abbildung 3.7.: Schematischer Vergleich der möglichen Abläufe beim Import von externen Daten

Liegen reine RDF-Daten vor, müssen diese lediglich abgerufen und aus dem Dokument extrahiert werden. Dieser Schritt beschränkt sich auf das Einlesen (Parsen) des Dokuments. Das Format, in dem die Daten serialisiert sind, muss dabei vom System unterstützt werden. In der Regel, wird es sich um RDF/XML-Dokumente handeln, da dieses Format vom W3C als Standard vorgesehen ist [Beckett, 2004]. Es finden aber auch andere Formate wie N3, Turtle oder die von Talis[19] vorgeschlagene JSON-Serialisierung Anwendung.

[19] `http://n2.talis.com/wiki/RDF_JSON_Specification`

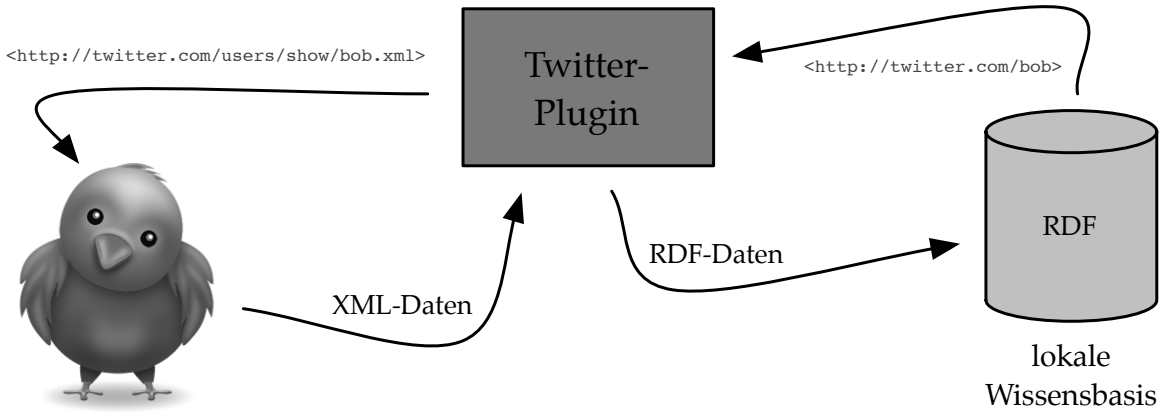

Abbildung 3.8.: Schematischer Ablauf der Generierung von RDF aus Twitter-Daten

Bei eingebetteten RDF-Daten kann es sich z. B. um RDFa handeln. RDFa basiert bereits auf RDF, die Daten sind jedoch in ein HTML-Dokument eingebettet. Ein solches HTML-Dokument muss daher mit einem speziellen RDFa-Parser eingelesen werden, um die Tripel aus dem Dokument zu extrahieren. Handelt es sich bei den eingebetteten Daten nicht um RDF-Daten, sondern bspw. um Microformats, muss das Dokument geparst und die Daten auf ein RDF-Vokabular abgebildet werden.

In derartigen Fällen, in denen keine RDF-Daten direkt verfügbar sind und die Daten nicht direkt in ein Dokument integriert sind, müssen letztere in einem ersten Schritt generiert werden. Diesbezüglich ist es notwendig, dass ein zu entwickelndes System anhand der URI erkennt, woher die Daten zu beziehen sind. Dies sollte mit Hilfe des Hostnamens der HTTP-URI geschehen. Handelt es sich z. B. um eine `twitter.com` URI, sollte das System erkennen, dass in diesem Fall die Programmierschnittstellen des Twitter-Dienstes genutzt werden müssen, vorausgesetzt dem System ist Twitter als Datenquelle bekannt. Es kann potentiell eine sehr große Anzahl unterschiedlicher Datenquellen geben, die nicht über ein einheitliches Verfahren angesprochen werden können. Daher muss das System in einer Art und Weise erweiterbar sein, dass es möglich ist, mit minimalem Aufwand weitere Datenquellen in das System einzubinden. Diese Erweiterungen sollten anhand der URI entscheiden, ob sie für diese RDF-Daten produzieren können und in einem zweiten Schritt aus den abgerufenen Daten RDF erzeugen. Die generierten RDF-Daten können anschließend, wie in den anderen Fällen, in die lokale Datenbasis importiert werden. Abbildung 3.8 schematisiert den Ablauf einer solchen Generierung von RDF-Daten am Beispiel Twitter.

```
CONSTRUCT { ?subject ?predicate ?object }
WHERE {
    ?subject ?predicate ?object
    FILTER (
      sameTerm(?predicate, <http://xmlns.com/foaf/0.1/mbox>) ||
      sameTerm(?predicate, <http://xmlns.com/foaf/0.1/phone>)
    )
}
```

Listing 3.4: SPARQL-Anfrage, um eine Ergebnismenge einzuschränken

Aus dem Anwendungsfall zu multimedialen Daten folgt zusätzlich, dass auch dann RDF-Daten generierbar sein müssen, wenn es sich bei einer Ressource um ein Dokument handelt, das bereits strukturierte Metadaten enthält. Diese müssen aus der Datei extrahiert und auf ein passendes RDF-Vokabular abgebildet werden. Da es sich bei solchen Dokumenten häufig um Binärdaten handelt, ist bei der Extraktion auf Werkzeuge zurückzugreifen, die das jeweilige Format unterstützen.

Ebene	Statement-Ebene
Eingabe	URI
Ziel	Import von RDF-Daten in die lokale Datenbasis
Subanforderungen	Erweiterbarkeit bezüglich der Datenquellen, Generierung von RDF-Daten, Auswahl relevanter Statements durch den Nutzer, Zwischenspeicherung von Anfragen

Tabelle 3.2.: Zusammenfassung der Importfunktion

Zusätzlich zu der Möglichkeit, alle verfügbaren Daten zu importieren, sollte das System in der Lage sein, eine vom Nutzer definierte Auswahl zu importieren. Dies könnte aus Nutzersicht über eine Liste realisiert werden, die alle gefundenen Statements enthält. Der Nutzer wählt die Statements aus, die er importieren möchte. Aus Applikationssicht könnte die Auswahl auf eine SPARQL-Anfrage abgebildet werden, die für eine spätere Synchronisation wiederverwendet werden kann. Der Anwendungsfall zum semantischen Kontaktdatenmanagement führt zu dieser Anforderung, denn in der Regel sollen nur gewisse Informationen, wie z. B. Telefonnummer und Anschrift importiert werden. Eine SPARQL-Anfrage, die nur bestimmte Statements berücksichtigt, ist in Listing 3.4 dargestellt.

Auch die Zwischenspeicherung der abgerufenen Daten durch das zu entwickelnde System ist notwendig, um möglichst effizient mit Anfragen an externe Ressourcen umzugehen. Einige externe Datenquellen begrenzen die Anzahl der möglichen Anfragen innerhalb eines gewissen Zeitraumes. Bei Twitter liegt diese Begrenzung derzeit bei 150 Anfragen pro Stunde. Da sich die Daten in den meisten Fällen aber nicht kontinuierlich ändern, ist eine Abfrage der Datenquellen mit gewissen Zeittoleranzen akzeptabel. Die Gültigkeit zwischengespeicherter Ergebnisse sollte für jede Datenquelle konfigurierbar sein, damit Datenquellen mit einer begrenzten Anfragekapazität und solche mit einer hohen Änderungsfrequenz, aufeinander abgestimmt werden können.

Tabelle 3.2 fasst die Anforderungen zusammen und ordnet sie den weiter oben definierten Ebenen der Datenintegration zu.

3.3.3. Synchronisation

Auch die Anforderung für eine Synchronisation lässt sich aus dem Anwendungsfall zum Kontaktdatenmanagement ableiten. Demnach sollte das zu entwickelnde System eine Möglichkeit bieten, Daten mit der ursprünglichen Datenquelle zu synchronisieren. Ein Synchronisationsvorgang besteht dabei aus

- dem erneuten Abrufen der Daten von der Datenquelle,
- der Auswahl der gewünschten Informationen aus den abgerufenen Daten und
- dem Ersetzen der ursprünglich importierten Daten durch die aktualisierten Daten.

Die Auswahl der relevanten Statements erfolgt intern über eine SPARQL-Anfrage (s. Listing 3.4 auf Seite 42), die auf die abgerufenen Daten angewendet wird. Das Ergebnis dieser Anfrage enthält die gewünschten und aktuellen Daten, die in die lokale Datenbasis importiert werden können und die alten Daten ersetzen. Das zu entwickelnde System muss demnach stets eine Kopie der zuletzt importierten Daten bereit halten, um diese später wieder aus dem Modell löschen zu können. Ein Entfernen aller Statements, die als Ergebnis der konfigurierten SPARQL-Anfrage auftreten, wäre unzureichend, da es möglich ist, dass solche Statements auf einem anderen Weg in die lokale Datenbasis gelangt sind, z. B. durch den Import über eine weitere Datenquelle. Ist die Information über den letzten Import nicht vorhanden, kann keine Synchronisation stattfinden, sondern lediglich ein erneuter Import.

RDF-Tripel sind innerhalb eines Graphen eindeutig. Es kann vorkommen, dass Datenquellen Statements liefern, die bereits im lokalen Datenbestand vorliegen und deshalb nicht erneut

importiert werden. Ein späteres Synchronisieren der Datenquelle könnte zur Folge haben, dass solche Statements entfernt würden, obwohl sie beim ursprünglichen Import nicht berücksichtigt worden sind. In der Regel führt dieses Verhalten jedoch zum gewünschten Effekt, wie das folgende Beispiel illustriert.

Es wird angenommen, dass die lokale Datenbasis ein Statement der Form

```
<http://example.org/me> foaf:mbox <mailto:me@example.org>
```

enthält. Die zu importierenden Daten beinhalten ebenso dieses Tripel, das dementsprechend ignoriert wird. Wird die Datenquelle nun zu einem späteren Zeitpunkt aktualisiert und enthält dann ein modifiziertes Tripel

```
<http://example.org/me> foaf:mbox <mailto:me-new@example.org>,
```

wird das lokale Statement ersetzt und fortan stehen die aktuellen Daten zur Verfügung. Es kann allerdings auch der Fall eintreten, dass die Emailadresse des Kontaktes im lokalen Modell gelöscht wird, wenn dieser Kontakt entscheidet, seine Emailadresse nicht mehr öffentlich freizugeben. Diesem Problem könnte das zu entwickelnde System durch einen entsprechenden Warnhinweis oder einen Bestätigungsdialog begegnen, der den Nutzer auf ein bloßes Löschen von Statements hinweist.

Ebene	Statement-Ebene, Modell-Ebene
Eingabe	URI, Graph
Ziel	Aktualisierte lokale Daten
Subanforderungen	Konfigurierbarkeit über SPARQL-Anfrage

Tabelle 3.3.: Zusammenfassung der Synchronisationsfunktion

Zusätzlich zur Synchronisation von importierten Statements, sollte das System eine Synchronisation auf Modell-Ebene ermöglichen. Hierfür werden die relevanten Daten ebenfalls mit Hilfe einer SPARQL-Anfrage identifiziert und zur Laufzeit abgefragt. Um ein effizientes Laufzeitverhalten zu ermöglichen, werden die Daten für eine konfigurierbare Zeitdauer zwischengespeichert. Ist dieser Zeitraum überschritten, liegen keine lokalen Daten vor oder wird eine Aktualisierung explizit gewünscht, erfolgt eine Kontaktaufnahme mit der externen Datenquelle, sowie der Abruf der Daten. Die zwischengespeicherten Daten sind in einem

Algorithmus 1 : Ablauf der Synchronisation auf Modell-Ebene

Data : Zeitstempel letzte Synchronisation t_0
Data : Zwischengespeicherte Daten *data*
Data : Gültigkeitsdauer N
Data : SPARQL-Anfrage q
Input : Aktueller Zeitstempel t

```
1 if data = NULL then                          /* Prüfe, ob bereits Daten vorliegen. */
2     data ← FetchData(q)
3 else
4     if t − t_0 > N then     /* Prüfe, ob die vorliegenden Daten noch gültig sind. */
5         data ← FetchData(q)
6     else
7         data ← data
```

lokalen RDF-Graphen abgelegt, der von anderen Graphen importiert werden kann. Eine Synchronisation ist deshalb trivial und verhält sich, wie in Algorithmus 1 dargestellt.

Tabelle 3.3 fasst die Anforderungen zusammen und ordnet sie den weiter oben definierten Ebenen der Datenintegration zu.

3.3.4. Integration heterogener Datenquellen in einer Wiki-Instanz

Eine weitere, aus den Anwendungsfällen abzuleitende Anforderung, betrifft die Möglichkeit, externe Datenquellen zur Laufzeit in das System einzubinden, um Daten aus anderen Wiki-Instanzen und Daten, die über SPARQL-Endpunkte bereitgestellt werden, zu integrieren. Um dieser Anforderung gerecht zu werden, müssen zwei Voraussetzungen erfüllt sein:

1. Es muss eine Möglichkeit geben, verschiedene RDF-Graphen unterschiedlichen Datenquellen zuzuordnen.
2. Da die lokalen Daten in der Regel in einer relationalen Datenbank oder einem Tripel-Store abgelegt sind, muss es möglich sein, dass heterogene Datenhaltungssysteme parallel angesprochen werden können.

Das zu entwickelnde System sollte zusätzlich zu den evtl. konfigurierbaren internen Datenquellen auch externe Datenquellen in Form von SPARQL-Endpunkten unterstützen, um eine

Kommunikation auf Modell-Ebene zu ermöglichen. Spezielle Funktionen, die keinem offenen Standard zugeordnet werden können und deshalb nur für die Kommunikation zwischen Instanzen des gleichen Wiki-Systems nutzbar sind, sind auf der Interwiki-Ebene ebenfalls zu unterstützen. Sämtliche Datenquellen müssen vom System gekapselt werden, so dass die Applikation weiterhin mit nur einer Datenquelle kommuniziert, die intern für die Beschaffung der notwendigen Daten verantwortlich ist. Die so entstehende Meta-Datenquelle muss demnach die folgende Funktionalität zur Verfügung stellen:

- Integration verschiedener lokaler und externer Datenquellen in einer Datensicht,
- Weiterleitung sämtlicher Anfragen an die jeweils zuständige Datenquelle,
- Integration der Ergebnismengen,
- ggf. Sortierung und Begrenzung der Ergebnismengen.

Um sicherzustellen, dass die für die Applikation notwendigen Systemdaten verfügbar sind, setzt das zu entwickelnde System eine konfigurierbare, lokale Datenquelle voraus und alle zusätzlichen lokalen und externen Datenquellen lassen sich über Konfigurationsparameter aktivieren. Für externe Datenquellen müssen

- die URI des zu befragenden Endpunktes,
- ggf. die URI eines Graphen innerhalb des Endpunktes, sowie
- ggf. eine SPARQL-Anfrage, welche die Ergebnismenge reduziert

konfiguriert sein.

Ebene	Modell-Ebene, Interwiki-Ebene
Eingabe	Konfiguration von unterschiedlichen Datenquellen
Ziel	Integration heterogener Datenquellen in einer Datensicht
Subanforderungen	Kapselung verschiedener Datenquellen in einer Meta-Datenquelle, parallele Nutzbarkeit verschiedener Datenhaltungssysteme, Unterstützung von SPARQL-Endpunkten, Unterstützung von Wiki-Endpunkten

Tabelle 3.4.: Zusammenfassung der Anforderung zur Integration heterogener Datenquellen in einer Wiki-Instanz

Für Endpunkte, bei denen eine einschränkende SPARQL-Anfrage konfiguriert ist, existiert ein lokales Modell, das die Ergebnisse der Anfrage zwischenspeichert, um ein effizientes Laufzeitverhalten der Applikation zu ermöglichen. Diese Daten werden in gewissen Zeitabständen aktualisiert.

Tabelle 3.4 fasst die Anforderungen zusammen und ordnet sie den weiter oben definierten Ebenen der Datenintegration zu.

3.3.5. Authentifizierung auf UI- und API-Ebene

Abschließend lässt sich aus den Anwendungsfällen herleiten, dass das zu entwickelnde System die Authentifizierung sowohl auf UI-Ebene, als auch auf API-Ebene unterstützen sollte. Die Authentifizierung auf UI-Ebene wird von allen gängigen Wiki-Systemen unterstützt. Allerdings müssen zu diesem Zweck in der Regel separate Nutzeraccounts auf allen Instanzen des jeweiligen Systems eingerichtet werden. Dies hat gerade im Hinblick auf das Szenario, dass Daten auf vielen verschiedenen Systemen verteilt sind, den Nachteil, dass sich der Nutzer diverse Zugangsdaten merken muss.

Auf API-Ebene unterstützen einige der existierenden Wiki-Applikationen die im HTTP-Protokoll vorgesehene Basis-Authentifizierung. Dieses Verfahren ist generell nur auf geschützten HTTP-Kanälen anwendbar, da mit jeder Anfrage vertrauliche Nutzerdaten übertragen werden. Diese Tatsache in Verbindung mit den diversen Nutzeraccounts, die ein Endnutzer benötigt, machen eine sinnvolle Nutzung in Bezug auf eine Interwiki-Kommunikation unpraktikabel.

Geeignete Lösungen für beide Probleme sind daher in das zu entwickelnde System zu integrieren. Auf UI-Ebene sollte das System den OpenID-Standard unterstützen, da dieses Protokoll inzwischen weit verbreitet ist und die Wiederverwendung eines Nutzeraccounts auf verschiedenen Systemen ermöglicht. Das System sollte dabei zum Einen OpenIDs bereitstellen, also lokalen Nutzern bei der Registrierung eine OpenID zuweisen (Provider). Zum Anderen sollte es aber in jedem Fall existierende OpenIDs bei einer Registrierung akzeptieren (Consumer) und den jeweiligen Nutzer fortan mit dieser authentifizieren. Zusätzlich sollte auch die Möglichkeit bestehen, Nutzer mit Hilfe des FOAF+SSL-Protokolls zu registrieren und zu authentifizieren, insbesondere, da FOAF+SSL auf Semantic Web Standards basiert und in Bezug auf OpenID eine Reihe zusätzlicher Möglichkeiten bietet. Auf die Integration von OpenID ist dennoch nicht zu verzichten, da die Adaption dieses Protokolls im Vergleich zu FOAF+SSL sehr viel stärker ist, zumindest zum heutigen Zeitpunkt. Auf diese Weise wird

eine möglichst hohe Interoperabilität mit anderen Web-Applikationen gewährleistet.

Für die Authentifizierung auf API-Ebene könnte der OAuth-Standard (s. Kapitel 4) in Verbindung mit OpenID zum Einsatz kommen. Hierbei weist der Nutzer einer Applikation, die auf Daten zugreifen soll, bestimmte Rechte zu. Die Zuweisung dieser Rechte findet auf der Applikation statt, welche die schutzbedürftigen Informationen enthält. Mit Hilfe kryptographischer Verfahren wird sichergestellt, dass die Applikationen ohne direkte Nutzerbeteiligung geschützte Daten austauschen können. Im initialen Freigabeprozess ist der Nutzer jedoch direkt involviert, weshalb auch hier ein Verfahren wie OpenID zum Einsatz kommen sollte. Es ist ebenfalls denkbar, dass auf API-Ebene eine modifizierte Variante des FOAF+SSL-Protokolls verwendet wird. Das hätte den Vorteil, dass der gesamte Authentifizierungsprozess mit nur einem Protokoll umgesetzt werden könnte.

Ebene	Modell-Ebene, Interwiki-Ebene
Eingabe	notwendige Nutzerinformationen
Ziel	Authentifizierung auf UI- und API-Ebene
Subanforderungen	Wiederverwendung von Nutzeraccounts

Tabelle 3.5.: Zusammenfassung der Anforderungen zur Authentifizierung auf UI- und API-Ebene

Tabelle 3.5 fasst die Anforderungen zusammen und ordnet sie den weiter oben definierten Ebenen der Datenintegration zu.

3.4. Nichtfunktionale Anforderungen

Im folgenden Abschnitt werden Anforderungen zusammengefasst, die keiner Funktion zugeordnet werden können, aber dennoch Anforderungen an das zu entwickelnde System darstellen.

3.4.1. Bedienbarkeit

Die Popularität von Wiki-Applikationen ist nicht zuletzt darauf zurückzuführen, dass auch Anwender ohne technischen Hintergrund solche Applikationen sehr schnell bedienen können. Das zu entwickelnde System sollte demnach auch durch Nutzer ohne Wissen über

semantische Technologien nutzbar sein. Überdies sollte es den anspruchsvollen Nutzer dabei unterstützen, seine Aufgaben schnell und korrekt erledigen zu können.

3.4.2. Architektur

Die Architektur des Systems muss so beschaffen sein, dass die Erweiterbarkeit in allen relevanten Bereichen gewährleistet ist. Insbesondere sollten nach Möglichkeit bereits existierende Erweiterungsmechanismen der Ziel-Applikation wiederverwendet werden, um die Komplexität des Systems nicht unnötig zu erhöhen. An Stellen, an denen es sinnvoll erscheint, sollte das System selbst als Erweiterung realisiert werden, damit der Kern der eigentlichen Applikation nicht über das Notwendige hinaus erweitert werden muss. Daher kann das System bzw. können Teile des Systems später je nach Bedarf deaktiviert werden, ohne die eigentliche Applikation zu beeinträchtigen.

3.4.3. Performanz

Die Performanz betreffend, darf das zu entwickelnde System die Leistung der Zielapplikation nicht beeinträchtigen. Ressourcenintensive Funktionen müssen demnach in einer Weise ausgeführt werden, die das Antwortverhalten der Applikation nicht negativ beeinflusst. Zusätzlich müssen Maßnahmen getroffen werden, welche die Kosten bei einer wiederholten Ausführung reduzieren, z. B. durch Zwischenspeicherung von Ergebnissen.

3.4.4. Sicherheit

Im Hinblick auf die Sicherheit, muss das zu entwickelnde System die Sicherheitsanforderungen der Zielapplikation erfüllen. Insbesondere sind die Zugriffsrechte auf Ressourcen stets zu beachten. Das betrifft vor allem das Schreiben von Daten innerhalb der Applikation. Ein Nutzer, der bspw. nur Leserechte an einer bestimmten Ressource besitzt, darf durch das System niemals dazu befähigt werden, diese Ressource zu modifizieren.

4. Gegenwärtiger Stand der Technik

Das vorliegende Kapitel fasst den gegenwärtigen Stand der Technik zusammen. Es werden zum Einen Publikationen berücksichtigt, die für eine Datenintegration relevante Facetten thematisieren und Lösungen oder Lösungsansätze liefern. Zum Anderen werden Prototypen und Werkzeuge vorgestellt und deren Funktionalität bezüglich einer Integration von Informationen aus externen Quellen evaluiert. Zunächst erfolgt eine Darstellung von Publikationen und Standards bezogen auf die Semantic Web Basisinfrastruktur. Im zweiten Abschnitt stehen konkrete Dienste, Werkzeuge und Wikis im Fokus der Betrachtung.

4.1. Semantic Web Basisinfrastruktur

OKKAM Entity Name System

Bei OKKAM handelt es sich um ein Forschungsprojekt, das von der Europäischen Union im Rahmen des Seventh Framework Programme gefördert wird. Ein erklärtes Ziel des Projekts ist die Schaffung eines offenen Dienstes, der die Wiederverwendung von Entitäten ermöglicht und fördert. In [Bouquet et al., 2007] und [Bouquet et al., 2008] stellen die Autoren das Entity Name System (ENS) vor, ein auf das Internet zugeschnittenes System, um der von den Autoren attestierten Identitätskrise des Semantic Web zu begegnen. Dazu werden sog. Entitäten und logische Ressourcen unterschieden. Logische Ressourcen beschreiben abstrakte Objekte, wie Eigenschaften, Relationen und Zuweisungen. Das ENS soll dabei bewusst nicht die Verwendung gleicher URIs für logische Ressourcen vorschreiben, da auch innerhalb einzelner Domänen teilweise sehr unterschiedliche Sichtweisen existieren können. Stattdessen soll das System die Wiederverwendung von URIs für Entitäten ermöglichen, da eine solche bisher lediglich aus praktischen Gründen nicht angewendet wird. Als Beispiel führen Sie Protégé auf, einen weit verbreiteten Ontologie-Editor. Dieser erzeugt für jede neue Instanz innerhalb einer Ontologie eine neue URI. OKKAM will dieser „schlechten Praxis“ entgegenwirken und stellt dafür u. a. einen Plugin für Protégé bereit, das neu erzeugten

Instanzen eine vom ENS bereitgestellte URI zuweist. Um eine Zuordnung zu einer URI zu ermöglichen, speichert OKKAM informelle Beschreibungen für Entitäten. Für eine Entität des Typs Person sind das z. B. der Name und die Zugehörigkeit zu einer Organisation. Dabei legen die Autoren Wert darauf, dass es sich um untypisierte Daten handelt, die den Nutzer dabei unterstützen sollen, festzustellen, ob es sich um die gewünschte Entität handelt. Zusätzlich können allerdings sog. ontology references angelegt werden. Solche Referenzen sind URIs zu externen Datenquellen, die weitere Informationen über die Entität enthalten.

Eine prototypische Realisierung von OKKAM besteht aus dem OKKAM-Core, der grundlegende Dienste, wie das Anlegen und Speichern von URIs bereitstellt und die Bearbeitung der Entitäts-Beschreibungen (Profile) ermöglicht. Aufbauend auf diesem Kern, wird eine Reihe von Diensten zur Verfügung gestellt, die u. a. die Suche nach existierenden Entitäten mit Hilfe von unterschiedlichen Kriterien ermöglichen. Entitäten werden zum Einen durch einen (semi-)automatischen Prozess erzeugt, bei denen existierende Datenquellen, wie Listen (z. B. Wikipedia) und Datenbanken abgerufen oder angefragt werden, um solche Informationsbausteine auf Entitäten abzubilden. Mögliche Konflikte werden dabei von einem menschlichen Benutzer aufgelöst. Zum Anderen ist eine manuelle Eingabe von Entitäten über eine Web-Schnittstelle, sowie eine API möglich.

Evaluiert wurde der Prototyp mit zwei RDF-Datensätzen, die Daten zur International Semantic Web Conference (ISWC) und zur European Semantic Web Conference (ESWC) enthielten, zwei angesehenen Konferenzen im Semantic Web Kontext. Diese Datensätze verwendeten durchweg unterschiedliche URIs. Ziel hierbei war es festzustellen, welche Personen an beiden Konferenzen teilgenommen haben. Hierfür wurden sämtliche Bezeichner durch OKKAM-URIs ersetzt. Anschließend wurde manuell überprüft, wie viele der tatsächlichen Übereinstimmungen durch OKKAM entdeckt wurden. Es stellte sich heraus, dass eine vollständige und korrekte Ergebnismenge generiert wurde, wobei die Autoren zugeben, dass es sich um ein stark konstruiertes Szenario handelt.

Das ENS ist als föderierte Architektur realisiert, bestehend aus globalem OKKAM-Dienst und optionalen lokalen OKKAM-Knoten, die zusätzlich private Entitäten enthalten können, aber auch Eigenschaften, die nicht der Öffentlichkeit zugänglich gemacht werden. Somit werden auch Sicherheits- und Besitzaspekte thematisiert. Sämtliche Bezeichner, die von OKKAM erzeugt werden, sind absolute URIs, welche die Beschreibung der jeweiligen Entität über Linked Data bereitstellen.

Zusammenfassend lässt sich feststellen, dass das ENS für eine Ressourcen-Wiederverwendung sehr gut geeignet ist. Über die bloßen Eigenschaften einer Entität, lässt

sich auf diese Weise feststellen, ob ein Bezeichner bereits existiert und wiederverwendet werden kann. Allerdings sind sämtliche über das ENS bereitgestellte Entitäten-Bezeichner URIs aus dem OKKAM-Namensraum. Diese stellen zwar Daten über Linked Data bereit, jedoch in einem sehr begrenzten Rahmen. Demnach eignet sich das OKKAM ENS im Kontext dieser Arbeit nur bedingt, da insbesondere die Integration von Daten, die über Linked Data bereitgestellt werden, von großer Relevanz ist.

Sindice

Sindice[20] ist ein Dienst für die Suche nach RDF-Dokumenten, die Aussagen über bestimmte Ressourcen treffen [Tummarello et al., 2007a]. Zu diesem Zweck können Ressourcen über

- URIs,
- Inverse Functional Properties (IFPs) und
- Schlüsselwörter

identifiziert werden. Dies macht es zum Einen möglich, alternative Datenquellen zu finden, wenn eine URI bereits gegeben ist. Zum Anderen können über IFPs und Schlüsselwörter RDF-Dokumente gefunden werden, die über relevante Ressourcen Aussagen treffen. Im Fall von IFPs hat dieses Vorgehen den Vorteil, dass auch für nicht dereferenzierbare Bezeichner, wie bspw. Emailadressen oder ISBN-Nummern, RDF-Daten gefunden werden können. Bei der Suche über Schlüsselwörter wird es gar erst möglich, relevante Ressourcen aufzuspüren, welche die gewählten Terme verwenden. In Verbindung mit dem Linked Data Paradigma können URIs identifiziert werden, über die Daten direkt abrufbar sind. Damit wird eine URI-Wiederverwendung gefördert, aber nicht erzwungen, wie es z. B. beim OKKAM ENS der Fall ist. Sämtliche Aussagen, die über derart gefundene Ressourcen getroffen werden, können später selbst von Sindice indiziert werden, insofern sie öffentlich zugänglich sind. Somit können Ressourcen durch eine URI-Wiederverwendung um alternative Datenquellen ergänzt werden. Die Autoren führen diesbezüglich das Beispiel einer Produktsuche an. Möchte ein Individuum sich bspw. über ein bestimmtes Mobiltelefon informieren, so würde ein sehr einseitiges Bild entstehen, wenn dafür lediglich die Informationen des Herstellers in Betracht gezogen werden.

Sindice wurde in erster Linie nicht für den Endanwender entwickelt, sondern als Dienst, der es anderen Diensten und Applikationen ermöglicht, relevante RDF-Dokumente aufzuspü-

[20] `http://sindice.com`

ren. Zu diesem Zweck stellt Sindice verschiedene Programmierschnittstellen bereit, die dem REST-Paradigma folgen. Des Weiteren können sämtliche Ergebnisse im JSON-, RDF/XML und zusätzlich im ATOM[21]-Format abgerufen werden. Bei ATOM handelt es sich um ein Format zur Syndizierung von Inhalten. Unter diesen Voraussetzungen ist eine Verwendung der Sindice-Dienste in einem sehr breiten Anwendungsspektrum möglich. Zusätzlich steht eine HTML-Schnittstelle bereit, über die Endanwender ebenfalls den Sindice-Suchdienst nutzen können.

Sämtliche Ergebnisse sind gewichtet und in Bezug auf diese Gewichtung absteigender Reihenfolge sortiert. Für die Berechnung der Gewichtung werden u. a. die Hostnamen der URIs mit einbezogen. Dabei gelten solche Dokumente als besonders relevant, dessen Hostnamen mit den verwendeten Ressourcen übereinstimmen, also folglich als Linked Data zur Verfügung stehen. Zusätzlich werden Techniken aus dem Bereich des Information Retrieval angewendet.

Die Architektur von Sindice ist so beschaffen, dass eine möglichst hohe Performanz und Skalierbarkeit erreicht werden kann. Diesbezüglich werden drei nichtfunktionale Anforderungen an das System definiert:

1. Minimale Indexgröße in Bezug auf Speicherplatz,
2. möglichst kurze Antwortzeiten,
3. kontinuierliche Aktualisierung des Indexes.

Um diese Ziele zu erreichen, wird das System in Komponenten aufgeteilt, die unabhängig operieren. Die einzelnen Komponenten sind in eine sog. indexing pipeline und eine quering pipeline aufgeteilt. Die indexing pipeline ist u. a. für

- das Parsen der Dokumente,
- die Extraktion der URIs, IFPs und Schlüsselwörter,
- Linked Data Extraktion und
- die Aktualisierung der entsprechenden Indizes verantwortlich.

Die quering pipeline beinhaltet u. a. die Komponenten für eine Gewichtung der Ergebnisse und die Konvertierung der Ergebnisse in das gewünschte Ausgabeformat.

Sindice kann zum Einen mit Daten umgehen, die über Linked Data bereitstehen. Zum Anderen kann der Dienst aber auch mit SPARQL-Endpunkten und RDF-Datensätzen arbeiten.

[21] `http://www.atompub.org/rfc4287.html`

Diesbezüglich können Datenquellen eine Semantic Sitemap [Cyganiak et al., 2007] definieren, die u. a. Angaben zu verfügbaren SPARQL-Endpunkten und RDF-Exporten der Daten enthalten kann.

Im direkten Vergleich mit dem OKKAM ENS bietet Sindice den Vorteil, dass es sich bei sämtlichen Ergebnissen einer Sindice-Suchanfrage um abrufbare Links zu RDF-Daten oder RDF-ähnlichen Daten handelt. Auf diese Weise lassen sich nicht nur relevante URIs für Ressourcen identifizieren, sondern es können bereits vorhandene Informationen über solche Ressourcen weiterverwendet werden. Daneben stellt Sindice für Ergebnisse einen Titel zur Verfügung, der es den Nutzern auch ermöglicht, Ressourcen zu unterscheiden, wenn sie nicht mit dem Konzept der URI vertraut sind.

MOAW und Watson

In [Gridinoc and d'Aquin, 2008] stellen die Autoren Meaning of a Word (MOAW) vor, ein Werkzeug, das die Suche nach URIs in HTML-Seiten ermöglicht, in denen ein Eingabefeld existiert. MOAW wurde ursprünglich entwickelt, um in einer nächsten Version des Linked Data Browsers Tabulator eingesetzt zu werden, mit dem Zweck, semantische Annotationen schnell und einfach bearbeiten zu können. Es stellt sich jedoch heraus, dass MOAW überall dort eingesetzt werden kann, wo ein HTML-Formular existiert und die Nutzung von URIs sinnvoll erscheint.

Das Werkzeug wurde über ein JavaScript realisiert, ein sog. Bookmarklet. Ein Bookmarklet besteht aus JavaScript-Code, der in ein HTML-Anker-Element eingebettet ist und somit u. a. in der Lesezeichenleiste eines Browsers abgelegt werden kann. Beim Klick auf einen solchen Link, wird der JavaScript-Code ausgeführt. Im Fall von MOAW wird zu diesem Zweck jedes in einer HTML-Seite vorhandene `<input>`-Element mit einer Funktion belegt, die eine Auto-Vervollständigung ermöglicht. MOAW basiert auf jQuery[22] und dem jQuery auto-complete-Plugin[23], das für die Nutzung eines Dienstes zur URI-Suche modifiziert wurde.

Für die Suche selbst kommt Watson[24] zum Einsatz, ein Dienst, der ebenfalls eine Suche nach Entitäten und RDF-Dokumenten über Schlüsselwörter ermöglicht. Zusätzlich können Metadaten zu gefundenen Entitäten abgefragt werden, um Evidenz zu erhalten, dass es

[22] `http://jquery.com/`
[23] `http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/`
[24] `http://watson.kmi.open.ac.uk/Overview.html`

sich um das gewünschte Konzept handelt. Watson stellt sowohl ein Web-Formular, als auch verschiedene Programmierschnittstellen bereit, u. a. eine REST-API, so dass jede Suchanfrage über eine URL definiert werden kann.

FOAF+SSL

FOAF+SSL [Story et al., 2009] bezeichnet ein Verfahren zur Authentifizierung auf der Basis von Webstandards. Es ist so konzipiert, dass ein vollständig verteiltes System für Web-Identitäten möglich ist, das komplett auf zentrale Instanzen verzichtet. Zu diesem Zweck wird jeder Identität eine URI zugewiesen, in diesem Kontext auch WebID genannt. Daneben basiert es auf dem HTTP-Protokoll, d. h. die URIs lassen sich über HTTP dereferenzieren (`http://`-URIs). Das FOAF+SSL-Protokoll setzt voraus, dass Daten, die über eine WebID abgerufen werden, nur vom Eigentümer der URI, in den meisten Fällen also dem Eigentümer der Domain modifiziert und bereitgestellt werden können. Für den eigentlichen Vorgang zur Authentifizierung werden SSL bzw. TLS und Zertifikate verwendet. SSL/TLS Verbindungen sichern im heutigen Internet die meisten Transaktionen ab, bei denen schutzbedürftige Informationen ausgetauscht werden. Dabei werden vor allem serverseitige Zertifikate eingesetzt, die über kryptographische Verfahren in Verbindung mit einem streng hierarchischen System zur Schlüsselverteilung, die Vertrauenswürdigkeit des Anbieters sicherstellen. Der Nutzer authentifiziert sich in den meisten Fällen jedoch durch eine Kombination aus Benutzername und Passwort, obwohl das Verfahren auch Zertifikate auf Nutzerseite vorsieht. Dieser Umstand ist der Tatsache geschuldet, dass die Signierung von Zertifikaten in einem solchen System von einer vertrauenswürdigen Stelle vorgenommen werden muss, was in der Regel mit einem finanziellen Aufwand verbunden ist.

Bei FOAF+SSL werden Zertifikate auf Nutzerseite eingesetzt, wobei auf ein strikt hierarchisches System zur Signierung verzichtet wird. Stattdessen können Zertifikate von beliebigen Instanzen signiert werden, inklusive dem Nutzer selbst. Im Gegenzug wird die URI der Identität fest in das Zertifikat integriert. Dies wird mit Hilfe sog. Subject Alternative Names (SAN) realisiert, einer Erweiterung, die mit Version 3 des X.509-Standards [Cooper et al., 2008] für Zertifikate eingeführt wurde. Durch die Nutzung eines solchen Zertifikats, lassen sich von einem Server, eine SSL/TLS gesicherte Verbindung vorausgesetzt, die folgenden Aussagen treffen:

1. Die anfragende Partei (Client) besitzt den privaten Schlüssel zu dem öffentlichen Schlüssel, der im benutzten Zertifikat enthalten ist. Dies wird durch SSL/TLS sicherge-

stellt, da eine Verbindung sonst fehlgeschlagen wäre.

2. Der Client behauptet, die Identität zu besitzen, die über den SAN festgelegt wurde.

Der Server versucht, mit Hilfe der gegebenen WebID, RDF-Daten abzurufen. Dies erfolgt ausschließlich über die angegebene URI, an die nur der Eigentümer Daten binden kann. Abschließend muss auf Serverseite sichergestellt werden, dass in den abgerufenen Daten Informationen zu dem verwendeten öffentlichen Schlüssel enthalten sind, für den der Nutzer den privaten Schlüssel besitzt. Ist der dem Server zur Verfügung stehende Schlüssel in diesen Daten enthalten, kann der Server verifizieren, dass der verbundene Client dem Nutzer entspricht, der über die WebID definiert ist.

In einem zweiten Schritt erfolgt eine Autorisierung. Es ist nicht erforderlich, dass der Server dabei dem FOAF+SSL-Protokoll folgt, vielmehr kann ein beliebiges Verfahren Anwendung finden, z. B. kann auf serverseitig gehaltene Zugriffslisten zurückgegriffen werden, um zu überprüfen, ob ein Nutzer autorisiert ist, auf eine bestimmte Ressource zuzugreifen. Das FOAF+SSL-Protokoll sieht hierfür die Verwendung des FOAF-Vokabulars vor, um ein Vertrauensnetz (Web of Trust) zu schaffen.

OpenID

OpenID[25] nutzt wie FOAF+SSL URIs zur Authentifizierung, arbeitet dabei aber wesentlich weniger dezentral, wobei ein vollständig dezentrales Szenario grundsätzlich möglich wäre. Allerdings müsste dann jeder OpenID-Nutzer seinen eigenen OpenID-Provider bereitstellen, was mit einem wesentlich höheren Aufwand verbunden ist, als das Hinterlegen eines RDF-Dokuments.

Möchte sich ein Nutzer an einer Webseite anmelden, die eine Anmeldung per OpenID unterstützt, so gibt er eine ihm zugeordnete OpenID in ein entsprechendes Eingabefeld ein und der Server (Relying Party) ermittelt den OpenID-Provider des Nutzers. Anschließend stellt er eine Verbindung zum Provider her, um mit diesem mit Hilfe von kryptographischen Methoden ein „Geheimnis“ für den aktuellen Authentifizierungsvorgang zu vereinbaren. In einem nächsten Schritt wird der Nutzer auf die Webseite des Providers geleitet, um dort ggf. seine Identität zu beweisen. Bei einer erfolgreichen Authentifizierung auf Provider-Seite, wird der Nutzer an die Relying Party weitergeleitet und ist dort, nachdem die Echtheit der

[25] `http://openid.net/`

Anfrage über kryptographische Verfahren sichergestellt wurde, authentifiziert. Des Weiteren definiert OpenID Methoden, um bei einem Registrierungsprozess Nutzerinformationen austauschen zu können. OpenID wird mittlerweile auf mehr als 40.000 Webseiten erfolgreich eingesetzt.

OAuth

OAuth [Atwood et al., 2009] ist ein Verfahren zur API-basierten Authentifizierung eines Nutzers. Im Gegensatz zu anderen Verfahren, bei denen die Applikation die Zugangsdaten des Nutzers kennen muss, können Applikation und Datenquelle beim OAuth-Protokoll mit Hilfe von kryptographischen Verfahren ohne eine Übertragung dieser sensiblen Daten kommunizieren. Der Nutzer muss dafür lediglich einer Nutzung der angefragten Daten auf Seite der Datenquelle zustimmen. Die OAuth-Spezifikation definiert zum Einen Methoden, um unsichere Verbindungen abzusichern, zum Anderen kann die Komplexität einer Verifikation der Echtheit von Anfragen aber auch delegiert werden, indem die Kommunikation über einen SSL/TLS gesicherten Kanal stattfindet.

Zusammenfassend betrachtet, bietet das FOAF+SSL-Protokoll den Vorteil, dass es auf Semantic Web Technologien basiert und bei richtiger Umsetzung den größten Nutzerkomfort ermöglicht, da hier weder Nutzername noch Passwort vom Endnutzer bereitgestellt werden müssen. Allerdings gestaltet sich der Umgang mit Zertifikaten in heutigen Webbrowsern für den Laien noch etwas umständlich, so dass diesbezüglich Probleme auftreten können. Außerdem müssen zunächst Systeme etabliert werden, die eine effektive Zertifikatsverwaltung ermöglichen, insbesondere in Bezug auf die Invalidierung von Zertifikaten, da sonst eine Nutzung außerhalb des persönlichen Webbrowsers unpraktikabel erscheint. Im Gegensatz dazu muss sich der Nutzer bei OpenID zwar einen Nutzernamen und ein Passwort merken, jedoch entfällt dafür die Zertifikatsverwaltung. OpenIDs können also in jedem Webbrowser verwendet werden. OAuth stellt momentan die einzige, standardisierte Möglichkeit für eine Authentifizierung auf API-Ebene dar. Das FOAF+SSL-Protokoll könnte jedoch leicht erweitert werden, um ebenfalls eine Authentifizierung auf API-Ebene zu realisieren.

4.2. Dienste, Werkzeuge und Wikis

Tabulator und Tabulator Redux

Tabulator ist ein generischer RDF-Browser, der es ermöglicht, RDF-Daten über URLs zu dereferenzieren. Er wird u. a. von Tim Berners-Lee am Massachusetts Institute of Technology (MIT) entwickelt und in [Berners-Lee et al., 2006a] erstmals vorgestellt. Die Applikation ist vollständig in JavaScript realisiert und kann deshalb in jedem modernen Webbrowser ausgeführt werden. Der Tabulator arbeitet in zwei unterschiedlichen Modi,

- dem Explorer-Modus und
- dem Analyse-Modus.

Im Explorer-Modus kann der Nutzer verfügbare Daten in einer Baumstruktur betrachten. Die einzelnen Knoten entsprechen dabei den gefunden URIs. Wird ein Knoten, durch Klick auf ein entsprechendes Symbol geöffnet, so können sämtliche zugehörige Prädikat/Objekt-Paare betrachtet werden. Handelt es sich bei einem Objekt um einen Verweis auf eine Ressource, so wird diese automatisch dereferenziert und kann, falls Daten verfügbar sind, ebenfalls detailliert betrachtet werden. Im Analyse-Modus können an den lokal erzeugten Graphen SPARQL-Anfragen gestellt werden und die Ergebnisse lassen sich in verschiedenen Sichten darstellen. So ist es zum Einen möglich, die selektierten Daten in einer Tabelle aufbereiten zu lassen. Zum Anderen können bestimmte Daten auch auf einer Karte oder in einer Kalenderansicht präsentiert werden. Die initiale Auswahl der Daten erfolgt nach dem Query-By-Example-Prinzip, indem der Nutzer Knoten des Baumes auswählt und so ein Muster erzeugt, mit dessen Hilfe die Daten gefiltert werden können. Dieses Muster ist als SPARQL-Anfrage repräsentiert und kann vom erfahrenen Benutzer später modifiziert werden.

Tabulator dereferenziert URLs, wenn der Nutzer eine Ressource im Explorer-Modus betrachtet. Dies ist zu Beginn die URL, mit welcher der Nutzer das Browsen begonnen hat. Zusätzlich werden alle URLs überprüft, die als Objekt in einem Statement verwendet werden. Solche URIs können dann, falls Daten verfügbar sind, ebenfalls exploriert werden. Wenn mindestens eine der folgenden Voraussetzungen erfüllt ist, kann die Applikation Daten zu einer gegebenen URL extrahieren:

1. Die Daten sind als Linked Data verfügbar und können somit mit einem `Accept: application/rdf+xml` HTTP-Header abgerufen werden. Tabulator akzeptiert in diesem Fall XML-serialisierte RDF-Daten, wobei das umschließende `<rdf:RDF>`-Element

fehlen darf, da das Format durch den Header definiert ist.

2. Liegen die Daten als (X)HTML vor, wird nach <link>-Elementen gesucht, um auf den Ort und die Art der strukturierten Daten zu schließen.
3. Wenn das abgerufene Dokument die Verwendung von GRDDL [Connolly, 2007] anzeigt, so werden die Daten entsprechend der GRDDL-Spezifikation extrahiert. GRDDL steht für Gleaning Resource Descriptions from Dialects of Languages und hat den Status einer W3C-Recommendation. Es ermöglicht die Extraktion von RDF-Daten aus XML- und XHTML-Dokumenten mittels Transformationen.

Zusätzlich finden innerhalb der Applikation einige Inferenz-Prozesse statt. So werden bspw. `owl:sameAs`-Links ausgewertet, d. h. entsprechend deklarierte Knoten werden fusioniert. Des Weiteren werden Knoten zusammengeführt, die identische Werte für Functional Properties oder Inverse Functional Properties aufweisen.

In [Berners-Lee et al., 2007] stellen die Autoren um Tim Berners-Lee unter dem Titel Tabulator Redux eine erweiterte Version der Tabulator-Applikation vor, die es ermöglichen soll, Daten innerhalb des Browsers zu bearbeiten und die jeweiligen Quellen zu aktualisieren. Dabei adressieren die Autoren auch Probleme, die im Zusammenhang mit der Herkunft der Daten auftreten können. So besteht der lokale Graph, den der Nutzer durch das Browsen im Semantic Web erzeugt, aus der Vereinigung vieler Teilgraphen. Zusätzlich können auch Statements enthalten sein, die durch Inferenz hinzugefügt wurden. Diesen Problemen wird in Tabulator begegnet, indem die Subjekt-URI eines Statements die Datenquelle bezeichnet. Der von den Autoren skizzierte Prototyp ignoriert aus Komplexitätsgründen Fragen der Zugriffskontrolle und wählt einen offenen Ansatz für Schreibrechte auf die Testinstanz. Jeder ist berechtigt, in einem definierten Namensraum Ressourcen anzulegen und zu bearbeiten. Wird auf eine Ressource zugegriffen, die bisher nicht existiert, so wird für diese Ressource eine leere Datei erzeugt und das zunächst leere Ergebnis ausgeliefert. Für den eigentlichen Schreibvorgang wird das HTTP-Protokoll um einen nicht standardisierten Header der Form `MS-Author-Via: WebDAV` bzw. `MS-Author-Via: SPARQL` erweitert, wobei im ersten Fall die Aktualisierung über WebDAV realisiert wird. Im zweiten Fall dagegen kommt SPARQL/Update [Seaborne and Manjunath, 2007], eine nicht-standardisierte Erweiterung zu SPARQL, zum Einsatz.

Weitere generische Semantic Data Browser sind u. a. Magpie [Dzbor et al., 2003], Haystack [Quan et al., 2003] und Piggy Bank [Huynh et al., 2007], wobei diese allesamt zunächst

lokal installiert werden müssen. Disco[26] ist ein weiterer webbasierter Ansatz, der aber im Gegensatz zu Tabulator Ressourcen serverseitig dereferenziert.

Sindice RDFizer API

Sindice bietet zusätzlich zu den Suchfunktionen einen Dienst, der es ermöglicht, RDF-Daten zu beziehen, die durch die Extraktion von Microformats oder RDFa [Adida and Birbeck, 2008] aus HTML-Dokumenten generiert werden. Auf diese Weise können RDF-Daten auch dann gewonnen werden, wenn entsprechende Parser nicht zur Verfügung stehen. Durch die weite Verbreitung von Microformats im Web 2.0 Kontext und die steigende Akzeptanz, kann die Menge an verfügbaren RDF-Daten so enorm gesteigert werden. Der Dienst wird, wie auch die Suchfunktion über eine REST-Schnittstelle zur Verfügung gestellt. Es genügt also eine URI der Form

```
http://api.sindice.com/rdfizer/v1/FORMAT/HOST
```

aufzurufen, wobei `FORMAT` abhängig vom gewünschten Ziel-Format durch `rdf, xml, n3, ntriples` oder `turtle` ersetzt werden muss. Der `HOST`-Parameter muss entsprechend durch die Ressourcen-URI ersetzt werden, die jeweils von Interesse ist.

OpenLink Virtuoso Sponger

Virtuoso Sponger ist Teil der kommerziellen Virtuoso Universal Server Platform[27], ein Produkt der Firma OpenLink, das u. a. einen performanten und skalierbaren Triple-Store bereitstellt. Sponger ist ein erweiterbares System mit dem Datenquellen, die nicht auf RDF basieren, integriert werden können. Es wird standardmäßig mit Unterstützung für diverse Dienste, wie z. B. Delicious[28] oder Flickr und Dateiformate, wie z. B. Microsoft Office und PDF ausgeliefert. Zusätzlich kann das System durch sog. Cartridges erweitert werden, wobei eine solche Erweiterung aus einem Metadata Extractor und einem Ontology Mapper besteht.

Insbesondere im Hinblick auf die Erweiterbarkeit von Sponger, deckt das System die Anforderung zur Anbindung von externen Datenquellen hinreichend gut ab. Da Sponger jedoch

[26] `http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/disco/`
[27] `http://www.openlinksw.com/virtuoso/`
[28] `http://delicious.com/`

direkt in Virtuoso integriert ist, kann es auch nur in Kombination mit diesem eingesetzt werden. Generell wäre im Hinblick auf diese Funktionalität aber die Unabhängigkeit von einem bestimmten Datenhaltungssystem wünschenswert, um das System flexibel zu halten.

Triplify und weitere Ansätze

Triplify ist ein leichtgewichtiger Ansatz, um Daten in relationalen Datenbanken als Linked Data verfügbar zu machen [Auer et al., 2009]. Dazu werden HTTP-URIs auf zuvor definierte SQL-Anfragen abgebildet und die resultierenden Ergebnisse in RDF konvertiert. Es existieren bereits Triplify-Konfigurationen für eine Vielzahl populärer Web-Applikationen, wie bspw. WordPress[29] und Drupal[30]. Außerdem wurde Triplify anhand des OpenStreetMap[31]-Projekts evaluiert. Im Juli 2008 verfügte das Projekt über 160 Gigabyte an Geodaten, die mit Hilfe von Triplify als Linked Data verfügbar gemacht wurden[32].

Ein weiteres Projekt, das sich mit der Extraktion von RDF-Daten befasst, ist Swignition[33], wobei zusätzlich zu Microformats und RDFa eine Vielzahl weiterer Formate unterstützt werden, u. a. auch RDF-Formate selbst. So können für Ressourcen, die über HTTP erreichbar sind, z. B. HTTP-Header Informationen, wie `Content-Type` oder `Last-Modified` extrahiert werden. Auch die verfügbaren Ausgabeformate gehen über bloße RDF-Serialisierungen weit hinaus. Informationen, die einen zeitlichen Bezug besitzen, können bspw. im iCalendar-Format [Dawson and Stenerson, 1998] exportiert werden, ein Standardformat zur Speicherung von Kalenderinformationen, das von allen gängigen Kalenderwerkzeugen unterstützt wird.

Das Projekt um Krextor [Lange, 2009] befasst sich ebenfalls mit der RDF-Extraktion, wobei hier versucht wird, eine erweiterbare Architektur zur Extraktion von Daten aus XML-basierten Sprachen zu schaffen. Der Fokus liegt dabei auf der einfachen Erweiterbarkeit, so dass neue domänenspezifische XML-Sprachen mit geringem Aufwand auf eine RDF-Repräsentation abgebildet werden können.

Ein Beispiel für eine domänenspezifische Extraktion von RDF-Daten ist RDFohloh. Ohloh[34] ist ein Web 2.0 Verzeichnis für Open Source Projekte. Es bietet, wie viele Web 2.0 Dienste,

[29] `http://wordpress.org/`
[30] `http://drupal.org/`
[31] `http://www.openstreetmap.org/`
[32] `http://linkedgeodata.org/`
[33] `http://buzzword.org.uk/swignition/`
[34] `http://www.ohloh.net/`

eine proprietäre Programmierschnittstelle, um auf bestimmte Daten des Dienstes zugreifen zu können. RDFohloh nutzt diese Schnittstelle, um Daten zu gewinnen und bildet diese auf das Description of a Project (DOAP)[35] RDF-Vokabular ab. Des Weiteren stellt es sämtliche Daten als Linked Data bereit.

Semantic MediaWiki (SMW) / SMW+

In [Krötzsch et al., 2006] stellen die Autoren Semantic MediaWiki (SMW) vor. SMW ist eine semantisch angereicherte Wiki-Applikation, die auf dem weit verbreiteten MediaWiki-System basiert. MediaWiki wird u. a. im Wikipedia-Projekt seit Jahren erfolgreich eingesetzt. Eines der Hauptziele bei der Entwicklung von SMW ist die Schaffung einer semantischen Wikipedia. Aus diesem Grund wird SMW als Erweiterung zu MediaWiki realisiert, d. h. es existieren weiterhin Artikel, die aus freiem Text bestehen. Der Nutzer hat mit SMW zusätzlich die Möglichkeit, gewisse strukturierte Informationen mit in das System einzupflegen. SMW setzt dabei auf eine Erweiterung der von MediaWiki genutzten textbasierten Wiki-Syntax, um eine semantische Annotation zu ermöglichen. Diese semantischen Annotationen werden in speziellen Datenbanktabellen gespeichert und können somit später leicht extrahiert werden. Es existieren zahlreiche Erweiterungen unter der Bezeichnung SMW+, die es u. a. ermöglichen, einen externen Triple-Store einzubinden. Durch die Abfrage einer solchen Datenquelle, bspw. mit SPARQL, können zusätzliche strukturierte Informationen zur Laufzeit in die Wiki-Seiten integriert werden. Daneben bietet SMW+ die Möglichkeit, Daten von beliebigen Webservices abzurufen und ebenfalls in das Wiki-System zu integrieren. SMW ist eine Open Source Software und ist unter der GNU General Public License (GPL) lizensiert.

IkeWiki / Knowledge in a Wiki (KiWi)

Ein weiteres semantisches Wiki ist IkeWiki [Schaffert, 2006]. In IkeWiki wird großer Wert auf die einfache Nutzbarkeit der Applikation gelegt, vor allem durch Nutzer ohne oder mit wenig technischem Hintergrund. Aus diesem Grund und aus Gründen der Kompatibilität setzt IkeWiki eine an MediaWiki angelehnte Wiki-Syntax ein. Zusätzlich kommt ein What You See Is What You Get (WYSIWYG) Editor zum Einsatz. Für semantische Annotationen werden Semantic Web Standards, wie RDF und OWL eingesetzt und in einem separaten Triple-Store abgelegt. Die Anwendung basiert auf Java und dem Jena Semantic Web Framework[36]. Zu-

[35] http://trac.usefulinc.com/doap
[36] http://jena.sourceforge.net/

sätzlich unterstützt die Applikation Multimediadaten, wie z. B. Bilder und Audiodateien. Bei Bildern werden zusätzlich evtl. enthaltene Metadaten zur Importzeit extrahiert. Audiodateien, wie bspw. MP3-Dateien werden erkannt und ein entsprechendes Abspielplugin zur Laufzeit angeboten. IkeWiki ist ebenfalls eine Open Source Software und ist unter der GPL lizensiert.

KiWi[37] ist ein auf IkeWiki basierendes semantisches Wiki mit Fokus auf strukturierte Informationen. Es basiert folglich ebenfalls auf JAVA und Jena, bietet aber zusätzlich viele Web 2.0 typische Funktionen, die über den intensiven Gebrauch von JavaScript realisiert werden. KiWi ermöglicht es ebenfalls, zu textuellen Informationen Metadaten zu hinterlegen. Im Unterschied zu SMW handelt es sich dabei um RDF-Metadaten. Eine weitere Eigenschaft von KiWi ist, dass Multimediadaten wie z. B. Bilder samt (Exif-)Metadaten dargestellt werden können. KiWi ist ebenfalls unter einer Open-Source Lizenz verfügbar.

Metaweb / Freebase

Metaweb ist eine proprietäre Software des Unternehmens Metaweb Technologies[38]. Sie dient als Grundlage für den Online-Dienst Freebase[39]. Freebase ist eine sehr große Wissensbasis, die eine Vielzahl strukturierter Informationen enthält. Obwohl die Software hinter Freebase nicht frei verfügbar ist, sind die Daten für jeden frei zugänglich. Freebase integriert u. a. Daten von Wikipedia und MusicBrainz[40]. Gegenwärtig existiert für Freebase ein RDF-Dienst, der die Freebase-Inhalte als Linked Data zur Verfügung stellt.

UfoWiki

Bei UfoWiki, das in [Passant and Laublet, 2008a] präsentiert wird, handelt es sich nicht um ein einzelnes semantisches Wiki, sondern vielmehr um einen Wiki-Server, der mehrere Wiki-Instanzen beherbergen kann. Die Daten der einzelnen Wiki-Instanzen werden komplett in einem zentralen RDF-Backend gehalten, wodurch es u. a. möglich wird, dass verschiedene Wiki-Instanzen Daten teilen können. Daneben können externe RDF-Datenquellen über Linked Data angesprochen werden und die entprechenden Daten in die lokale Datenhaltung integriert werden. Für die Modellierung der Wiki-Daten wird die SIOC-Ontologie [Breslin

[37] http://www.kiwi-project.eu/
[38] http://www.metaweb.com/
[39] http://www.freebase.com/
[40] http://musicbrainz.org/

et al., 2005] verwendet, respektive das types-Modul, das u. a. Klassen für Wiki-Artikel definiert. Außerdem kommt ein Tag-Vokabular[41] und die Meaning Of A Tag (MOAT) Ontologie [Passant and Laublet, 2008b] zum Einsatz, letztere, um verwendeten Schlagwörtern eine Semantik zu verleihen. Durch die zentrale Datenhaltung in Kombination mit den verwendeten Ontologien und deren Erweiterungen, ist es in UfoWiki möglich, dass jedes Wiki unabhängig agiert und lediglich die eigenen Daten für Anfragen heranzieht. Andererseits ist es aber auch realisierbar, dass explizit Daten aus anderen Instanzen integriert werden.

OntoWiki

OntoWiki ist ein Werkzeug, das ein agiles und verteiltes Wissensmanagement ermöglicht [Auer et al., 2006]. Es folgt der Wiki-Philosophie und bietet deshalb viele Funktionen, die Wikis erfolgreich gemacht haben, u. a. eine Versionsverwaltung und die Möglichkeit zur Diskussion. Im Gegensatz zu den meisten Wiki-Applikationen stellt OntoWiki jedoch nicht das Konzept der Wiki-Seite in den Fokus, sondern Ressourcen, über welche Aussagen getätigt werden können. Zu diesem Zweck basiert OntoWiki vollständig auf dem RDF-Datenmodell, d. h. sämtliche Daten und Metadaten werden in RDF dargestellt. Aufgrund dieser konzeptionellen Unterschiede wird OntoWiki auch als ein Daten-Wiki bezeichnet.

OntoWiki ist eine webbasierte Applikation, dessen serverseitige Funktionalität vollständig in PHP realisiert wurde. Die Wahl fiel u. a. deshalb auf PHP, um eine möglichst einfache Installation auf den gängigen Serversystemen zu ermöglichen, die von vielen Dienstleistern bereitgestellt werden. Seit Version 0.9 der Applikation, die eine vollständig überarbeitete Code-Basis besitzt, basiert OntoWiki auf der Erfurt API[42], einer Sammlung von Programmierschnittstellen für semantische Web-Anwendungen. Die Funktionalität von Erfurt war bis zur Version 0.9 fest mit OntoWiki verwoben und nutzte intern in weiten Teilen RAP[43], eine ebenfalls PHP-basierte API. Diese Funktionen wurden nun aus dem OntoWiki-Kern extrahiert und in Erfurt integriert. Erfurt verwendet intern das Zend-Framework[44] und bietet u. a. folgende Funktionen:

- Abstraktion von verschiedenen Systemen zur Datenhaltung (MySQL, Virtuoso, etc.),
- backendunabhängiges Hinzufügen/Löschen von Statements,

[41] http://www.holygoat.co.uk/projects/tags/
[42] http://aksw.org/Projects/Erfurt
[43] http://sourceforge.net/projects/rdfapi-php/
[44] http://framework.zend.com/

- Versionierung von Ressourcen,
- eine RDF-basierte Authentifizierung,
- eine RDF-basierte Zugriffskontrolle auf Graph- und Aktionsebene,
- ein Caching-System, um die Ausführung komplizierter SPARQL-Anfragen zu beschleunigen,
- eine Plugin-Architektur, sowie
- Import-/Export-Funktionalität in verschiedenen Formaten.

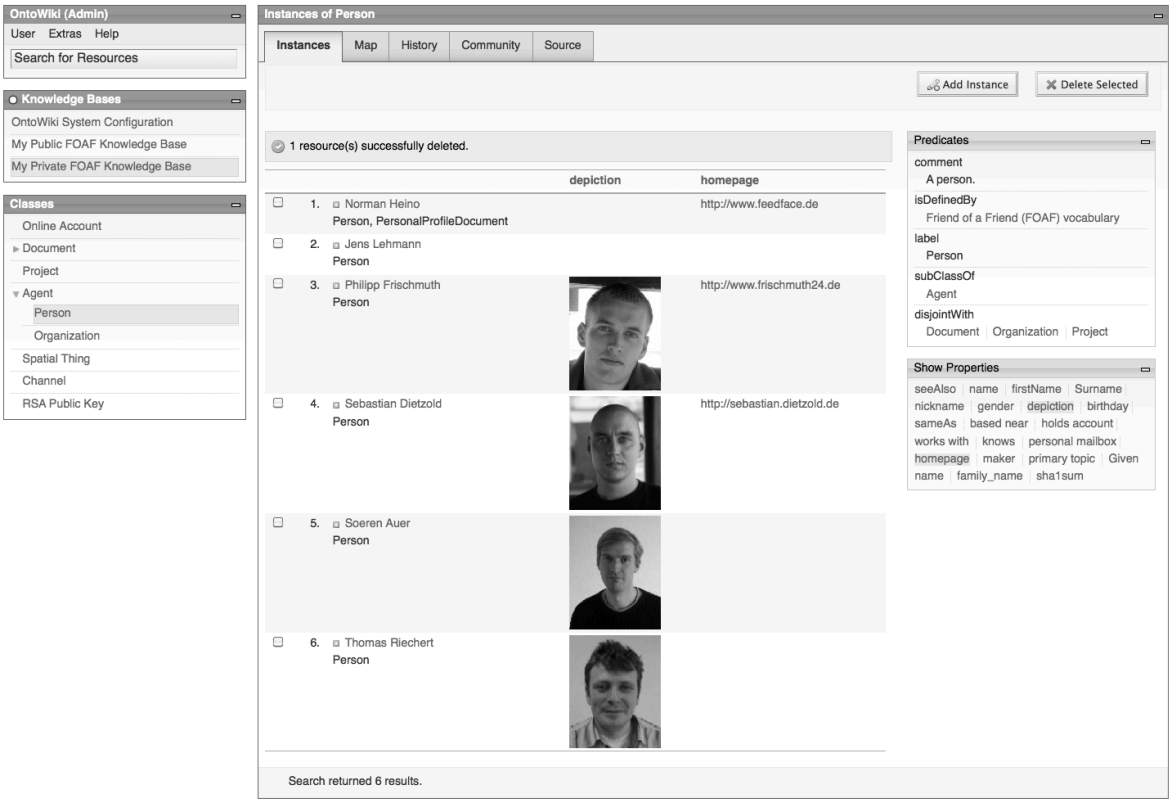

Abbildung 4.1.: OntoWiki in der Listenansicht, mit Instanzen der Klasse `foaf:Person` und den zusätzlich eingeblendeten Eigenschaften `foaf:depiction` und `foaf:homepage`

Die Entwicklung von Erfurt ist sehr eng mit der Entwicklung von OntoWiki verbunden. Es ist jedoch geplant, Erfurt zukünftig auch getrennt zu veröffentlichen, um anderen semantischen Projekten eine Nutzung zu ermöglichen.

OntoWiki besteht seit Version 0.9 aus einem sehr schlanken Kern, der auf verschiedene Weise erweitert werden kann. Viele der in OntoWiki standardmäßig verfügbaren Funktionen sind selbst als Erweiterung realisiert. OntoWiki wird deshalb auch als OntoWiki Application Framework [Heino et al., 2009] bezeichnet und kann leicht an domänenspezifische Problemstellungen angepasst werden. In der aktuellen Version bietet OntoWiki u. a. die folgenden Funktionen:

- Erstellen/Importieren neuer Wissensbasen,
- Bearbeiten von Ressourcen über RDFa basierte Widgets und Turtle-Syntax,
- tabellarische Darstellung aller vorhandenen Statements zu einer gewählten Ressource,
- Browsen von Ressourcen über Wiki-Links,
- Darstellung von Bildern in der Detail- und Listenansicht,
- Darstellung von Ressourcen auf einer Karte, falls Geokoordinaten verfügbar sind (transitiv),
- eine Versionsverwaltung,
- eine Kommentarfunktion,
- Darstellung ähnlicher Ressourcen,
- Navigation über Klassenhierarchie,
- Anzeigen aller Instanzen zu einer Klasse,
- optionale Darstellung vorhandener Eigenschaften in der Listenansicht, sowie
- Export von Ressourcen und Wissensbasen in verschiedenen Formaten.

Abbildung 4.1 stellt OntoWiki in der Listenansicht dar und visualisiert u. a. die Funktion, zusätzliche Eigenschaften in die Liste zu integrieren.

Die genannten Eigenschaften und insbesondere die Erweiterbarkeit von OntoWiki lassen es im Vergleich zu anderen Wiki-Applikationen als einen guten Kandidaten erscheinen, um die in Kapitel 3 definieren Anforderungen prototypisch zu implementieren und zu evaluieren. OntoWiki ist unter einer Open Source Lizenz verfügbar und wird u. a. an der Universität Leipzig entwickelt. In den folgenden Kapiteln wird OntoWiki als Basis für sämtliche Betrachtungen dienen.

5. Spezifikation

Im folgenden Kapitel soll das zu entwickelnde System anhand der Anforderungen für eine Implementierung in OntoWiki spezifiziert werden. Zu diesem Zweck werden zunächst die für eine Realisierung der zuvor definierten Anforderungen relevanten Bestandteile von OntoWiki untersucht. Darauf aufbauend erfolgt eine detailierte Betrachtung der einzelnen Funktionen des Systems.

5.1. OntoWiki-Architektur

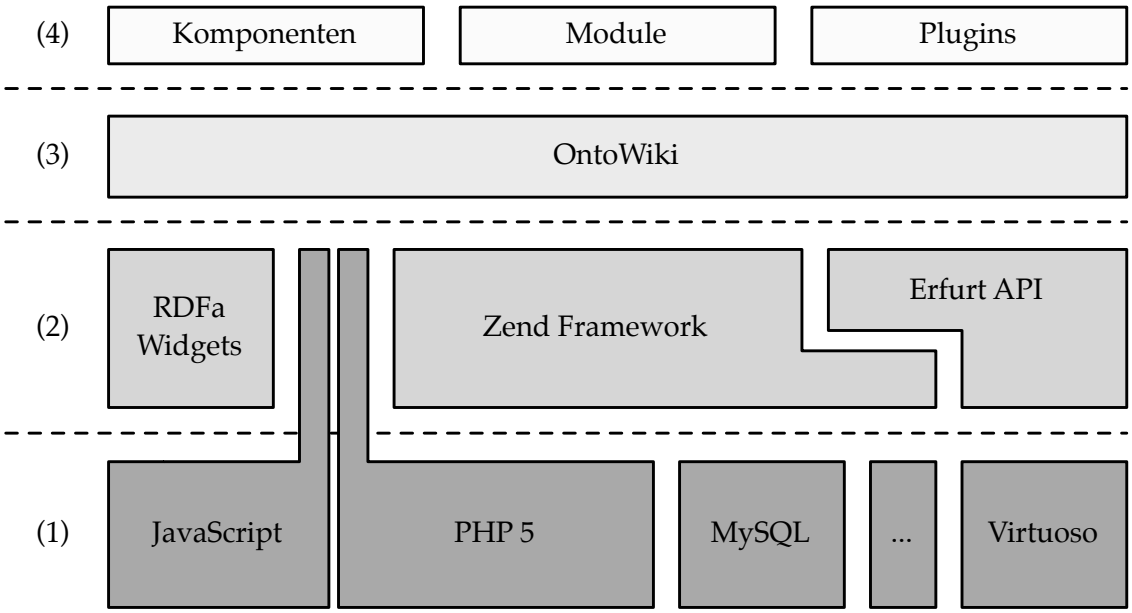

Abbildung 5.1.: Darstellung der OntoWiki-Architektur in Ebenen – (1) Basistechnologien inkl. Datenhaltung, (2) Bibliotheken, (3) OntoWiki-Kernfunktionalität, (4) Erweiterungen

Um zu evaluieren, welche Möglichkeiten existieren, die zu entwickelnden Funktionen in OntoWiki zu integrieren, wird in einem ersten Schritt die Architektur des Systems analysiert. Wie der Name OntoWiki Application Framework bereits vermuten lässt, wurde die Applikation so konstruiert, dass eine Anpassung an unterschiedliche Problemstellungen mit wenig Aufwand möglich ist. Abbildung 5.1 untergliedert die Architektur von OntoWiki in Ebenen und gibt damit einen groben Überblick über den Aufbau der Anwendung. Die für die Spezifikation des zu entwickelnden Systems relevanten Ebenen, werden in den folgenden Abschnitten vorgestellt und die vorhandenen Möglichkeiten zur Erweiterung kritisch betrachtet.

5.1.1. Erfurt API und Zend Framework

OntoWiki basiert sowohl auf der Erfurt API, als auch auf dem Zend Framework. Das Zend Framework, im Folgenden Zend genannt, ist eine auf PHP basierende, objektorientierte Bibliothek, die eine Vielzahl an Funktionalität bereitstellt, um die Entwicklung von Web-Anwendungen zu erleichtern und zu beschleunigen. OntoWiki basiert auf dem Model-View-Controller (MVC) Entwurfsmuster und setzt u. a. zu diesem Zweck Zend ein. Bei einer MVC-Architektur wird die Geschäftslogik strikt von der Präsentationslogik getrennt. Eine spezielle Steuerungslogik verbindet beide Welten. Dadurch entsteht eine sehr saubere Architektur und Zend unterstützt diesen Architekturstil.

Die Erfurt API, im Folgenden Erfurt genannt, wurde mit dem Ziel entwickelt, die Realisierung von semantischen Web-Applikationen zu ermöglichen. Um dieses Ziel zu erreichen, nutzt und erweitert Erfurt in Teilen die Zend-Funktionalität, zusätzlich stellt es aber auch neue Funktionen zur Verfügung. Abbildung 5.2 stellt die Beziehungen zwischen Erfurt und Zend dar. Eine der zentralen Funktionen von Erfurt, ist die Abstraktion verschiedener Systeme zur Datenhaltung. Zu diesem Zweck kommt das Adapter-Entwurfsmuster zum Einsatz. Für jeden unterstützten Triple-Store existiert eine Adapter-Klasse, die backendspezifische Eigenschaften kapselt. Die Applikation greift zu jeder Zeit über eine einheitliche Schnittstelle, die

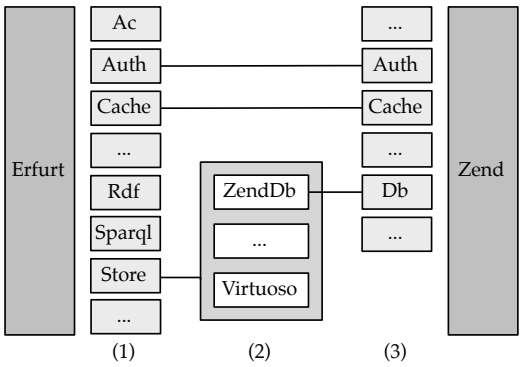

Abbildung 5.2.: Beziehungen zwischen Erfurt und Zend – (1) Erfurt-Pakete, (2) Adapter für Systeme zur Datenhaltung, (3) Zend-Pakete

`Erfurt_Store`-Klasse auf die Daten zu. Zur Zeit existieren ein Adapter für Virtuoso und einer, der die Speicherung von RDF-Daten in einer relationalen Datenbank ermöglicht. Um wiederum verschiedene Datenbanksysteme zu abstrahieren, kommt das `Zend_Db`-Paket zum Einsatz.

Des Weiteren enthält Erfurt ein Paket, das eine Benutzer-Authentifizierung ermöglicht. Hierbei wird ebenfalls auf Zend-Funktionalität zurückgegriffen, wobei wiederum ein spezieller Adapter realisiert wurde, um die für eine Authentifizierung relevanten Informationen in einem RDF-Graphen zu speichern.

Eine Möglichkeit der Erweiterung von Erfurt stellt also die Umsetzung von spezifischen Adapter-Klassen dar, um

- zusätzliche Backend-Systeme einzubinden bzw.
- das System zur Benutzer-Authentifizierung zu erweitern.

Beide Fälle sind für das zu implementierende System von Relevanz. Zusätzlich verfügt Erfurt über ein Plugin- und Ereignis-System, d. h. es können Erweiterungen realisiert werden, die nicht direkt in die Code-Basis von Erfurt integriert werden müssen, sondern unabhängig implementiert und verteilt werden können. Erfurt erzeugt an dafür vorgesehenen Stellen Ereignisse, für die sich Plugins registrieren können. Ist ein Plugin aktiviert und tritt ein Ereignis ein, für das dieses Plugin registriert wurde, so wird der Plugin-Code ausgeführt.

5.1.2. RDFa Widgets

Die RDFa Widget Bibliothek wird in OntoWiki eingesetzt, um dem Nutzer eine Möglichkeit zur komfortablen Bearbeitung von RDF-Daten zu bieten. Die RDFa Widgets basieren komplett auf JavaScript und sind nicht auf die ausschließliche Nutzung mit OntoWiki beschränkt. Stattdessen setzen sie lediglich eine mit RDFa annotierte HTML-Seite voraus und zusätzlich einige Metadaten, um den Ursprung und das Ziel der Daten zu bestimmen.

Die Bibliothek lässt sich durch die Implementierung neuer Widgets erweitern. Diesbezüglich muss eine JavaScript-Datei bereitgestellt werden, die definierte Methoden realisiert. Ein Widget kann sich bei der Bibliothek für bestimmte Ereignisse registrieren. Beim Eintreten des Ereignisses wird das Widget aktiviert und der Code ausgeführt. Solche Ereignisse können sich z. B. auf bestimmte Datentypen oder Relationen beziehen, aber auch darauf, ob es sich bei dem zu bearbeitenden Wert um ein Literal oder eine Ressource handelt. Letzteres begründet

die Relevanz der Widget Bibliothek für die Spezifikation, da aus den Anforderungen folgt, dass der Nutzer bei der Auswahl von Ressourcen stets zu unterstützen ist.

5.1.3. Erweiterung des OntoWiki-Kerns

Der Kern von OntoWiki wurde bewusst klein gehalten und große Teile der standardmäßig verfügbaren Funktionalität sind über Erweiterungen realisiert, um eine einfache Anpassung an unterschiedliche Anforderungen zu ermöglichen. Das zu entwickelnde System sollte überall dort, wo es möglich erscheint, ebenfalls mit Hilfe von Erweiterungen umgesetzt werden. Um die Tauglichkeit diesbezüglich zu überprüfen, werden die einzelnen Erweiterungsarten im Folgenden beschrieben.

Komponenten

Komponenten in OntoWiki stellen einen eigenen Controller und evtl. eigene Views zur Verfügung. Über Komponenten lassen sich z. B. neue Tabs in die Applikation integrieren. Ein Tab steht im OntoWiki-Kontext für eine dedizierte Sicht auf eine oder mehrere Ressourcen. Beispiele für solche Sichten sind die Kartenansicht und die Kalenderansicht. Komponenten sind aber nicht auf Tabs für die Sicht auf Ressourcen beschränkt. Sie können beliebige Views erzeugen, die einer bestimmten Aktion zugeordnet werden und bei einem Aufruf den Hauptinhalt der OntoWiki-Instanz generieren. Ein Beispiel für einen solchen allgemeingültigen View ist das Registrierungsformular, mit dem sich neue Nutzer am System registrieren können.

Jede Komponente erhält einen Namensraum, in dem sie beliebige Aktionen bereitstellen kann, solange sie im eigenen Namensraum eindeutig sind. Der Namensraum wird durch die Benennung der Komponente bestimmt. Zusätzlich können Komponenten JavaScript-Dateien und Cascading Style Sheets (CSS) einbetten und Übersetzungen hinzufügen. Über eine Hilfsklasse, deren Code auch ausgeführt wird, wenn die Komponente aktiviert ist, jedoch keine der Komponente zugeordnete Aktion ausgeführt wird, können Komponenten bspw. Menü-Einträge erzeugen.

Zusätzlich sei zu erwähnen, dass die Bereitstellung eines Views für eine Aktion optional ist, d. h. eine Komponente ist dazu nicht verpflichtet. Dies macht es möglich, Aktionen als Dienste zu realisieren, die dann z. B. über asynchrone Anfragen genutzt werden können.

Module

Module sind kleine Fenster, die an verschiedenen Stellen in OntoWiki integriert werden können. Zu diesem Zweck registrieren sich Module für einen oder mehrere Kontexte. So ist es z. B. möglich, Fenster der Seitenleiste hinzuzufügen, die u. a. auch die Klassenhierarchie beinhaltet. Des Weiteren können Module aber auch in das Hauptfenster eingebettet werden, in Fällen in denen ein Modul mit einer oder mehreren Ressourcen arbeitet. Module können ebenfalls JavaScript und CSS integrieren und stellen ihren Inhalt über festgelegte Methoden zur Verfügung. Für die Generierung des Fensterinhalts sind sie selbst verantwortlich, wobei sie dabei auf ein Template zurückgreifen können. Ein Template ist eine HTML-Datei, die mit variablen Platzhaltern versehen wird, die vom Modul mit Werten belegt werden.

Plugins

Das Plugin-System von OntoWiki basiert auf dem Plugin-System, das von Erfurt bereitgestellt wird. Es wird das identische Ereignis-System verwendet. Alle für Erfurt entwickelten Plugins können dadurch auch in OntoWiki eingesetzt werden. OntoWiki erzeugt zusätzlich zu den von Erfurt erzeugten Ereignissen, eine Vielzahl weiterer Ereignisse, für die sich Plugins registrieren können. Ein Plugin stellt keine eigenen Sichten bereit, kann aber je nach Ereignis Inhalte und somit das Erscheinungsbild von OntoWiki modifizieren.

Controller-Plugins

Eine weitere Möglichkeit zur Erweiterung von OntoWiki stellen Controller-Plugins dar, wenngleich diese nicht zu den offiziell unterstützten Erweiterungsarten für OntoWiki zählen. Die Möglichkeit, Controller-Plugins zu entwickeln, entstammt dem Zend Framework und wurde von diesem unverändert übernommen. Während eines Controller-Lebenszyklusses werden zu definierten Zeitpunkten, bestimmte Methoden von registrierten Plugins aufgerufen und deren Code ausgeführt. OntoWiki stellt ein solches Plugin z. B. für eine HTTP Basis-Authentifizierung bereit.

Aufbauend auf den Betrachtungen zur Architektur von OntoWiki, sollen in den folgenden Abschnitten sämtliche Funktionen des Systems spezifiziert werden.

5.2. Suche und Auswahl von Ressourcen

Diese Funktion deckt die Anforderung ab, dass das zu entwickelnde System eine Suche nach relevanten Ressourcen ermöglicht und den Nutzer bei der Auswahl der gewünschten Ressource unterstützt.

5.2.1. Suchfunktion

Für die Suche nach Ressourcen ist gefordert, dass sowohl lokale, als auch externe Informationen mit einbezogen werden. Für die externe Suche nach Ressourcen muss zusätzlich die Möglichkeit bestehen, das System um weitere Suchdienste zu erweitern. Algorithmus 2 spezifiziert den generellen Aufbau der Suchfunktion, wobei die einzelnen Bestandteile im Folgenden näher erläutert werden.

```
1  SELECT DISTINCT ?uri ?o
2  WHERE {
3    { ?uri ?p ?o .
4      FILTER ((isLiteral(?o) && regex(str(?o), "leipzig", "i")) ||
5        regex(str(?uri),
6          "(http(s)?://.*(/|#)leipzig).*$|!(http(s)?://).*leipzig.*$", "i")
7      )
8    }
9    UNION
10   { ?s ?p ?uri .
11     FILTER (isIRI(?uri) &&
12       regex(str(?uri),
13         "(http(s)?://.*(/|#)leipzig).*$|!(http(s)?://).*leipzig.*$", "i")
14     )
15   }
16 }
17 ORDER BY ?uri
18 LIMIT 10
```

Listing 5.1: SPARQL-Anfrage zur Suche nach dem Begriff `leipzig`

Der Suchalgorithmus arbeitet in zwei Modi, einem für die normale Suche nach Ressourcen und einem für die Suche nach Eigenschaften und Relationen. Die zwei Modi unterscheiden sich durch die Nutzung unterschiedlicher SPARQL-Anfragen. Die Erweiterungen werden

Algorithmus 2 : Suche nach relevanten Ressourcen in verschiedenen Quellen und Vereinigung der Teilergebnisse

Input : Stichworte T, mit $t_1, \ldots, t_n \in$ T und $n = |\text{T}|$
Output : geordnete Liste U, mit den gefundenen URIs
Data : selektierte Graph-URI g
Data : SPARQL-Anfrage s
Data : Such-Plugins P, mit $p_1, \ldots, p_m \in$ P und $m = |\text{P}|$

```
1  foreach t ∈ T do                    /* Prüfe alle Terme, ob es sich um URI handelt. */
2      if IsTermUri(t) = TRUE then
3          U ← ∅
4          return U

   // SPARQL-Anfrage an die lokale Datenbasis stellen.
5  FullResult ← ∅
6  SparqlResult ← SparqlQuery(s, g)
7  FullResult ← SparqlResult

8  foreach p ∈ P do                    /* Führe alle aktiven Erweiterungen aus. */
9      FullResult ← FullResult ∪ ExecutePlugin(p)

10 foreach t ∈ T do                    /* Prüfe alle Terme, ob es sich um QNames handelt. */
11     ExpandResult ← ∅
12     if IsQName(t) = TRUE then
13         ExpandResult ← ExpandResult ∪ Expand(t)

14 FullResult ← FullResult ∪ ExpandResult

   // Füge dem Ergebnis eine mit T generierte URI hinzu.
15 FullResult ← FullResult ∪ GenerateUri(T)

16 U ← OrderByRelevance(FullResult)
17 return U
```

über den verwendeten Modus informiert und können ihr Verhalten dementsprechend anpassen. In Listing 5.1 und Listing 5.2 sind die SPARQL-Anfragen dargestellt, die für die lokale Suche in den verschiedenen Modi genutzt werden. Zusätzlich sei erwähnt, dass sich die durch den Suchalgorithmus erzeugte Ergebnismenge begrenzen lässt. Es sei außerdem bemerkt, dass bei mehreren Suchbegriffen eine UND-Verknüpfung der Begriffe stattfindet und die Anfrage dementsprechend erweitert wird. Aus Gründen der Übersichtlichkeit wurde an dieser Stelle auf die Darstellung verzichtet.

Die Anfrage aus Listing 5.1 liefert alle Ressourcen (`?uri`), die

- als Subjekt in einem Statement auftreten, dessen Objekt (`?o`) ein Literal ist und zusätzlich den regulären Ausdruck erfüllt,
- als Subjekt in einem Statement auftreten, deren URI den regulären Ausdruck erfüllt oder
- die als Objekt in einem Statement auftreten, deren URI ebenfalls den regulären Ausdruck erfüllt.

Im ersten Teil der Anfrage (Zeile 3) wird zusätzlich das jeweilige Objekt selektiert, um ein späteres Ranking der Ergebnisse zu realisieren. Der reguläre Ausdruck wird erfüllt, wenn ein Stichwort in dem jeweiligen Term an beliebiger Stelle vorkommt. Bei URIs wird zusätzlich überprüft, ob es sich um HTTP-URIs handelt. In solch einem Fall, wird lediglich der lokale Teil der URI betrachtet. Dieses Vorgehen wurde gewählt, da Versuche gezeigt haben, dass verwendete Stichworte gelegentlich im Namensraum einer URI auftreten. Da die lokale Datenbasis oft sehr viele URIs enthält, die in einem lokalen Namensraum angesiedelt sind, ergeben sich eine Fülle an Treffern, die in vielen Fällen für den Nutzer nicht relevant sind.

Die Anfrage aus Listing 5.2 wird eingesetzt, wenn nach Eigenschaften und Relationen gesucht wird. Diese Anfrage liefert alle Ressourcen, die zusätzlich

- als `rdf:Property`, `owl:DatatypeProperty` oder `owl:ObjectProperty` deklariert wurden und die
  - als Subjekt in einem Statement auftreten, dessen Objekt (`?o`) ein Literal ist und zusätzlich den regulären Ausdruck erfüllt oder
  - als Subjekt in einem Statement auftreten, deren URI den regulären Ausdruck erfüllt und solche
- die als Prädikat in einem Statement auftreten, deren URI den regulären Ausdruck erfüllt.

```
SELECT DISTINCT ?uri ?o
WHERE {
  { ?uri ?p ?o . ?uri rdf:type ?o2 .
    FILTER (
      sameTerm(?o2, rdfs:Property) || sameTerm(?o2, owl:DatatypeProperty) ||
      sameTerm(?o2, owl:ObjectProperty)
    )
    FILTER (regex(str(?uri), "mbox", "i") ||
      (isLiteral(?o) && regex(str(?o), "mbox", "i"))
    )
  }
  UNION
  { ?s ?uri ?o .
    FILTER (regex(str(?uri), "mbox", "i"))
  }
}
ORDER BY ?uri
LIMIT 10
```

Listing 5.2: SPARQL-Anfrage zur Suche nach dem Begriff `mbox` im Property-Modus

In bestimmten Teilen der Anfrage wird wieder das Objekt (`?o`) des Statements selektiert, um eine spätere Gewichtung der Ergebnisse zu ermöglichen. In dieser Anfrage wurde auf die Sonderbehandlung von HTTP-URIs verzichtet, da die o. g. negativen Auswirkungen hier nicht auftreten.

Eine weitere Anforderung an die Suche ist, dass der Nutzer der Suchfunktion stets mit einer aussagekräftigen textlichen Repräsentation der Ressourcen-URI versorgt wird. Zu diesem Zweck kann auf OntoWiki-Funktionalität zurückgegriffen werden. OntoWiki bietet die Möglichkeit, über konfigurierbare Eigenschaften, einen Titel für Ressourcen abzufragen. Intern wird dafür eine komplexe SPARQL-Anfrage erzeugt, mit der nach Möglichkeit die Titel für mehrere Ressourcen parallel abgerufen werden. Außerdem wird gefordert, dass die Herkunft der Suchergebnisse für den Nutzer stets ersichtlich ist. Für das Ergebnis einer Suchanfrage ergibt sich somit, dass jedes Teilergebnis statt aus einer bloßen URI aus einem Tupel der Form

$$e := (u, t, q), \text{ mit } u \in \mathrm{U}, t = T(u) \text{ und } q = Q(u)$$

besteht, wobei U die Menge der gefundenen URIs bezeichnet und T bzw. Q Funktionen sind, die URIs auf die Menge der verfügbaren Titel bzw. aktiven Suchdienste abbilden.

5.2.2. Ranking der lokalen Suchergebnisse

Da die SPARQL-Spezifikation eine Volltextsuche in ihrer aktuellen Version nicht vorsieht, müssen die Ergebnisse der SPARQL-Anfrage nachträglich nach ihrer Relevanz für den Nutzer sortiert werden. Zu diesem Zweck wurde eine einfache Metrik entwickelt, die in Algorithmus 3 spezifiziert wird.

Algorithmus 3 : Einfache Metrik zur Berechnung der Relevanz eines Ergebnisses in Bezug auf die Anfrage

Input : Stichworte T, mit $t_1, \ldots, t_n \in \mathrm{T}$ und $n = |\mathrm{T}|$
Input : URI u
Input : Literal l oder *NULL*
Output : Gewicht g, mit $g \in \mathrm{R}, g \geq 0, g \leq 1$

```
1  g ← 0.0
2  Local ← GetUriLocalPart(u)
3  if HasUriLocalPart(u) = TRUE then
4      foreach t ∈ T do
5          if InString(t, Local) then
6              g ← g + 0.4 ÷ |T|
7  if HasValue(l) then
8      foreach t ∈ T do
9          if InString(t, l) then
10             g ← g + 0.5 ÷ |T|
11     s1 ← ConcatString(T)
12     l1 ← StringLength(s1)
13     s2 ← RemoveWhitespace(l)
14     l2 ← StringLength(s2)
15     if s1 = s2 then
16         g ← g + 0.1
17 return g
```

Es wird davon ausgegangen, dass ein Treffer in einem Literal ein etwas höheres Gewicht

hat, als ein Treffer im lokalen Teil einer URI. Deshalb wird das maximal erreichbare Gewicht für Literale um 0.1 größer gewählt. Zusätzlich führen Literale, die sämtliche Suchbegriffe enthalten, zu einem Zusatzgewicht von 0.1. Es ergibt sich damit ein Maximalgewicht von 1.0 und ein Minimalgewicht von 0.0. Das Maximalgewicht wird erreicht, wenn sämtliche Suchbegriffe sowohl im lokalen Teil der URI, als auch im Literal auftreten. Zusätzlich ist sichergestellt, dass ein Auftreten der Suchbegriffe in einem Literal zu einer höheren Wertung führt, als das bloße Auftreten in einer URI.

5.2.3. Erweiterung der Suche

Die Erweiterbarkeit der Suchfunktion muss sichergestellt sein. Aus diesem Grund wird eine Möglichkeit geschaffen, die Suche über leichtgewichtige Plugins zu erweitern. Entsprechende Plugins müssen in die Lage versetzt werden, an einer bestimmten Stelle im Suchablauf eigene Suchergebnisse zu erzeugen und diese der Suchfunktion zu übergeben. Das Gesamtergebnis der Suche sollte durch die Suchfunktion und nicht durch die jeweiligen Plugins zusammengesetzt werden.

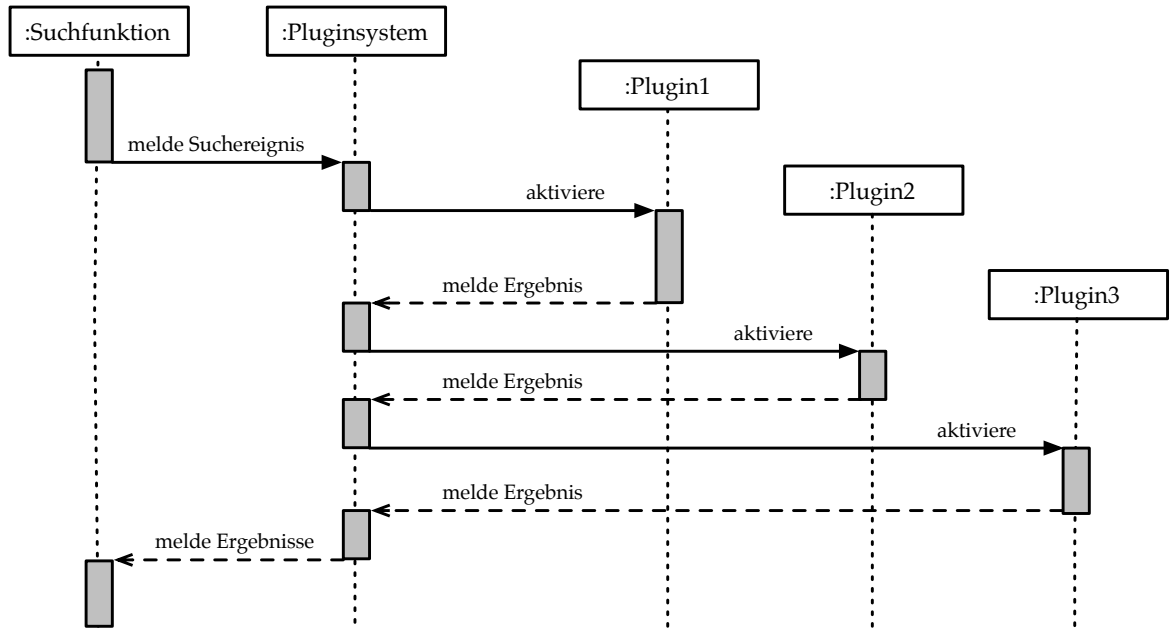

Abbildung 5.3.: Ablauf einer Suche mit Plugins

Die Anforderungen an die Erweiterbarkeit sind mit den in OntoWiki verfügbaren Erwei-

terungsarten vollständig erfüllbar. Aus diesem Grund wird für die Realisierung der Such-Plugins auf OntoWiki-Plugins zurückgegriffen. In die Suchfunktion wird ein neues Ereignis integriert, für das sich die Plugins registrieren können. Sämtliche Plugins müssen lediglich auf dieses eine Ereignis reagieren und die jeweilige Suchoperation ausführen. Den Plugins sollten dazu die gleichen Informationen bereitstehen, wie der primären Suchfunktion.

Diese beinhalten

- die Suchbegriffe,
- den Suchmodus und
- Informationen darüber, ob die Ergebnismenge begrenzt werden soll.

Abbildung 5.3 stellt den Ablauf einer Suche mit Plugins in einem Sequenzdiagramm dar.

Sindice-Plugin

Um die Erweiterbarkeit der Suchfunktion zu demonstrieren, soll des Weiteren ein Plugin für den Sindice-Suchdienst implementiert werden. Zu diesem Zweck wird ein OntoWiki-Plugin realisiert, welches das entsprechende Ereignis behandelt. Bei einer Aktivierung des Plugins wird eine HTTP-Anfrage an die Sindice REST-API gestellt. Die Anfrage hat die Form

```
http://api.sindice.com/v2/search?qt=term&page=1&q=,
```

wobei der `q`-Parameter durch die jeweiligen Stichworte ergänzt wird, welche durch ein + miteinander verbunden werden.

Das Ergebnis der Anfrage ist ein JSON-kodiertes Array, das u. a. die Liste der Treffer enthält. Jeder Eintrag enthält eine URI und einen Titel. Diese beiden Informationen werden genutzt, um die Ergebnistupel zu generieren. Da die Ergebnisse einer Sindice-Anfrage bereits gewichtet sind, wird die Reihenfolge der Ergebnisse beibehalten.

5.2.4. Vereinigung der Suchresultate

Die Integration der jeweiligen Ergebnismengen zu einem Gesamtergebnis, erfolgt nach einem einfachen Schema, das die erhöhte Relevanz lokaler Suchergebnisse berücksichtigt:

1. Die gewichteten, lokalen Ergebnisse werden dem Gesamtergebnis hinzugefügt.

2. Für jedes aktive Plugin werden die Teilergebnisse dem Gesamtergebnis hinzugefügt. Dabei wird die Sortierung der jeweiligen Plugin-Ergebnisse berücksichtigt. Die Ergebnisse der verschiedenen Plugins werden in der Reihenfolge integriert, in der sie ausgeführt wurden.
3. Die evtl. vorhandenen expandierten URIs werden dem Gesamtergebnis hinzugefügt. Expandierte URIs werden erzeugt, wenn in den Stichworten qualifizierte Namen (QNames) enthalten sind.
4. Die generierte URI wird dem Gesamtergebnis hinzugefügt. Die generierte URI wird aus dem lokalen Namensraum in Verbindung mit den gegebenen Stichworten gebildet, um in letzter Instanz die Möglichkeit zu bieten, neue URIs anzulegen.

Sollte eine Begrenzung der Anzahl der Ergebnisse gewünscht sein, wird das Hinzufügen weiterer Ergebnisse abgebrochen, sobald die entsprechende Anzahl erreicht ist.

5.2.5. Integration in die OntoWiki-Oberfläche

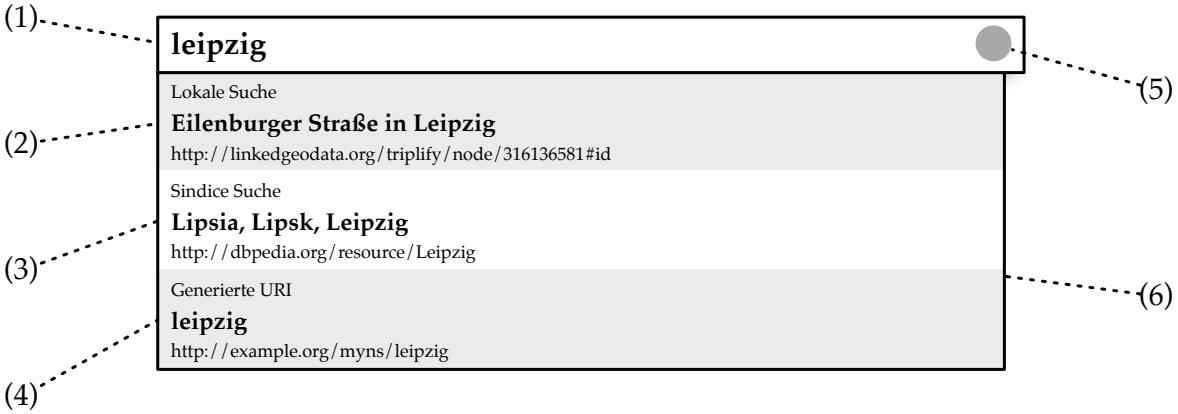

Abbildung 5.4.: Entwurf der Benutzungsoberfläche für die Suche – (1) Eingabefeld, (2) Ergebnis der lokalen Suche, (3) Ergebnis der Sindice-Suche, (4) generierte URI, (5) Platzhalter für eine Animation, um den Nutzer auf eine Suchaktivität hinzuweisen, (6) dynamisch erzeugte Liste mit Ergebnissen

Die Suchfunktionalität soll an zwei Stellen in OntoWiki integriert werden. Zum Einen ist es bei der Bearbeitung von Statements wünschenswert, nach existierenden Ressourcen zu suchen. Zum Anderen sollte es die Möglichkeit geben, beliebige Ressourcen in OntoWiki

zu betrachten. Für die Auswahl der gewünschten Ressource, ist ebenfalls die Integration der Suchfunktion vorgesehen. Abbildung 5.4 zeigt einen Entwurf der Benutzungsoberfläche für die Suche. Im Folgenden werden nun die konkreten Stellen betrachtet, an denen die Integration erfolgt.

Widget

Da es sich bei OntoWiki um eine wikiartige Applikation handelt, ist es besonders wichtig, dass ein einfaches Editieren der Daten ermöglicht wird. Diesbezüglich setzt OntoWiki eine JavaScript basierte Widget Bibliothek ein. Obwohl diese Bibliothek sich zur Zeit noch in der Entwicklung befindet, existieren bspw. schon Widgets für Literalwerte und ein spezielles Widget zur Bearbeitung von Datumsangaben. Für das Editieren von Relationen zwischen Ressourcen existiert bisher nur ein rudimentärer Ansatz, der lediglich die direkte Eingabe einer URI erlaubt. Auch für die Auswahl neuer Eigenschaften und Relationen ist bisher eine manuelle Eingabe erforderlich. Um die Anforderungen an das System dennoch zu erfüllen, soll ein Widget für Ressourcen entwickelt werden, das sowohl die Suche nach Ressourcen, als auch die Suche nach Eigenschaften bzw. Relationen ermöglicht.

Für die Suche wird der weiter oben spezifizierte Dienst genutzt und auf JavaScript-Seite integriert. Eine Besonderheit stellt die Tatsache dar, dass der Suchdienst spezifisch für OntoWiki entwickelt wurde, die Widget Bibliothek aber mit dem Ziel entsteht, in diversen RDFa-basierten Web-Applikationen nutzbar zu sein. Deshalb wäre es notwendig, die vollständige Suchfunktion in der JavaScript-Bibliothek zu duplizieren. Da sich diese Arbeit aber schwerpunktmäßig mit Daten-Wikis beschäftigt, wird darauf verzichtet. Stattdessen soll eine Abfrage in das Widget integriert werden, die prüft, ob das Widget innerhalb einer OntoWiki-Umgebung genutzt wird. Im positiven Fall wird der OntoWiki-Dienst genutzt. Für den Fall der Nutzung außerhalb von OntoWiki, wird lediglich die Sindice-Suche integriert, um dennoch eine Suche zu ermöglichen.

Das Widget nutzt die von der Bibliothek bereitgestellten Oberflächenelemente, wie Buttons und Rahmen und integriert zusätzlich ein einzelnes Eingabefeld, wie es in Abbildung 5.4 dargestellt ist.

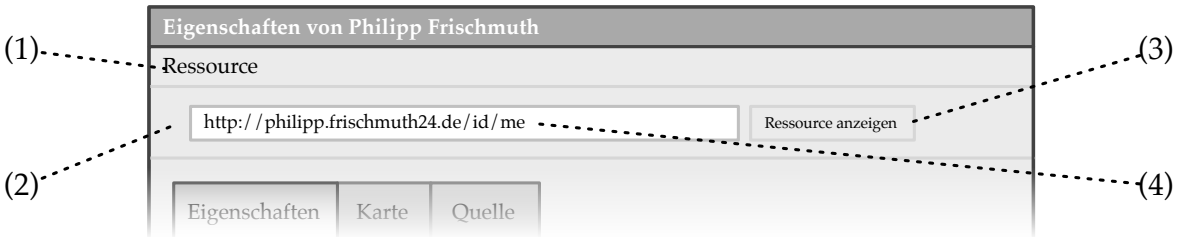

Abbildung 5.5.: Entwurf der Benutzungsoberfläche für die Adressleiste – (1) Menü mit Option zur Aktivierung/Deaktivierung der Adressleiste, (2) Eingebettete Adressleiste mit Eingabefeld und Button, (3) Button zum Öffnen der Detailansicht für die gewählte Ressource, (4) URI der aktuell betrachteten Ressource

Adressleiste

Aus den Anforderungen folgt u. a. auch, dass es möglich sein soll, das System als Browser zu verwenden. In OntoWiki ist es derzeit nur begrenzt möglich, an einer beliebigen Stelle mit dem Erkunden von Ressourcen zu beginnen. Es existieren dazu derzeit drei Möglichkeiten:

1. Zugang über einen von OntoWiki zur Verfügung gestellten Link. Diesbezüglich ist die Verlinkung der gewünschten Ressource mit anderen Ressourcen notwendig, z. B. durch die Übernahme der Rolle des Wertes in einer Relation.
2. Nutzung der in OntoWiki integrierten Volltextsuche. Diese Art der Suche ist allerdings auf lokale Daten beschränkt und das verwendete Backend muss diese Suche unterstützen.
3. Übergabe einer beliebigen URI als Parameter in der Adressleiste des Browsers. Diese Option ist für den normalen Nutzer nicht handhabbar und stellt somit keine Möglichkeit dar.

Es ergibt sich also die Notwendigkeit einer zusätzlichen Eingabemöglichkeit für Ressourcen, die von der oben spezifizierten Suchfunktion Gebrauch macht. Zu diesem Zweck soll eine Adressleiste in OntoWiki integriert werden, die vom Nutzer optional aktiviert werden kann. Es ist vorgesehen, die Adressleiste in die Detailansicht einer Ressource zu integrieren, da der direkte Aufruf einer Ressource zu dieser Ansicht führt. Nutzer sind aufgrund der Verwendung von Webbrowsern mit Adressleisten vertraut. Aus demselben Grund erwartet ein Nutzer eine Adressleiste am oberen Bildrand. Die Adressleiste soll deshalb an promi-

nenter Stelle in der Detailansicht integriert werden. Abbildung 5.5 zeigt einen Auszug der entstehenden Oberfläche.

Um dem Nutzer einen möglichst komfortablen Arbeitsablauf zu ermöglichen, wird die Adressleiste über das Ressourcen-Menü aktiviert und deaktiviert, wobei die aktuell gewählte Option für die gesamte Sitzung erhalten bleibt. Beim Aufruf der Detailansicht mit aktivierter Adressleiste, enthält diese die URI der aktuell dargestellten Ressource. Durch die Eingabe beliebiger Stichworte, wird die Suche aktiviert und das Suchergebnis dem Nutzer wie in Abbildung 5.4 dargestellt präsentiert. Bei der direkten Eingabe einer URI liefert die Suche ein leeres Resultat und irritiert den Nutzer somit nicht durch die Anzeige von Suchergebnissen. Mit Hilfe der Eingabetaste und über den entsprechenden Button, kann der Nutzer die Funktion zur Anzeige der Ressource aktivieren.

5.3. Import und Synchronisation von Statements

Das zu entwickelnde System muss, um die Anforderungen zu erfüllen, die Möglichkeit bieten, sowohl RDF-Daten, als auch Daten, die nicht auf RDF basieren, in die lokale Datenbasis zu importieren und mit der Datenquelle zu synchronisieren. Hierfür wird das System zunächst im Hinblick auf den Datenimport spezifiziert. Darauf aufbauend erfolgt eine Betrachtung der Datensynchronisation.

Ein System zum Datenimport auf Statementebene erscheint im RDF-Kontext zunächst trivial, da die Kompatibilität der Daten per se gegeben ist. Es muss lediglich geklärt werden, wie auf die Daten zugegriffen werden kann und ob die Daten in einem unterstützten Serialisierungsformat vorliegen. Ersteres ist im hier betrachteten Kontext ebenfalls relativ einfach zu beantworten, da Daten über die URI dereferenziert werden. Für das Abrufen von solchen RDF-Daten (Linked Data) würde es genügen, bspw. eine OntoWiki-Komponente zu implementieren, die einen Dienst bereitstellt, um RDF-Daten über eine URI abzurufen und zu importieren. Da die Unterstützung von reinen RDF-Daten aber nicht ausreicht, um sämtliche Anforderungen abzudecken, muss hier ein anderer Weg beschritten werden.

Zusätzlich existiert die Anforderung, dass das System erweiterbar gestaltet wird, um zu einem späteren Zeitpunkt weitere Datenquellen anbinden zu können. Um dies in einer Art und Weise zu realisieren, die es ermöglicht, auf das Erzeugen von RDF-Daten spezialisierte Erweiterungen zu entwickeln, soll ein neuer leichtgewichtiger Erweiterungstyp spezifiziert werden. Da das Erzeugen von RDF-Daten für viele semantische Web-Applikationen von

Interesse sein kann, erfolgt die Integration nicht auf OntoWiki-Ebene, sondern auf Erfurt-Ebene. Erweiterungen dieser Art werden im Folgenden als Wrapper bezeichnet, angelehnt an den englischen Begriff für Hülle.

5.3.1. Wrapper-Architektur

Die Wrapper-Architektur soll in Anlehnung an die bereits existierende Plugin-Architektur in Erfurt spezifiziert werden. Abbildung 5.6 zeigt das zu entwickelnde Erfurt-Paket mit seinen Bestandteilen und Beziehungen.

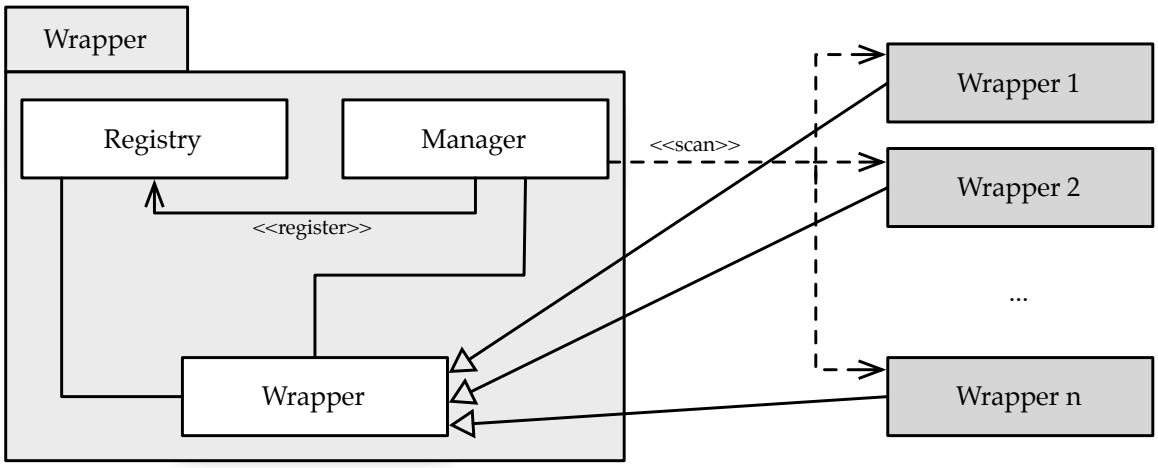

Abbildung 5.6.: Paketdiagramm der Erfurt Wrapper-Komponente

Das Importieren von Daten über Wrapper gliedert sich in einen dreistufigen Prozess, der im Folgenden erläutert wird.

1. Zunächst muss ein Wrapper für eine gegebene URI entscheiden, ob diese behandelt werden soll. Dies kann z. B. über einen regulären Ausdruck festgestellt werden, wenn ein Wrapper bestimmte URIs voraussetzt. Es ist aber auch denkbar, dass ein Wrapper diesbezüglich eine SPARQL-Anfrage stellt, um bestimmte Eigenschaften und Relationen einer Ressource zu ermitteln. Es ist angedacht, dass dieser Prozess möglichst wenig Zeit konsumiert, da diese Prüfung je nach Anwendung evtl. für viele URIs bzw. für viele Wrapper durchgeführt werden muss.
2. Im nächsten Schritt muss überprüft werden, ob für eine gegebene (behandelte) URI

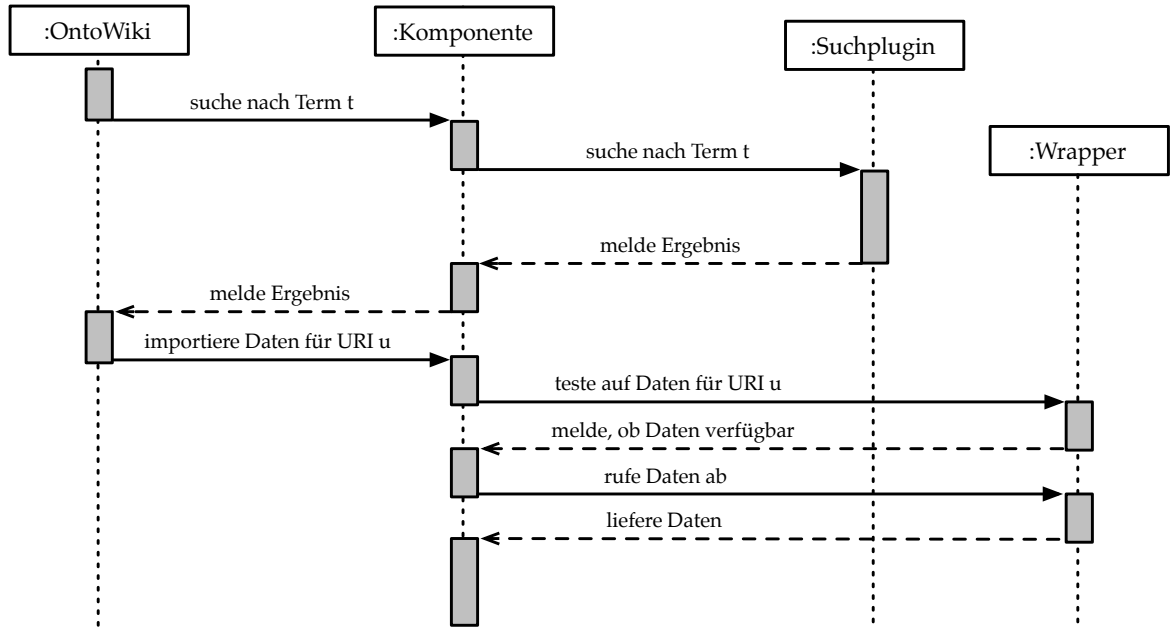

Abbildung 5.7.: Sequenzdiagramm zum Ablauf des Importvorgangs in Verbindung mit einer Suche

Daten verfügbar sind. Dies wird in vielen Fällen das Abrufen der Daten von der Datenquelle implizieren. Es ist aber möglich, dass für bestimmte Datenquellen effizientere Verfahren genutzt werden können, um die Verfügbarkeit von Daten zu determinieren. Genau wie im ersten Schritt sollte die Überprüfung mit einer Antwort der Form `true/false` abgeschlossen werden, um dem System zu signalisieren, ob Daten vorhanden sind oder nicht.

3. In einem letzten Schritt wird der Wrapper ausgeführt. Es ist vorgesehen, dass ein Wrapper am Ende dieses Schrittes eine Liste mit verfügbaren Statements an das System zurückliefert. Um das System flexibel zu halten, werden hier allerdings keine Einschränkungen vorgenommen. Stattdessen kann jede Wrapper-Realisierung entscheiden, wie sie mit generierten Daten umgeht. Demnach wäre es auch denkbar, dass ein Wrapper intern Daten der lokalen Datenbasis modifiziert. Diese Freiheit wurde jedoch nur gewährt, um eine maximale Flexibilität zu ermöglichen. Für den Import und die spätere Synchronisation wird nun davon ausgegangen, dass Wrapper die gefundenen Daten zurück liefern, ohne dabei Daten selbständig zu modifizieren.

Um Wrapper in OntoWiki nutzbar zu machen, muss zusätzlich eine Komponente realisiert werden, die Dienste bereitstellt, um einen bestimmten Wrapper auf verfügbare Daten hin zu überprüfen und Daten in ein lokales Modell zu importieren. Zu diesem Zweck soll die identische Komponente Anwendung finden, die im Zusammenhang mit der Suche spezifiziert wurde. Sie soll um die zwei genannten Dienste erweitert werden. Abbildung 5.7 visualisiert einen typischen Arbeitsablauf anhand eines Sequenzdiagramms.

Linked Data Wrapper

Um die geforderten Anforderungen in Bezug auf den Import von Linked Data zu erfüllen, wird auf Basis der spezifizierten Architektur ein Wrapper speziell für Linked Data entwickelt. Dieser Wrapper behandelt sämtliche HTTP-URIs. Um zu prüfen, ob Daten für eine URI vorhanden sind, wird versucht, diese zu dereferenzieren. Erhält der Wrapper eine gültige Antwort auf seine Anfrage, wird der Typ des erhaltenen Dokuments überprüft. Handelt es sich um ein Dokument in einem der unterstützten RDF-Serialisierungsformate, wie XML, Turtle oder JSON, wird das Dokument mit dem entsprechenden Parser verarbeitet. Enthält ein solches Dokument Verweise auf weitere Dokumente, die mit Hilfe der Relationen `rdfs:seeAlso`, `rdfs:isDefinedBy` oder `owl:sameAs` gebildet wurden, sollten diese Dokumente ebenfalls verarbeitet werden. Liegt ein HTML-Dokument vor, wird dieses auf Links zu alternativen Dokumenten überprüft, die RDF-Daten enthalten. Werden solche Links gefunden, erfolgen erneute Anfragen für diese Dokumente. Gleiches trifft zu, wenn auf eine Anfrage mit einem Verweis auf eine andere URL geantwortet wird. Dieses Verfahren wird wiederholt, bis eine gewünschte Antwort gefunden wurde oder ein definiertes Limit erreicht wurde. Stehen am Ende des Vorgangs RDF-Daten zur Verfügung, werden diese zwischengespeichert und es wird die Antwort geliefert, dass Daten gefunden wurden.

Twitter und Exif Wrapper

Für eine Integration von Daten, die nicht aus RDF-Datenquellen stammen, werden zwei zusätzliche Wrapper realisiert. Der Twitter-Wrapper nutzt die von Twitter bereitgestellte API und bildet die abgerufenen Daten auf die SIOC-Ontologie ab. Er lässt sich lediglich auf URIs der Domain `twitter.com` anwenden und auf solche, die als `foaf:OnlineAccount` deklariert wurden und als `foaf:accountServiceHomepage` die Twitter-Homepage festlegen.

Der Exif-Wrapper lässt sich auf URIs anwenden, die durch ein entsprechendes Suffix auf das JPEG- oder TIFF-Format schließen lassen. Des Weiteren werden URIs unterstützt, die

vom Typ `foaf:Image` sind, wobei weitere Klassen konfiguriert werden können. Mit dem Exif-Wrapper werden vorhandene Exif-Daten aus Bildern extrahiert und auf das Exif-RDF-Schema[45] abgebildet.

Beide Wrapper werden im Gegensatz zum Linked Data Wrapper lediglich als Prototypen implementiert, um die Funktionalität der Wrapper-Architektur zu evaluieren.

5.3.2. Synchronisation

Die Synchronisation von Statements mit einer Datenquelle erfolgt ebenfalls auf Basis der Wrapper-Architektur. Zusätzlich wird für jede synchronisierbare Ressource eine Konfiguration angelegt, welche die für eine Synchronisation benötigten Informationen enthält. Bspw. wird über diese Konfiguration auch festgelegt, welcher Teil der gefundenen Daten mit der lokalen Datenbasis synchronisiert werden soll. Dies wird über die Definition einer SPARQL-Anfrage realisiert, die auf die gefundenen Daten angewendet wird. Die hier verwendete Synchronisation beschreibt ein unidirektionales Vorgehen, d. h. es werden lediglich lokale Daten aktualisiert und keine externen Daten geschrieben.

Vokabular

Das Vokabular wird in den Namesraum `http://ns.ontowiki.net/SysOnt/` integriert, der im Folgenden mit `sysont` abgekürzt wird. Für die Definition einer Synchronisations-Konfiguration werden eine Klasse und verschiedene Eigenschaften bzw. Relationen definiert. Diese werden folgend aufgeführt und beschrieben.

- `sysont:SyncConfig` wird als `owl:Class` definiert und beinhaltet sämtliche Konfigurationsinstanzen. Des Weiteren wurde diese Klasse als `owl:equivalentClass` zu einer komplexen Klasse definiert, die Einschränkungen bezüglich der Kardinalität der Eigenschaften `sync:wrapperName`, `sync:targetModel` und `sync:syncResource` festlegt. Die genannten Eigenschaften sind für eine Konfiguration zwingend erforderlich und werden deshalb mit Hilfe von `owl:minCardinality` und dem Wert `1` entsprechend vorgeschrieben.
- `sysont:wrapperName` bezeichnet den Namen des Wrappers, der für die Synchronisation genutzt wird. Diese Eigenschaft wurde zum Einen als `owl:DatatypeProperty` definiert, da Wrapper über eine Zeichenkette identifiziert werden. Zum Anderen wurde

[45] `http://www.w3.org/2003/12/exif/`

sie als `owl:FunctionalProperty` deklariert, da zu jeder Konfigurationsinstanz genau ein Wrapper erforderlich ist.

- `sysont:syncQuery` definiert eine SPARQL-Anfrage, die auf die erzeugten Daten angewendet wird, bevor diese synchronisiert werden. Sie ist ebenfalls vom Typ `owl:DatatypeProperty`, sowie `owl:FunctionalProperty`.
- `sysont:lastSyncDateTime` legt den Zeitpunkt der letzten Synchronisation fest. Dabei wird der `xsd:dateTime` Datentyp eingesetzt. Diese Eigenschaft ist optional, muss aber, falls vorhanden, einen eindeutigen Wert besitzen. Daher wurde auch diese Eigenschaft als `owl:FunctionalProperty` definiert.
- `sysont:lastSyncPayload` enthält die Daten der letzten Synchronisation als serialisiertes PHP-Array. Dies ist notwendig, da sich die lokalen Daten zwischen zwei Synchronisationen ändern können. Um lediglich die Daten zu aktualisieren, die auch durch den Import aus der jeweiligen Datenquelle entstanden sind, muss eine Referenz auf die Daten zum Zeitpunkt der Synchronisation gehalten werden. Aus Kompatibilitäts- und Performanzgründen, wurde dieses Verfahren gewählt. Denkbar wäre auch eine Speicherung der Daten in einer der RDF-Serialisierungformen, wie z. B. RDF/XML. Diese Eigenschaft ist ebenfals optional, muss aber, falls vorhanden, wieder einen eindeutigen Wert besitzen. Daher wurde auch diese Eigenschaft als `owl:FunctionalProperty` deklariert.
- `sysont:checkHasChanged` gibt an, ob eine Ressource auf Updates überprüft werden soll. Diese Eigenschaft kann Werte aus dem Wertebereich `xsd:boolean` annehmen und legt fest, ob diese Synchronisationskonfiguration für eine automatische Synchronisation berücksichtigt werden soll. Auch hierbei handelt es sich um eine optionale Eigenschaft vom Typ `owl:FunctionalProperty`.
- `sysont:targetModel` definiert den Graphen, der die zu synchronisierenden Statements enthält. Dies ist notwendig, da OntoWiki mit verschiedenen Graphen arbeitet, anstelle eines einzelnen Graphen, der alle Daten enthält. Der Wertebereich dieser Relation wird mit `rdfs:range` auf die Klasse `sysont:Model` eingeschränkt. Diese Relation ist für jede Konfigurationsinstanz zwingend erforderlich (`owl:minCardinality = 1`) und muss einen eindeutigen Wert besitzen (`owl:FunctionalProperty`).
- `sysont:syncResource` bezeichnet die Ressource, deren Synchronisation beabsichtigt ist. Der Wertebereich dieser Eigenschaft wird mit `rdfs:range` auf den Datentyp `xsd:anyURI` eingeschränkt, d. h. es sind hier keine unbenannten Ressourcen und Li-

terale erlaubt. Auch für diese Eigenschaft gilt, dass jede Konfigurationsinstanz genau einen Wert für diese Eigenschaft besitzen muss. Entsprechend wurden auch für diese Eigenschaft die nötigen OWL-Einschränkungen vorgenommen.

Mit dem hier definierten Vokabular lassen sich alle für eine Synchronisation notwendigen Informationen modellieren. Das Schema ist vollständig kompatibel mit dem von OntoWiki und Erfurt verwendeten Vokabular für die Systemkonfiguration und kann von der Komponente in dieses integriert werden.

Automatische Synchronisation

Die automatische Synchronisation wird ausgeführt, wenn eine Ressource in der Detailansicht betrachtet wird und zusätzlich die entsprechende Konfigurationsoption aktiviert ist. Dazu wird der HTTP-Header der Ressource von der Datenquelle abgerufen und auf den `Last-Modified`-Header hin überprüft. Ist dieser vorhanden, wird der Zeitstempel mit dem Zeitpunkt der letzten Synchronisation verglichen. Ergibt sich, dass die Ressource auf der Seite der Datenquelle aktualisiert wurde, wird dem Nutzer die Option zur sofortigen Synchronisation angeboten. Es wäre außerdem möglich, dass eine Synchronisation ganz ohne Nutzereinflüsse gestartet wird.

5.3.3. Integration in die OntoWiki-Oberfläche

Die Integration in die OntoWiki-Oberfläche erfolgt an mehreren Stellen. Zum Einen werden Menü-Einträge hinzugefügt, die das Menü einer Ressource um Optionen zum Import und zur Synchronisation erweitern. Für jeden aktiven Wrapper, der die jeweilige Ressource behandelt, wird ein Menüeintrag erzeugt. Dabei wird unterschieden, ob eine Ressource für eine Synchronisation konfiguriert ist. Wenn eine Konfiguration existiert, wird eine Option zur Synchronisation angeboten. Wenn keine Konfiguration existiert, wird stattdessen die Import-Option angeboten. Zum Anderen wird ein Formular in OntoWiki integriert, das die Konfiguration der Synchronisation ermöglicht. Außerdem werden die von OntoWiki bereitgestellten Nachrichten genutzt, um den Benutzer über den Status des Imports oder der Synchronisation zu informieren. Diese werden ebenfalls verwendet, um den Nutzer zu informieren, wenn aktualisierte Daten vorliegen. In solch einem Fall wird ein Button in die Nachricht integriert, der mit der Synchronisationsfunktion belegt ist.

5.4. Multistore-Fähigkeit

Eine weitere Anforderung an das zu entwickelnde System, ist die Integration heterogener Datenquellen auf Modell- und Interwiki-Ebene. Um diese Anforderungen zu erfüllen, wird in einem ersten Schritt ein Backend in Erfurt integriert, das die Nutzung mehrerer Backends ermöglicht. Dieses Meta-Backend ist für den Zugriff auf die eigentlichen Datenquellen verantwortlich. Somit kann jede auf Erfurt basierende Applikation dieses Meta-Backend verwenden, da sich die Schnittstelle nicht ändert.

Erfurt macht intern Gebrauch von einem Systemmodell und für einige Funktionen wird ein SQL-basiertes Backend benötigt. Deshalb soll das Meta-Backend, im Folgenden Multistore genannt, eines der Standard-Backends voraussetzen (Default-Backend), um diese Funktionen bereitzustellen. Zusätzlich können weitere Backends konfiguriert werden. Die Konfiguration erfolgt über feste Konfigurationsparameter. Denkbar wäre aber auch eine Konfiguration über die Systemontologie. Neue Modelle können lediglich im Default-Backend angelegt werden und sämtliche SQL-Funktionen werden ebenfalls über das Default-Backend realisiert.

Darüber hinaus leitet Multistore alle Anfragen an das für die jeweilige Graph-URI konfigurierte Backend weiter. Dieser Prozess ist trivial, da lediglich eine Zuordnung von einem Graph zu einem Backend erfolgen muss. Diese Zuordnung ist durch die manuelle Konfiguration gegeben und bedarf deshalb keiner detaillierten Betrachtung. Die einzige Schwierigkeit ergibt sich bei der SPARQL-Funktionalität. Hier müssen Anfragen ggf. an unterschiedliche Backends weitergeleitet werden, wenn ein Graph andere Graphen importiert, die nicht in dem selben Backend gehalten werden. Die Ergebnisse von verteilten Anfragen müssen im Anschluss wieder zusammengeführt werden. Wirkliche Schwierigkeiten ergeben sich aber erst bei der Verwendung von `LIMIT` und `OFFSET`, sowie bei `DISTINCT`-Anfragen. Aus Komplexitätsgründen können diese Probleme hier nicht behandelt werden. Mit DARQ [Quilitz and Leser, 2008] existiert aber bereits ein Ansatz, um föderierte SPARQL-Anfragen zu ermöglichen. In diesem Zusammenhang gehen die Autoren auch auf die damit einhergehenden Probleme ein und stellen Lösungsansätze vor. Für den Fall, dass keine Beziehungen zwischen Graphen aus unterschiedlichen Backends bestehen, verhält sich Multistore wie ein normales Backend, mit dem Unterschied, dass mehrere Backends parallel in einer OntoWiki-Instanz integriert werden können.

5.4.1. SPARQL-Backend

Auf der Grundlage von Multistore, kann ein weiteres Backend spezifiziert werden, welches SPARQL-Endpunkte kapselt. Auf diese Weise lassen sich externe Datenquellen zur Laufzeit integrieren. Somit sind die Daten stets aktuell und synchron mit der Datenquelle. Da standardkonforme SPARQL-Endpunkte nicht schreibbar sind, sondern nur mit Hilfe des SPARQL-Protokolls befragt werden können, muss auf dieser Ebene lediglich die SPARQL-Funktionalität realisiert werden. Dies wird erreicht, indem eine HTTP-Anfrage an den konfigurierten Endpunkt gestellt wird, ggf. mit einer der konfigurierten Graph-URIs. Zusätzlich muss die vom Endpunkt generierte Antwort im SPARQL-Results Format geparst werden und in das von Erfurt verwendete Format umgewandelt werden. Daneben soll eine Funktion integriert werden, die eine HTTP Basis Authentifizierung erlaubt, so dass auch auf geschützte Endpunkte zugegriffen werden kann.

5.4.2. OntoWiki-Backend

Das OntoWiki-Backend ist eine nicht-standardkonforme Erweiterung des SPARQL-Backends und ist deshalb der Interwiki-Ebene zuzuordnen. Es soll dazu dienen, externe Datenquellen nicht nur lesen, sondern auch schreiben zu können. Diesbezüglich wird die Update-Schnittstelle von OntoWiki genutzt. Somit ist es möglich, externe Datenquellen, sowohl lesend, als auch schreibend zur Laufzeit in OntoWiki zu integrieren. Auf dieser Ebene wird die Authentifizierung über eine proprietäre Erweiterung des FOAF+SSL-Protokolls realisiert, die im folgenden Abschnitt spezifiziert wird.

5.5. Authentifizierung auf UI- und API-Ebene

Für die Authentifizierung werden zwei Erweiterungen spezifiziert, eine für das OpenID-Protokoll und eine für die FOAF+SSL Spezifikation. OpenID wird dabei lediglich auf UI-Ebene integriert. FOAF+SSL hingegen, wird sowohl auf UI-, als auch auf API-Ebene integriert, wobei die Integration auf API-Ebene durch eine Modifikation des Protokolls erreicht wird.

5.5.1. OpenID

OpenID soll in das System lediglich mit Konsumenten-Funktionalität (Consumer) integriert werden, d. h. Nutzer können sich mit einer vorhandenen OpenID am System registrieren und anmelden. Diese Funktionalität wird mit Hilfe von Zend in den Erfurt-Kern integriert. Es wird aus Komplexitätsgründen auf die Realisierung einer Produzenten-Komponente (Provider) verzichtet. Da OpenID jedoch weit verbreitet ist, wird die Implementierung einer Consumer-Komponente angestrebt. Für den Interwiki-Anwendungsfall ist OpenID aber nur begrenzt verwendbar, da die Authentifizierung lediglich auf UI-Ebene erfolgen kann. Zusätzlich müsste hier ein Protokoll wie OAuth zum Einsatz kommen, was die Komplexität des Systems jedoch erheblich steigern würde.

5.5.2. FOAF+SSL

FOAF+SSL soll ebenfalls in Erfurt integriert werden, um eine Authentifizierung auf UI- und API-Ebene zu ermöglichen. Für FOAF+SSL ist eine SSL/TLS-gesicherte Verbindung notwendig, wenn die Überprüfung im System selbst stattfinden soll. Es soll zusätzlich möglich sein einen externen Identity Provider (IdP) einzusetzen, der über eine gesicherte Verbindung angesprochen wird. Ein IdP übernimmt die Verifizierung eines Nutzerzertifikats und liefert die enthaltene WebID, wenn eine Überprüfung innerhalb des Systems nicht möglich ist. Liegt eine gesicherte Verbindung vor, kann dieser Schritt innerhalb von Erfurt ausgeführt werden. Hierfür sind die folgenden Schritte notwendig:

1. Auslesen der WebID aus dem Nutzerzertifikat,
2. Dereferenzierung der WebID und Parsen der RDF-Daten,
3. Vergleich der Schlüsselinformationen aus den FOAF-Daten mit den Schlüsselinformationen aus dem Zertifikat.

Wenn beide Schlüssel übereinstimmen, ist der Nutzer authentifiziert. In einem zweiten Schritt muss er autorisiert werden. Diesbezüglich wird überprüft, ob er in der lokalen Nutzerdatenbank vorhanden ist. Für den Fall, dass ein entsprechender Nutzer existiert, werden die nötigen Rechte gesetzt. In diesem Aspekt unterscheidet sich ein FOAF+SSL Nutzer nicht von einem Nutzer, der mit Nutzername und Passwort registriert wurde. Existiert zu der gegebenen WebID bisher kein Nutzer in der lokalen Datenbank, besteht die Möglichkeit, automatisch einen Nutzer anzulegen. Somit entfällt der Schritt einer Registrierung. Des Weiteren soll es möglich sein, dass Nutzer selbstsignierte Zertifikate erstellen können, wenn

OntoWiki entsprechend konfiguriert wurde. Zusätzlich soll eine Komponente in OntoWiki realisiert werden, die Nutzern WebIDs zur Verfügung stellt, wenn diese keine besitzen. Auf diese Weise wird OntoWiki zu einem Provider für WebIDs.

Für die Interwiki-Kommunikation ist es außerdem notwendig, dass verschiedene OntoWiki-Instanzen Daten austauschen können, ohne dass ein Nutzer direkt beteiligt ist. Hierfür wird das FOAF+SSL Protokoll in einer Weise erweitert, die eine Delegation von Nutzerrechten durch den Nutzer selbst ermöglicht. Die OntoWiki-Instanz, die im Namen eines Nutzers agieren soll, muss ebenfalls über eine WebID verfügen, im Folgenden AgentID genannt, damit diese gleichermaßen über FOAF+SSL authentifizierbar ist. Der Nutzer kann in den gleichen Daten, in denen auch die Schlüsselinformationen abgelegt sind, Aussagen darüber treffen, an welche Agenten er den Zugriff delegieren möchte oder anders gesagt, welche Agenten in seinem Namen agieren dürfen. Da der Nutzer Kontrolle über diese Informationen hat, kann er die Delegation jederzeit widerrufen. Für die Delegation wird eine neue Relation `sysont:delegatesAccess` eingeführt.

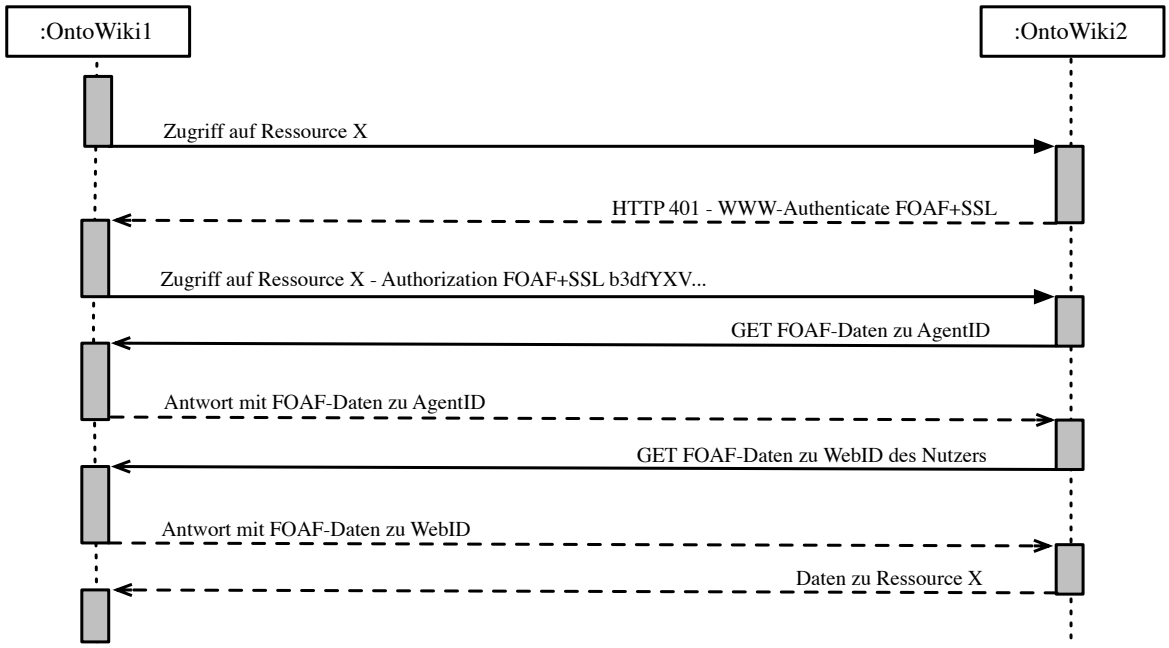

Abbildung 5.8.: Sequenzdiagramm zum Ablauf der Kommunikation unter Nutzung des erweiterten FOAF+SSL Protokolls. Es wird vorausgesetzt, dass der Nutzer auf beiden Instanzen einen Account und Zugriffsrechte auf die Ressource X besitzt.

Das authentifizierende OntoWiki überprüft zunächst die Echtheit des Agenten über das FOAF+SSL Protokoll. Wenn dieser Vorgang erfolgreich abgeschlossen wurde, liest die OntoWiki-Instanz die Nutzer-URI aus, die mit der Anfrage geliefert wurde. Hierfür wird ein `Authorization FOAF+SSL`-Header eingeführt, der einen Base64-kodierten Parameter der Form `ow_auth_user_key=`, gefolgt von der Nutzer-URI enthält. Da die OntoWiki-Instanzen über eine SSL/TLS geschützte Leitung miteinander kommunizieren und die Echtheit der anfragenden OntoWiki-Instanz verifiziert wurde, kann die antwortende OntoWiki-Instanz davon ausgehen, dass die Daten authentisch sind. In einem nächsten Schritt muss die Nutzer-URI dereferenziert werden. Letztere muss die Information enthalten, dass die anfragende Instanz berechtigt ist, im Namen des Nutzers Daten abzurufen. Wird dieser Schritt erfolgreich abgeschlossen, wird der Nutzer in einem letzten Schritt mit dem lokalen Zuganskontrollsystem überprüft und seine Rechte determiniert.

Außerdem kann bei fehlender Authentifizierung ein `WWW-Authenticate FOAF+SSL`-Header in die Antwort integriert werden, der signalisiert, dass die OntoWiki-Instanz mit dem erweiterten FOAF+SSL-Protokoll umgehen kann.

Dementsprechend wird es möglich, dass Nutzer verschiedener OntoWiki-Instanzen mit nur einem Nutzernamen, der WebID, arbeiten können, wobei die WebID bspw. von der Heimatinstanz bereitgestellt wird. Des Weiteren kann die Heimatinstanz, wenn vom Nutzer gewünscht, Aktionen im Namen des Nutzers ausführen, ohne den Nutzer selbst an der Kommunikation zu beteiligen. Abbildung 5.8 stellt dieses Szenario dar.

5.6. Zusammenfassung und Überblick

Tabelle 5.1 stellt zusammenfassend dar, wie die einzelnen Bestandteile des Systems in OntoWiki bzw. Erfurt integriert werden sollen. Abbildung 5.9 zeigt die zu implementierenden Klassen mit den zugehörigen Paketen und visualisiert die Beziehungen zwischen den einzelnen Bestandteilen des Systems. Im folgenden Kapitel erfolgt eine Betrachtung der konkreten Realisierung der einzelnen Komponenten in OntoWiki und Erfurt.

Ebene	**OntoWiki**	**Erfurt**
Ressourcen-Ebene	Datagathering Komponente	–
	Datagathering Plugin	
	Sindice Plugin	
	Adressleiste	
	Resource Widget	
Statement-Ebene	Datagathering Komponente	Wrapper-Architektur
	Datagathering Plugin	Linked Data Wrapper
		Twitter Wrapper
		Exif Wrapper
Modell-Ebene	–	Multistore Backend
		SPARQL Backend
Interwiki-Ebene	–	OntoWiki Backend
Metaebene Zugriffskontrolle	Auth Komponente	OpenID Consumer
		FOAF+SSL Consumer
		FOAF+SSL Provider

Tabelle 5.1.: Zuordnung der Bestandteile des zu implementierenden Systems zu den Ebenen der Datenintegration

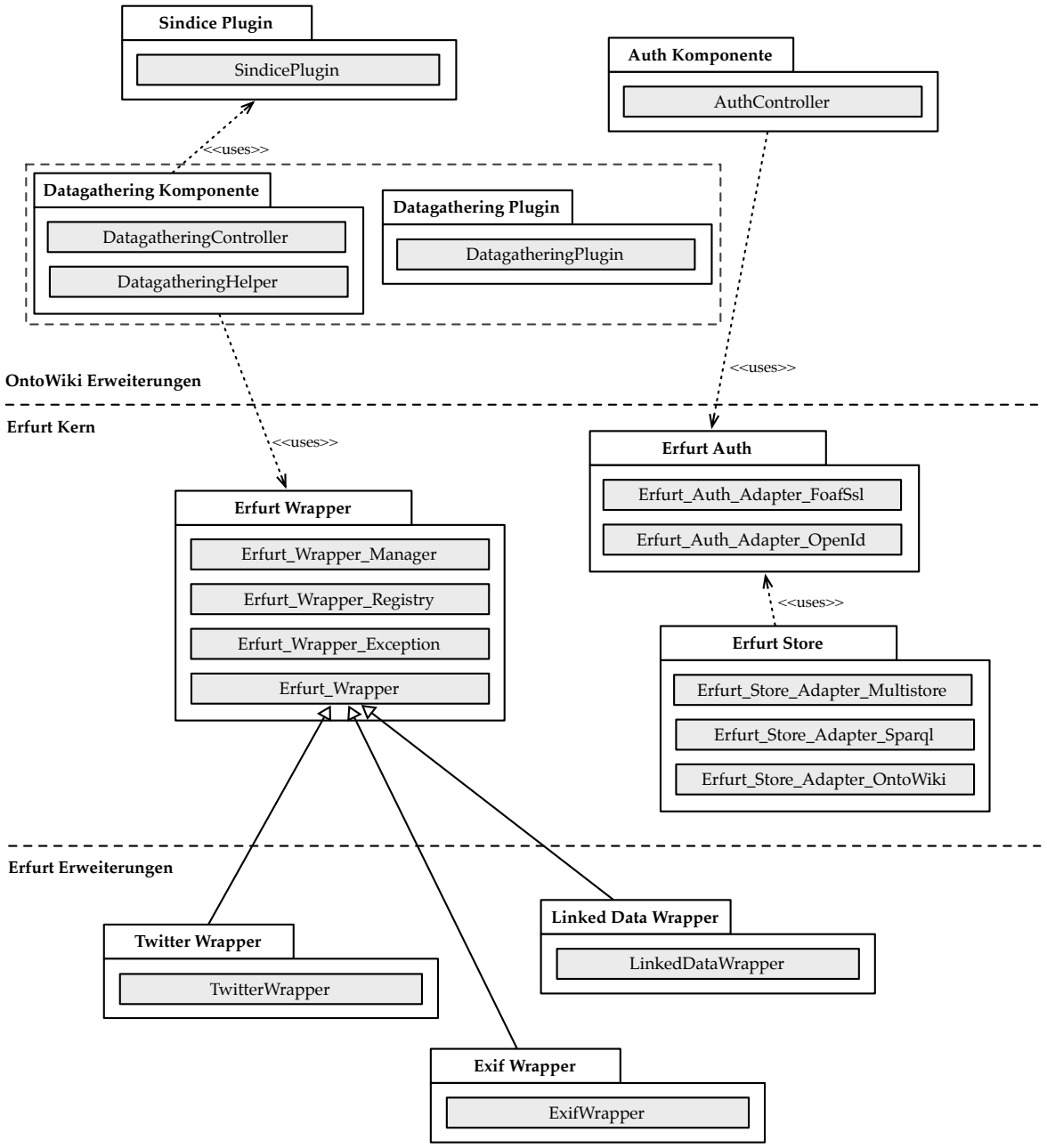

Abbildung 5.9.: Paketdiagramm der zu implementierenden Bestandteile mit den zugehörigen Klassen

6. Implementierung

6.1. OntoWiki-Erweiterungen

6.1.1. Datagathering-Komponente

DatagatheringController
+ searchAction(): void + testAction(): void + importAction(): void + initAction(): void + configAction(): void + syncAction(): void + modifiedAction(): void

Abbildung 6.1.: Controller-Klasse der Datagathering-Komponente mit öffentlichen Methoden

In der Datagathering-Komponente wird die Funktionalität zur Suche nach Ressourcen, zum Import von Statements und zur Synchronisation von Statements implementiert. Da es sich um eine OntoWiki-Komponente handelt, wird die Hauptfunktionalität über eine Controller-Klasse realisiert. Wenn im Folgenden von Parametern im Zusammenhang mit Controller-Klassen gesprochen wird, sind damit nicht die klassischen Übergabeparameter gemeint, die einer Funktion übergeben werden, sondern Anfrageparameter, die mit einer HTTP-Anfrage geliefert werden. Ähnliches gilt für Rückgabewerte. Im Kontext von Controllern werden die Resultate direkt in die HTTP-Antwort geschrieben. Abbildung 6.1 zeigt die Controller-Klasse mit sämtlichen Diensten, die sie bereitstellt.

Suche

Die Suchfunktion wird über die Funktion `searchAction` realisiert und über eine URL der Form `<OntoWikiBase>/datagathering/search` bereitgestellt, wobei `<OntoWikiBase>` für die OntoWiki Basis-URL steht. Der Dienst erwartet die folgenden Parameter:

- `q` enthält eine durch Leerzeichen separierte Stichwortliste. Dieser Parameter wird für die Suche vorausgesetzt. Fehlt er, wird die Anfrage mit einem HTTP-Statuscode 400

(Bad Request) beantwortet.

- `mode` bestimmt den Suchmodus. Dieser optionale Parameter kann die Werte `default` und `props` enthalten, wobei `default` der Standardwert ist und für den normalen Suchmodus steht. Wird stattdessen `props` für diesen Parameter übergeben, erfolgt eine Suche speziell nach Eigenschaften und Relationen.
- `limit` begrenzt optional die Ergebnismenge. Standardmäßig enthält die Antwort maximal 20 Ergebnisse. Die maximal mögliche Ergbniszahl ist aus Performanzgründen auf 100 beschränkt, wodurch sich ergibt, dass ein Wert zwischen 1 und 100 konfiguriert werden kann.

Wie in Algorithmus 2 (Seite 73) spezifiziert wurde, wird zunächst überprüft, ob der übergebene Term eine URI darstellt. Ist dies der Fall, wird eine leere Antwort erzeugt und die Suche beendet. Andernfalls wird zunächst die lokale Datenbasis per SPARQL befragt. Zusätzlich gelangt je nach Suchmodus einer der spezifizierten Anfragen zum Einsatz. Die so entstehende Teilantwort wird bezüglich ihrer Größe mit der konfigurierten maximalen Ergebnisgröße verglichen. Sollte das Limit noch nicht erreicht sein, werden evtl. vorhandene Suchplugins aktiviert. Hierfür wird ein neues Ereignis `onDatagatheringComponentSearch` eingeführt, dessen Konstruktion in Listing 6.1 dargestellt ist.

```
1 require_once 'Erfurt/Event.php';
2 $event = new Erfurt_Event('onDatagatheringComponentSearch');
3 $event->translate  = $this->_owApp->translate;
4 $event->termsArray = $termsArray;
5 $event->modelUri   = $modelUri;
6 $pluginResult = $event->trigger();
```

Listing 6.1: Code-Auszug aus der Suchfunktion der Datagathering Komponente

Anschließend werden der Reihe nach alle Ergebnisse dem Gesamtergebnis hinzugefügt, bis das Limit erreicht wird oder keine weiteren Ergebnisse mehr hinzugefügt werden können. Ist das Limit noch nicht erreicht, wird zusätzlich versucht, qualifizierte Namen aus den Stichworten zu expandieren. Abschließend wird eine URI mit Hilfe der Base-URI des Modells und den gegebenen Suchbegriffen generiert.

Da die Suchfunktion vorerst ausschließlich über asynchrone Anfragen mittels JavaScript verwendet wird, werden die Suchergebnisse JSON-kodiert ausgegeben. Um auf JavaScript-Seite eine effiziente Auswertung der Ergebnisse zu ermöglichen, besteht das Gesamtergebnis aus

```
Leipzig|http://example.org/terms/Leipzig|Lokale Suche
Lipsia - Lipsk - Leipzig|http://dbpedia.org/resource/Leipzig|Sindice Suche
leipzig|http://philipp.frischmuth24.de/id/Leipzig|Generierte URI
```

Listing 6.2: Beispielhaftes Ergebnis einer Suchanfrage

einem String, wobei jeder Eintrag eine eigene Zeile darstellt. Die Bestandteile eines Ergebnistupels werden durch ein | voneinander getrennt. Listing 6.2 zeigt ein beispielhaftes Ergebnis einer Suchanfrage. Aus Gründen der Lesbarkeit wurde dabei auf eine JSON-Kodierung verzichtet.

Import

```
$message = $translate->_(
    'Data was found for the given URI. %1$d statements were added.'
);

$this->_owApp->appendMessage(
    new OntoWiki_Message(
        sprintf($message, $stmtAddCount),
        OntoWiki_Message::SUCCESS)
);
```

Listing 6.3: Erzeugen einer OntoWiki-Nachricht mit Informationen zum Importverlauf

Für den Import von Statements werden zwei Dienste vom `DatagatheringController` bereitgestellt, einer zur Überprüfung der Verfügbarkeit von Daten und einer zum Import sämtlicher verfügbarer Daten in die lokale Datenbasis. Während Ersterer in der Methode `testAction` realisiert ist, wird der Import-Dienst in der Methode `importAction` umgesetzt. Beide Dienste setzen die folgenden Parameter voraus:

- `uri` gibt die zu überprüfende URI an und ist zwingend erforderlich. Ein Fehlen dieses Parameters führt zu einer Antwort mit einem HTTP-Status 400 (Bad Request).
- `wrapper` bezeichnet den zu nutzenden Wrapper. Fehlt dieser Parameter, wird standardmäßig der Linked Data Wrapper verwendet.

Die Funktion zum Testen auf Verfügbarkeit, ruft lediglich die von der Wrapper-Architektur vorgesehene Methode auf und gibt das Ergebnis JSON-kodiert an den anfragenden Client

weiter. Die Import-Funktion testet zunächst ebenfalls die Verfügbarkeit von Daten und führt den jeweiligen Wrapper im Erfolgsfall aus. Sind im Resultat des Wrappers hinzuzufügende Daten vorhanden, werden diese im Anschluss importiert.

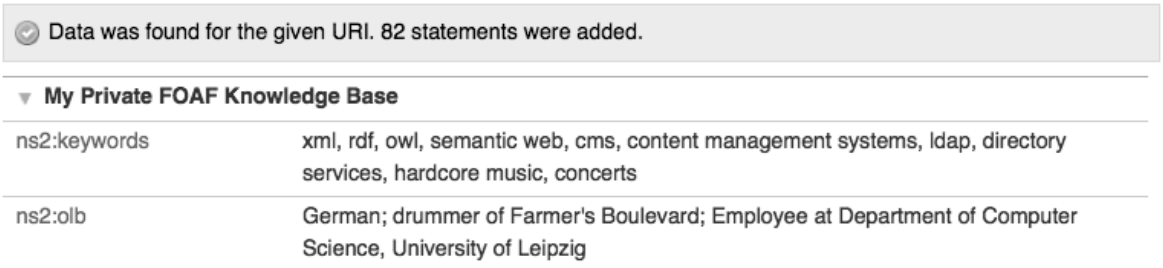

Abbildung 6.2.: OntoWiki-Nachricht aus Nutzersicht nach erfolgreichem Import von Statements

Zusätzlich werden für den Nutzer hilfreiche Informationen generiert und über das OntoWiki-Nachrichtensystem publiziert. Listing 6.3 zeigt die Generierung einer solchen Nachricht und Abbildung 6.2 das daraus folgende Resultat für den Nutzer.

Synchronisation

Die Funktionalität zur Synchronisation wird über insgesamt vier Dienste realisiert. Diese werden im Folgenden nacheinander vorgestellt.

`initAction` – Dieser Dienst ist für die Initialisierung der Synchronisationsumgebung zuständig und ausschließlich für den Administrator der OntoWiki-Instanz von Bedeutung. Es wird hier das Synchronisationsmodell erstellt, falls ein solches konfiguriert wurde. Anderenfalls wird die von OntoWiki und Erfurt verwendete Systemontologie eingesetzt. Daneben wird das in Kapitel 5 spezifizierte Vokabular in das Synchronisationsmodell importiert.

`configAction` – Über diese Funktion wird die Konfiguration einer Ressource für eine Synchronisation realisiert. Für diesen Dienst wird ein Template bereitgestellt, das ein Formular für die Eingabe der nötigen Informationen enthält. Abbildung 6.3 zeigt ein Bildschirmfoto des Formulars, wie es für den Nutzer dargestellt wird.

Die Konfiguration erfolgt in mehreren Schritten. Zunächst wird das Formular mit Standardwerten befüllt und dem Nutzer präsentiert. Die Formularwerte für die Uri, den Wrapper und das Modell sind dabei festgelegt und können nachträglich nicht verändert werden.

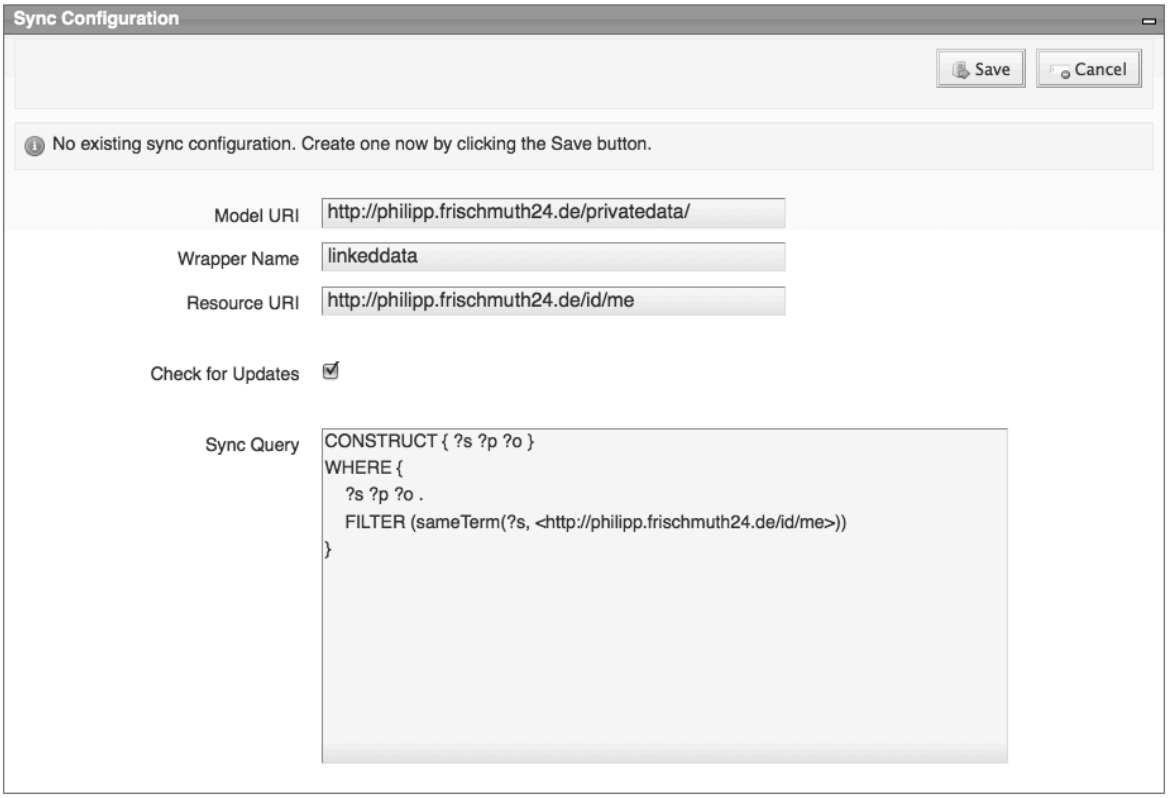

Abbildung 6.3.: Bildschirmfoto des Formulars für die Synchronisations-Konfiguration

Sie werden mit Hilfe von Aufrufparametern definiert. Diese sind identisch zu denen des Importdienstes. In einem zweiten Schritt werden die Formulardaten ausgewertet und eine Konfiguration im Synchronisationsmodell angelegt bzw. aktualisiert.

`syncAction` – Hier wird der eigentliche Synchronisationsvorgang ausgeführt. Auch hier werden die Parameter `uri` und `wrapper` erwartet, wobei letzterer optional ist. Ist dieser nicht vorhanden, werden sämtliche Synchronisationskonfigurationen genutzt, die für die gegebene URI gefunden werden. Für die Synchronisation wird in einem ersten Schritt die entsprechende Konfiguration abgerufen. Falls keine solche gefunden wird, bricht der Vorgang ab. Bei Erfolg werden anschließend die folgenden Schritte durchgeführt:

1. Die aktuellen Daten werden von der Datenquelle abgerufen.
2. Die abgerufenen Daten werden mit Hilfe der konfigurierten SPARQL-Anfrage gefiltert.
3. Falls Informationen über den letzten Synchronisationsvorgang vorhanden sind, wer-

den die bei der letzten Synchronisation hinzugefügten Statements gelöscht.

4. Die gefilterten Daten werden zur lokalen Datenbasis hinzugefügt.
5. Die Informationen zur letzten Synchronisation werden aktualisiert.

```
$headers = get_headers($uri, true);

if (isset($headers['Last-Modified'])) {
    $ts = 0;
    if (is_array($headers['Last-Modified'])) {
        foreach ($headers['Last-Modified'] as $mod) {
            $temp = strtotime($mod);
            if ($temp > $ts) {
                $ts = $temp;
            }
        }
    } else {
        $ts = strtotime($headers['Last-Modified']);
    }
}
```

Listing 6.4: Abrufen und Auswerten der Header-Informationen einer Ressource

modifiedAction – Dieser Dienst überprüft, ob eine Ressource seit der letzten Synchronisation aktualisiert wurde. Diesbezüglich ist es erforderlich, dass in der Konfiguration für die Synchronisation einer gegebenen Ressource die Option aktiviert ist, dass eine Überprüfung durchgeführt werden soll. Daneben muss der Zeitpunkt der letzten Synchronisation verfügbar sein. Um das Datum der letzten Bearbeitung der externen Ressource zu bestimmen, wird die Ressource angefragt und der Antwort-Header ausgewertet. Listing 6.4 zeigt die Stelle der Funktion, die für diesen Vorgang verantwortlich ist.

Sollte der Zeitstempel der Ressource aktueller sein, als jener der letzten Synchronisation, wird der Zeitstempel der letzten Modifizierung in die Antwort der Anfrage integriert. In allen anderen Fällen, wird `false` in die Antwort geschrieben. Somit kann auf JavaScript-Seite entschieden werden, ob eine Ressource aktualisiert wurde und ggf. sogar der exakte Zeitpunkt angegeben werden.

Weitere Bestandteile

Zu den weiteren Bestandteilen der Datagathering Komponente gehört die Hilfsklasse `DatagatheringHelper`. Im Gegensatz zur Controller-Klasse, wird diese Klasse bei jeder OntoWiki-Anfrage instanziiert, vorausgesetzt, die Komponente ist aktiv. Die Klasse besitzt eine `init`-Methode, in welcher die zur Komponente gehörende JavaScript-Datei `datagathering.js` inkludiert wird. Da die JavaScript-Funktionen teilweise das jQuery-autocomplete-Plugin[46] einsetzen, wird diese Bibliothek ebenfalls eingebunden.

```
1 var term = "leipzig geonames";
2
3 uriSearch(term, function(result) {
4     // In der Variable result steht das Ergebnis der Suche.
5     alert(result);
6 });
```

Listing 6.5: Beispiel für die Verwendung der Suchfunktion auf JavaScript-Seite

In `datagathering.js` werden auf JavaScript-Seite die Voraussetzungen für die dynamische Suche über ein Eingabefeld geschaffen. Hierfür werden die Menü-Elemente und die Adressleiste (s. Datagathering-Plugin) an JavaScript-Funktionen gebunden, die bei einer Aktivierung durch den Nutzer ausgeführt werden. Ein Teil des Codes wird ausgeführt, sobald das Dokument gerendert wurde. Daneben werden die Funktionen `showLocationBar()`, `hideLocationBar()` und `uriSearch(term,cb)` implementiert, die zum Einen für die Sichtbarkeit der Adressleiste und zum Anderen für das Ausführen der Suchfunktion verantwortlich sind. Listing 6.5 zeigt, wie die `uriSearch`-Funktion verwendet wird.

Des Weiteren enthält die Komponente noch zwei Übersetzungsdateien `datagathering-de.csv` und `datagathering-en.csv`. Diese beinhalten die Übersetzung einer Reihe lokalisierbarer Strings, um die Elemente der Benutzungsoberfläche auf die jeweils eingesetzte Sprache anzupassen.

[46] `http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/`

6.1.2. Datagathering-Plugin

Das Datagathering-Plugin besteht aus der Klasse `DatagatheringPlugin` und behandelt fünf, innerhalb von OntoWiki aktivierte Ereignisse. Dementsprechend erfolgt die Implementierung von fünf Methoden, die jeweils ausgeführt werden, sobald das entsprechende Ereignis eintritt. Es erfolgt im Folgenden eine Betrachtung und Beschreibung dieser Ereignismethoden.

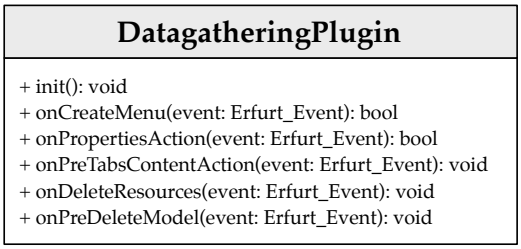

Abbildung 6.4.: Plugin-Klasse des Datagathering-Plugins mit öffentlichen Methoden

`onCreateMenu` ist die beim Erzeugen des OntoWiki Ressourcen-Menüs aufgerufene Methode. Dies betrifft sowohl das Kontextmenü einer Ressource, als auch das Menü am oberen Bildrand in der Detailansicht. Zunächst wird für alle aktiven Wrapper-Instanzen überprüft, ob sie die im Kontext befindliche Ressource behandeln und ob ggf. eine Synchronisationskonfiguration existiert. Existiert eine solche, wird ein entsprechender Menüeintrag generiert. Daneben wird ein Menüeintrag für die Konfiguration einer Synchronisation erzeugt. Die jeweiligen Einträge erhalten definierte CSS-Klassen, damit die entsprechende Aktion per JavaScript (s. Datagathering-Komponente) aktivierbar ist. Abbildung 6.5 zeigt ein so erweitertes Menü.

Abbildung 6.5.: Ausschnitt der Detailansicht mit aktiviertem Ressourcen-Menü – durch das Plugin hinzugefügte Adressleiste (1) und durch das Plugin hinzugefügte Menü-Einträge (2)

`onPropertiesAction` wird immer dann ausgeführt, wenn sich eine Ressource in der Detailansicht befindet. Hier entsteht u. a. der Menüeintrag für die Adressleiste, da dieser je nach aktuellem Status der Adressleiste (sichtbar oder nicht sichtbar) eine andere Funktion

besitzt. Daneben wird an dieser Stelle für Ressourcen, für die eine automatische Prüfung nach Aktualisierungen vorgesehen ist, der ensprechende HTML-Code erzeugt, der dem Nutzer diesen Umstand anzeigt. Die eigentliche Überprüfung findet auf JavaScript-Seite statt, um eine Verzögerung der Darstellung der Seite zu vermeiden. Die Darstellung für den Nutzer erfolgt mit Hilfe des OntoWiki-Nachrichtensystems. Abbildung 6.6 zeigt eine OntoWiki-Nachricht für den Fall, dass eine erfolgreiche Überprüfung stattgefunden hat. Sie beinhaltet sowohl den Zeitpunkt der letzten Modifizierung, als auch eine Schaltfläche, um die Ressource umgehend aktualisieren zu können.

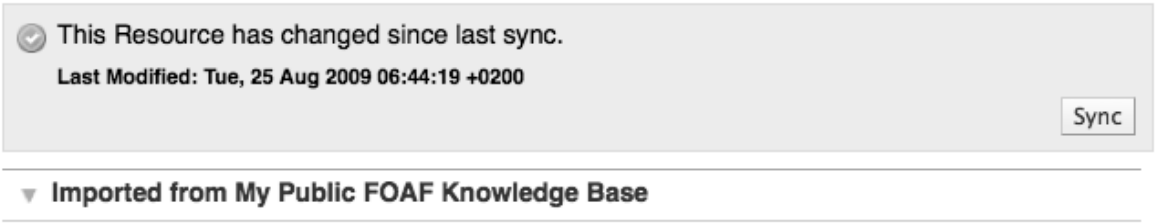

Abbildung 6.6.: Darstellung einer aktualisierten Ressource mit Informationen zum Zeitpunkt der letzten Modifizierung

`onPreTabsContentAction` ist für die Generierung und die Ausgabe des HTML-Codes für die Adressleiste zuständig. Die Adressleiste wird zunächst ausgeblendet, da die Aktivierung je nach Status auf JavaScript-Seite erfolgt. Abbildung 6.5 zeigt einen Ausschnitt der Detailansicht mit aktivierter Adressleiste.

`onDeleteResources` und **`onPreDeleteModel`** gelangen zum Einsatz, sobald eine Ressource gelöscht bzw. bevor ein Modell gelöscht wird. Die Behandlung dieser Ereignisse ist notwendig, da evtl. vorhandene Synchronisationskonfigurationen, die einer Ressource oder einem Modell zugeordnet sind, ebenfalls entfernt werden müssen.

Daneben beinhaltet das Datagathering-Plugin Dateien, die Übersetzungen für bestimmte Strings der Benutzungsoberfläche enthalten. Diese Übersetzungen sind in den Dateien `datagathering-de.csv` und `datagathering-en.csv` zu finden.

6.1.3. Auth-Komponente

AuthController
+ certAction(): void + usersAction(): void + agentAction(): void

Abbildung 6.7.: Controller-Klasse der Auth-Komponente mit öffentlichen Methoden

Die Auth-Komponente beinhaltet die Funktionalität, die OntoWiki um einen FOAF+SSL-Provider erweitert. Eine entsprechende Konfiguration der Umgebung vorausgesetzt, können mit dieser Komponente zum Einen WebIDs bereitgestellt werden. Zum Anderen ermöglicht diese Komponente die Bereitstellung einer eindeutigen AgentID, um das FOAF+SSL-Protokoll für eine Authentifizierung auf API-Ebene zu verwenden. Außerdem lassen sich selbstsignierte Zertifikate erzeugen. Um diese Funktionen zu erfüllen, stellt die Komponente drei Dienste bereit, die im Folgenden erläutert werden sollen.

Zertifikatserstellung

Dieser Dienst, der in der Methode `certAction` implementiert ist, ermöglicht es, einem Nutzer ein Zertifikat für eine Nutzung mit FOAF+SSL zu erstellen. Für diesen Vorgang ist eine gesicherte Verbindung zwingend erforderlich und auf dem Server, der OntoWiki beherbergt, muss OpenSSL verfügbar, sowie eine Certificate Authority (CA) eingerichtet sein. Sind diese Voraussetzungen erfüllt, kann der Nutzer über diesen Dienst

- eine neue WebID im Namensraum der OntoWiki-Instanz samt Zertifikat erstellen oder
- ein Zertifikat zu einer bestehenden WebID erzeugen.

Im zweiten Fall ist der Nutzer für die Übertragung der Schlüsselinformationen in die dereferenzierbaren Daten selbst verantwortlich. Im ersten Fall übernimmt diese Aufgabe die Komponente. Zusätzlich wird der Nutzer nach dem Erstellen des Zertifikats als Nutzer des Systems hinzugefügt. Für die Zertifikatserstellung kommt das `<keygen>`-Tag zum Einsatz, das im Browser ein neues Schlüsselpaar erzeugt und den öffentlichen Schlüssel, signiert, mit den Formulardaten an den Server übermittelt. Dieser kann mit den gegebenen Informationen ein signiertes Zertifikat erstellen, das die WebID als SAN enthält.

WebID und AgentID

Für WebIDs, die im OntoWiki-Namensraum erstellt werden, existiert ein spezieller Dienst, der die notwendigen RDF-Daten über die URI dereferenzierbar macht. Dieser Dienst ist in der Funktion `usersAction` implementiert. Alle von OntoWiki erzeugten WebIDs haben die Form `<OntoWikiBase>/auth/users/id/<UserId>`, wobei `<OntoWikiBase>` die OntoWiki Basis-URL bezeichnet und `<UserId>` der nutzerspezifische Teil einer WebID ist. Der Dienst erzeugt die RDF/XML-Serialisierung einer Nutzer-Ressource und gibt diese aus. Dabei wird die Symmetric Concise Bounded Description (SCBD) [Stickler, 2005] der Ressource erzeugt, um auch die Schlüsselinformationen mit zu erfassen, die über einen anonymen Knoten mit der Ressource verbunden sind. Die SCBD einer Ressource enthält alle Statements, bei denen die Ressource als Subjekt auftritt, wobei von verbundenen anonymen Knoten rekursiv ebenfalls die CBD gebildet wird, solange bis eine URI oder ein Literal auftritt. Daneben werden alle Statements mit einbezogen, bei denen die Ressource als Objekt auftritt.

Eine AgentID kommt im Zusammenhang mit der Authentifizierung auf API-Ebene zum Einsatz. Jede OntoWiki-Instanz mit aktivierter Auth-Komponente erhält eine AgentID, die wie eine WebID dereferenziert werden kann. Dieser Dienst ist in der `agentAction` realisiert. Jede OntoWiki-Instanz stellt genau eine AgentID bereit, deshalb sind die zu exportierenden RDF-Daten direkt in die Methode eingebettet. Die variablen Teile, die Schlüsselinformationen betreffend, lassen sich über die Konfiguration der Komponente anpassen.

6.1.4. Sindice Such-Plugin

Das Sindice Such-Plugin behandelt das `onDatagatheringComponentSearch`-Ereignis, das von der Datagathering-Komponente eingeführt und von dieser aktiviert wird. Die entsprechende Methode erzeugt eine HTTP-Anfrage an die Sindice-API, dessen Ergebnis ein JSON-kodiertes Array darstellt. Dieses wird dekodiert und anschließend verarbeitet. Die Verarbeitung besteht aus der Extraktion des Titels und des Links aus jedem Ergebnis. Das Plugin gibt ein Array zurück, das dann der Suchfunktion zur Verfügung steht.

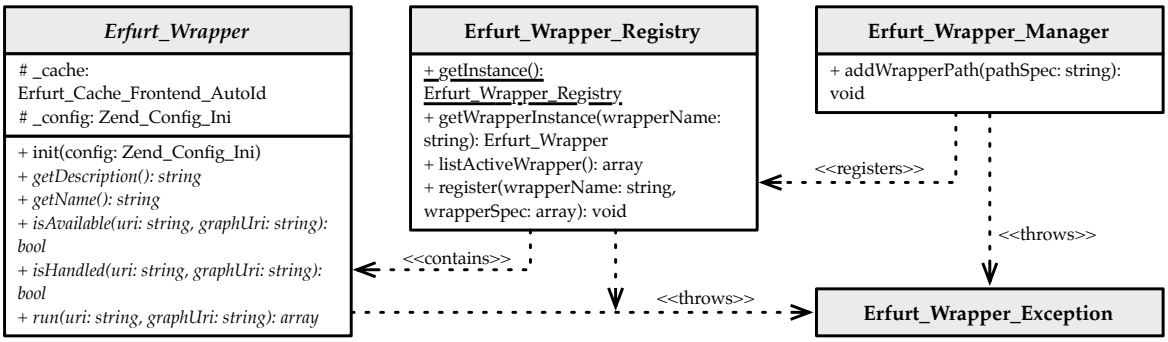

Abbildung 6.8.: Klassendiagramm des Wrapper-Pakets

6.2. Erfurt-Erweiterungen

6.2.1. Wrapper

Die Wrapper-Architektur wurde als Paket in Erfurt integriert und besteht aus vier Klassen, die in Abbildung 6.8 dargestellt sind. Die `Erfurt_Wrapper_Manager`-Klasse ist für das Durchsuchen von Verzeichnissen nach aktiven Wrappern verantwortlich. Zu diesem Zweck wurde eine Methode `addWrapperPath` implementiert, die als Parameter den Namen eines zu durchsuchenden Verzeichnisses erhält. Alle aktiven Wrapper werden über die zentrale `Erfurt_Wrapper_Registry` registriert. Diese Klasse wurde unter Anwendung des Singleton-Entwurfsmusters realisiert, d. h. es existiert zu jeder Zeit nur eine Instanz der Klasse, auf die über die Methode `getInstance` zugegriffen werden kann. Für die Registrierung eines Wrappers wird die Methode `register` verwendet, die als Parameter den eindeutigen Namen des Wrappers und ein Array mit der Wrapper-Spezifikation erhält. Dieses Array besitzt die folgenden Schlüssel:

- `include_path` – Enthält den vollständigen Pfad zum Verzeichnis des Wrappers.
- `class_name` – Enthält den Klassennamen des Wrappers, der verwendet wird, um den Wrapper zu instanziieren.
- `instance` – Ist anfänglich nicht gesetzt und enthält nach der Instanziierung das Objekt.
- `config` – Enthält ein Konfigurationsobjekt mit der Konfiguration des Wrappers.

Daneben existiert eine Methode `getWrapperInstance`, die das Objekt zum einem Wrapper liefert, der über einen Parameter spezifiziert wird. Die `listActiveWrapper`-Methode liefert

ein Array mit den Namen aller aktiven Wrapper.

Die `Erfurt_Wrapper`-Klasse bildet die Schnittstelle für sämtliche Wrapper-Erweiterungen. Sie realisiert die in Kapitel 5 spezifizierten Funktionen in Form der Methoden `isAvailable`, `isHandled` und `run`. Zusätzlich werden zwei weitere Methoden `getDescription` und `getName` eingeführt, die einer Integration in Oberflächen-Elemente dienen. Die Schnittstelle ist als abstrakte Klasse realisiert, um möglichen Wrapper Entwicklern einige Hilfsmittel an die Hand zu geben. So wird ein Cache-Objekt in die Klasse integriert, das nutzbar ist, um Daten über Anfragen hinweg zwischenzuspeichern. Des Weiteren stellt die Klasse ein Config-Objekt zur Verfügung, das die Konfiguration des Wrappers für eine weitere Nutzung verfügbar macht.

Alle am Wrapper-System beteiligten Klassen, inklusive der Wrapper-Erweiterungen, nutzen eine spezielle Ausnahme-Klasse, die den Namen `Erfurt_Wrapper_Exception` trägt. Sie dient dazu, bei der Nutzung der Wrapper-Architektur auf Fehler, die im Zusammenhang mit Wrappern entstehen, dediziert reagieren zu können.

Linked Data Wrapper

Der Linked Data Wrapper wurde auf Basis der Wrapper-Architektur entwickelt. Er kann RDF-Daten im RDF/XML-Format und im Turtle-Format verarbeiten. Daneben folgt der Wrapper `<link>`-Tags in HTML-Dokumenten, um für Ressourcen, die keine RDF-Daten direkt bereitstellen, Daten zu finden. Zusätzlich kann der Wrapper `Authorize`-Header auf Basis des erweiterten FOAF+SSL-Protokolls mit einer Anfrage senden, wenn dies von der Datenquelle verlangt wird. Auf diese Weise ist es möglich, mit dem Linked Data Wrapper auf geschützte Ressourcen zuzugreifen.

Twitter und Exif Wrapper

Neben dem Linked Data Wrapper sind Wrapper für die Twitter-API und Exif-Daten entstanden. Der Twitter-Wrapper greift über die Twitter-API auf Daten zu und bildet diese auf die SIOC-Ontologie ab. Der Exif-Wrapper extrahiert Exif-Daten aus JPEG- und TIFF-Bildern und bildet diese auf das Exif-RDF-Vokabular ab. Mit Hilfe dieser Wrapper wurde die breite Anwendbarkeit des Wrapper-Konzepts verifiziert.

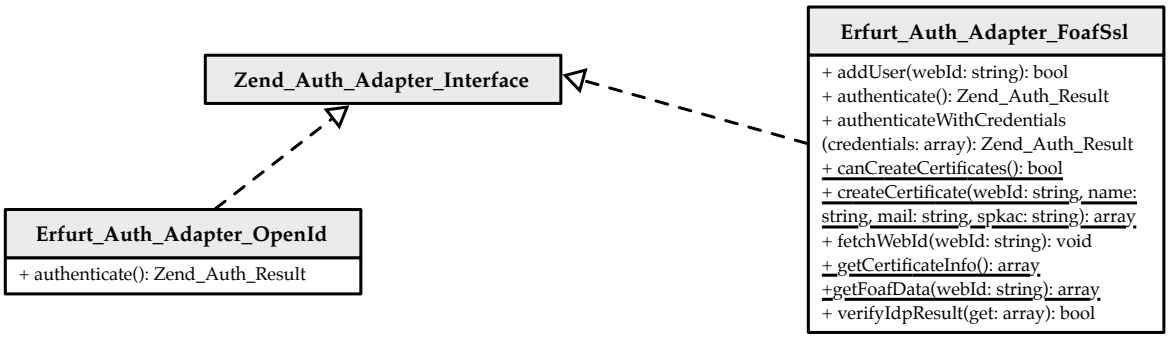

Abbildung 6.9.: Klassendiagramm mit den Adaptern für OpenID und FOAF+SSL

6.2.2. OpenID und FOAF+SSL

Es wurden für die Integration von OpenID und FOAF+SSL zwei neue Adapter-Klassen realisiert, die in Abbildung 6.9 dargestellt sind. Im Folgenden wird auf die Umsetzung der einzelnen Adapter näher eingegangen.

OpenID-Adapter

Der OpenID-Adapter implementiert eine einzige Methode `authenticate`, die von einer Zend-Schnittstelle festgelegt wird. Intern wird auf Zend-Funktionalität zurückgegriffen, um eine sichere Verbindung mit einem OpenID-Provider herzustellen. Die Methode gibt im Erfolgsfall keinen Wert zurück. Stattdessen wird der Nutzer in einem ersten Schritt zu seinem Provider weiter- und von dort an OntoWiki zurückgeleitet. Mit den übergebenen Parametern erfolgt der zweite Schritt der Authentifizierung. Die Verifizierung der Parameter geschieht ebenfalls durch Zend. Die Adapterklasse kapselt die Zend-Funktionalität und führt zusätzlich eine Überprüfung des Nutzers mit dem lokalen Zugriffskontrollsystem durch.

FOAF+SSL-Adapter

Der FOAF+SSL-Adapter implementiert eine Methode `authenticate`. Für den Authentifizierungsvorgang werden zwei verschiedene Vorgehensweisen verwendet. Wenn die auf Erfurt basierende Applikation in einer SSL/TLS gesicherten Umgebung arbeitet, findet die Authentifizierung innerhalb des Adapters statt. Anderenfalls, wird ein IdP eingesetzt. Dieser ist konfigurierbar und wird über eine HTTPS-Verbindung angesprochen. Der Nutzer wird,

wie bei OpenID, zum IdP weiter- und anschließend an eine zu bestimmende URL zurückgeleitet. In einem zweiten Schritt erfolgt eine Verifizierung der Antwort des IdP innerhalb des Adapters und bei Erfolg steht eine gültige WebID zur Verfügung. Im Anschluss kann überprüft werden, ob der Nutzer im lokalen System existiert und welche Rechte er besitzt. Eine vollständige Authentifizierung innerhalb des Adapters findet in 3 Schritten statt, die in der Spezifikation des Systems bereits erläutert wurden.

Des Weiteren stellt die Adapter-Klasse eine Methode `authenticateWithFoafSsl` bereit, die eine von einem Agenten gelieferte WebID authentifziert. Dazu wird zunächst der Agent (z. B. ein OntoWiki) mit dem FOAF+SSL-Protokoll authentifziert. Kann dieser Vorgang erfolgreich abgeschlossen werden, wird die vom Agenten übergebene WebID dereferenziert und auf ein Statement der Form <`WebId`> `sysont:delegatesAccess` <`AgentId`> überprüft, wobei <`WebId`> für die URI des Nutzers und <`AgentId`> für die URI des Agenten steht. Ist eine solche Information vorhanden, wird erneut geprüft, ob der Nutzer in der lokalen Datenbasis existiert.

Zusätzlich übernimmt der Adapter die Erstellung von Zertifikaten und das Anlegen neuer Nutzer, falls das System entsprechend konfiguriert ist. Das Anlegen neuer Nutzer wird zudem automatisch vorgenommen, wenn versucht wird, einen Nutzer mit einer gültigen WebID zu authentifizieren, dieser aber nicht im lokalen System registriert ist. Somit ist es möglich, Nutzer im System zu registrieren, ohne vorher den Umweg über ein Formular nehmen zu müssen.

6.2.3. Multistore

Um das Multistore-Konzept umzusetzen, werden drei neue Adapter-Klassen in das Store-Paket von Erfurt integriert, die im Folgenden jeweils vorgestellt werden. Abbildung 6.10 skizziert das generelle Store-Konzept und stellt die Konfigurationsmöglichkeiten, die sich mit Multistore ergeben, dar.

Multistore-Adapter

Die `Erfurt_Store_Adapter_Multistore`-Klasse bildet die Grundlage zur Nutzung verschiedener heterogener Backends in einer OntoWiki-Instanz. Die Klasse wurde so implementiert, dass stets ein Backend existiert, welches das System-Modell beinhaltet und zusätzlich SQL-Anfragen bearbeiten kann. Dieses Backend wird im Folgenden als Default-Backend be-

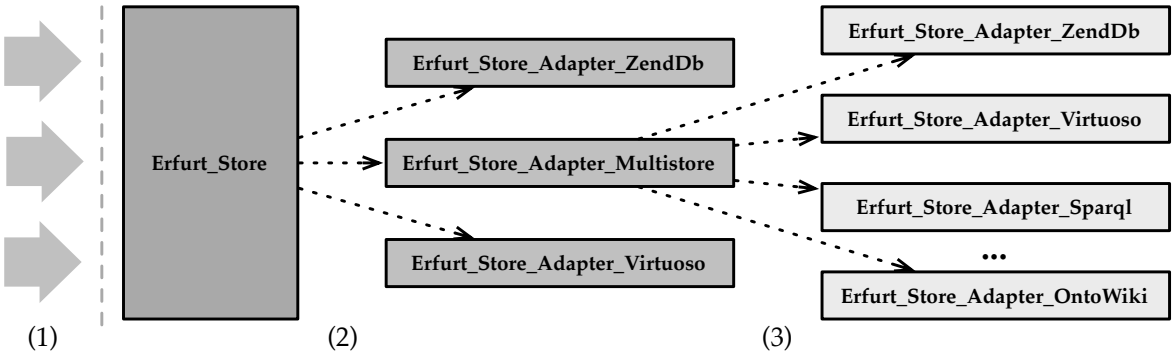

Abbildung 6.10.: Übersicht über die Multistore-Umgebung – (1) Eine Applikation, wie bspw. OntoWiki, greift ausschließlich über die Store-Klasse auf die Datenhaltung zu. (2) – Die Store Klasse verwendet intern genau ein konfiguriertes Backend, um auf Daten zuzugreifen. (3) – Multistore verteilt Anfragen intern an mehrere konfigurierbare Datenhaltungssysteme.

zeichnet. Für das Default-Backend kommen zur Zeit ausschließlich das ZendDb- und das Virtuoso-Adapter in Frage, da diese die für bestimmte Erfurt-Bestandeile notwendige SQL-Funktionalität bereitstellen. Zusätzlich zu genau einem Default Backend können beliebige weitere Backends konfiguriert werden. Dies wird über feste Konfigurationsparameter in der `ini`-Datei vorgenommen. Eine Konfiguration über das System-Modell ist derzeit nicht implementiert. Listing 6.6 zeigt einen Auszug einer möglichen Konfiguration mit einem Default-Backend und einem zusätzlichen SPARQL-Backend.

Sämtliche Methodenaufrufe werden an das jeweils zuständige Backend weitergeleitet. Da sämtliche Store-Methoden über einen Modell-Parameter verfügen, kann die Zuordnung eines Methodenaufrufs zu einem Backend über die Modell-URI erfolgen. Dementsprechend ist die Multistore-Umgebung so konfiguriert, dass verfügbare Modelle stets genau einem Backend zugeordnet sind. Lediglich bei der `sparqlQuery`-Methode kann es zu Problemen kommen, da Graphen evtl. weitere Graphen importieren, die einem anderen Backend zugeordnet sind. In solchen Fällen wird versucht, die Anfrage auf die verschiedenen Backends aufzuteilen und die Ergebnisse anschließend zu vereinigen.

SPARQL-Adapter

Die `Erfurt_Store_Adapter_Sparql`-Klasse ermöglicht den lesenden Zugriff auf entfernte SPARQL-Endpunkte. Sämtliche Methoden zum Editieren von Daten führen keinerlei

```
store.backend = multistore

store.multistore.default.backend         = zenddb
store.multistore.default.zenddb.dbname   = ontowiki
store.multistore.default.zenddb.username = ow
store.multistore.default.zenddb.password = ow
store.multistore.default.zenddb.dbtype   = mysql
store.multistore.default.zenddb.host     = localhost

store.multistore.additional.1.backend           = sparql
store.multistore.additional.1.sparql.serviceurl = "http://example.org/sparql"
store.multistore.additional.1.sparql.graphs.0   = "http://example.org/0.1/"
store.multistore.additional.1.graphs.0          = "http://example.org/0.1/"
```

Listing 6.6: Multistore-Konfiguration mit MySQL-Datenbank (Default) und einem zusätzlichen SPARQL-Endpunkt

Funktion aus, da hier das Standard-Protokoll umgesetzt wurde, das keine Aktualisierung von RDF-Graphen vorsieht. Daneben erfolgt die Implementierung einer HTTP Basis-Authentifizierung. Hierfür müssen Nutzername und Passwort in der Konfigurationsdatei angegeben werden.

OntoWiki-Adapter

Die `Erfurt_Store_Adapter_OntoWiki`-Klasse erweitert den SPARQL-Adapter um Funktionen zum Modifizieren von RDF-Graphen. Hierfür wird die von OntoWiki bereitgestellte Update-Schnittstelle verwendet, die über die HTTP-Parameter `insert` und `delete` die Aktualisierung von Graphen ermöglicht. Außerdem wird in diesem Adapter die Authentifizierung über das erweiterte FOAF+SSL-Protokoll realisiert. Somit ist es möglich, auf geschützte RDF-Graphen in entfernten OntoWiki-Instanzen zuzugreifen.

7. Zusammenfassung

Die Vision des Semantic Web sieht eine Erweiterung des ursprünglichen World Wide Web um eine semantische Schicht vor. Um diese Vision zu verwirklichen, gab es in den letzten Jahren große Bemühungen, wodurch eine Vielzahl neuer Standards, Technologien und Anwendungen entstanden sind. Die wichtigsten Neuerungen, sowie der Wandel des Webs wurden in Kapitel 2 dieser Arbeit vorgestellt und erörtert.

Die einheitliche Darstellung von Informationen in einem generischen Datenmodell, wie es RDF vorsieht, kann die Integration von Daten stark vereinfachen. Neue Applikationen, die dieses Datenmodell zugrunde legen, entstehen bzw. bereits existierende Applikationen werden an dieses angepasst. Der Bedarf, Informationen über Applikationsgrenzen hinweg zu integrieren, ist nicht erst mit den Bemühungen im Bereich des Semantic Web entstanden. Einhergehend aber mit neuen Technologien, ergeben sich hier neue Möglichkeiten. In Kapitel 3 wurden daher Anforderungen an ein System erarbeitet, das die Kommunikation zwischen semantischen Daten-Wikis im Hinblick auf die Datenintegration ermöglicht. Diesbezüglich wurden zunächst aussagekräftige Anwendungsfälle definiert, um im Anschluss die funktionalen und nichtfunktionalen Anforderungen an ein zu entwickelndes System abzuleiten.

Aufgrund intensiver Forschung in jüngster Zeit, existieren bereits eine Vielzahl an Publikationen und Prototypen, die Teilaspekte der für diese Arbeit relevanten Probleme thematisieren und Lösungsansätze anbieten. Verfügbare Problemlösungen mussten daher in die vorliegenden Betrachtungen mit einbezogen werden. Kapitel 4 gibt einen möglichst umfassenden Überblick über den aktuellen Entwicklungsstand und bewertet die jeweiligen Ergebnisse auf ihre Relevanz für diese Arbeit.

Aufbauend auf den Anforderungs- und Rechercheergebnissen, lässt sich prototypisch ein System zur Kommunikation zwischen Wiki-Instanzen in Bezug auf eine Datenintegration realisieren. Eine Integration sowohl auf Wiki-Ebene, als auch auf Graph-, Statement- und Ressourcen-Ebene, erscheint ebenso sinnvoll, wie die Integration von Sicherheitsmechanismen zum Schutz von vertrauenswürdigen Informationen. Um eine hohe Interoperabilität zu

erreichen, sollten nach Möglichkeit Standards zum Einsatz gelangen. In Kapitel 5 wurde ein System, das diesen Ansprüchen genügt, für eine Integration in OntoWiki spezifiziert. Kapitel 6 befasst sich darüber hinaus mit den Besonderheiten der konkreten Implementierung.

Evaluation

Im Rahmen der Evaluation wird das System hinsichtlich der gestellten Anforderungen bewertet. Das System unterstützt den Nutzer bei der Zuordnung von Ressourcen und der Auswahl von geeigneten URIs für Ressourcen. Diesbezüglich wurde eine Komponente in OntoWiki integriert, die eine erweiterbare Suchfunktion anbietet. Letztere lässt sich für die Suche nach beliebigen Ressourcen einsetzen, kann aber auch verwendet werden, um Eigenschaften und Relationen aufzuspüren. Wie gefordert, nutzt die Suche dabei lokale und externe Informationen, wobei externe Informationsquellen über Plugins eingebunden werden können. Die Erweiterbarkeit der Suchfunktion, konnte mit einem Plugin für den Sindice Suchdienst gezeigt werden. Eine Integration dieser Funktion in die OntoWiki Oberfläche erfolgte an jenen Stellen, an denen die Auswahl einer Ressource erforderlich ist. Insbesondere bei der Bearbeitung von Daten, einer für Wikis zentralen Funktion, wird die Suchfunktion bereits erfolgreich eingesetzt.

Daneben ermöglicht das System den Import und die Synchronisation externer Daten in die lokale Datenbasis. Die in Erfurt integrierte Wrapper-Architektur, gewährleistet die Erweiterbarkeit bezüglich verschiedener Datenquellen. Ein Wrapper für Linked Data, der auf dieser Architektur basiert, wird in der aktuellen OntoWiki Version für den Import von externen Daten intensiv verwendet. Somit konnte das gesamte System zum Datenimport bereits einer größeren Anzahl von Nutzern zugänglich gemacht werden. Dabei hat sich gezeigt, dass der Import sämtlicher verfügbarer Daten nicht immer sinnvoll ist, da mitunter sehr viele Daten für eine Ressource verfügbar sind. Daneben wurden in einem Praktikum weitere Wrapper realisiert und somit die Architektur evaluiert. Die Synchronisationsfunktion wurde anhand eines verteilten FOAF-Szenarios getestet. Dabei konnte die Funktionsweise gezeigt werden, ein Test mit einer größeren Nutzerzahl steht jedoch noch aus.

Das System unterstützt, wie gefordert, eine Datenintegration auf Modell- und Wiki-Ebene. Diesbezüglich wurde ein Multistore-Konzept umgesetzt, das die Integration von mehreren heterogenen Backends zur Laufzeit ermöglicht. Dieser Bestandteil des Gesamtsystems, wurde sowohl mit SPARQL-Endpunkten, als auch mit entfernten OntoWiki-Instanzen erfolgreich getestet.

Abschließend umfassen die in OntoWiki integrierten Erweiterungen einige Verbesserungen im Bereich der Authentifizierung auf UI- und API-Ebene. Diese Verbesserungen beinhalten die Unterstützung von OpenID und FOAF+SSL und wurden ebenenübergreifend an allen notwendigen Stellen des Systems integriert. Sämtliche vorgestelle Funktionen des Systems wurden, ebenfalls erfolgreich, mit geschützten Daten und einer damit einhergehenden Authentifizierung getestet.

Insgesamt lässt sich feststellen, dass fast alle Anforderungen an das System vollständig umgesetzt werden konnten. Teile des Systems konnten durch die Anwendung in einem Praktikum und durch die Integration in die aktuelle OntoWiki-Version[47] bereits einer intensiven Erprobung unterzogen werden.

Ausblick

Für eine produktive Nutzung des Systems in OntoWiki und Erfurt, bedürfen Teile des Systems einer Überarbeitung. Zukünftige Entwicklungen diesbezüglich könnten bspw. die Verbesserung des Importvorgangs betreffen. Der Nutzer sollte für jeden Importvorgang explizit auswählen können, welche Daten er importieren möchte. Statt bei einer Synchronisation alle Statements von der Datenquelle abzurufen, könnte eine zukünftige Version ein Verfahren wie RDFSync [Tummarello et al., 2007b] einsetzen. Daneben sollten Daten externer Endpunkte zwischengespeichert werden, um eine effiziente Ausführung von OntoWiki zu realisieren.

[47] OntoWiki 0.9 – `http://ontowiki.net/Projects/OntoWiki/Changelog#h456-2`

Anhang

A. Quellcode des Systems

Die Code-Basis von OntoWiki wird in einem Google Code Projekt gepflegt, das unter der URL `http://code.google.com/p/ontowiki/` erreichbar ist. Sämtliche Bestandteile des in dieser Arbeit entwickelten Systems wurden in das zugehörige Subversion-Repository eingepflegt und können über den Befehl

```
svn co http://ontowiki.googlecode.com/svn/trunk/ontowiki/src/ ontowiki
```

abgerufen werden. Revision `r4062` stellt den funktionstüchtigen Zustand des Systems zum Zeitpunkt der Abgabe dieser Arbeit dar.

Literatur

B. Adida and M. Birbeck. RDFa Primer. W3C Working Group Note, W3C, October 2008. URL http://www.w3.org/TR/xhtml-rdfa-primer/.

M. Atwood, D. Balfanz, D. Bounds, and R. M. Conlan. OAuth Core 1.0 Revision A. Specification, June 2009. URL http://oauth.net/core/1.0a.

S. Auer, S. Dietzold, and T. Riechert. OntoWiki – A Tool for Social, Semantic Collaboration. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *The Semantic Web – ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006. Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer, November 2006. ISBN 3-540-49029-9.

S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007, Proceedings*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, November 2007. ISBN 978-3-540-76297-3.

S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify – Light-Weight Linked Data Publication from Relational Databases. In *WWW'09, Proceedings of the 18th International World Wide Web Conference*, pages 621–630. ACM, April 2009. ISBN 978-1-60558-487-4.

D. Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, W3C, February 2004. URL http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

D. Beckett and T. Berners-Lee. Turtle – Terse RDF Triple Language. W3C Team

Submission, W3C, January 2008. URL http://www.w3.org/TeamSubmission/2008/SUBM-turtle-20080114/.

D. Beckett and J. Broekstra. SPARQL Query Results XML Format. W3C Recommendation, W3C, January 2008. URL http://www.w3.org/TR/2008/REC-rdf-sparql-XMLres-20080115/.

T. Berners-Lee. Linked Data. July 2006. URL http://www.w3.org/DesignIssues/LinkedData.html.

T. Berners-Lee and D. Connolly. Notation3 (N3): A readable RDF syntax. January 2008. URL http://www.w3.org/TeamSubmission/2008/SUBM-n3-20080114/.

T. Berners-Lee, J. Hendler, and O. Lassilla. The Semantic Web. *Scientific American*, 284(5): 34–44, May 2001.

T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax (RFC3986). Request for Comments, January 2005. URL http://www.ietf.org/rfc/rfc3986.txt.

T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, and D. Sheets. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *Proceedings of the The 3rd International Semantic Web User Interaction Workshop (SWUI2006)*, November 2006a.

T. Berners-Lee, W. Hall, J. A. Hendler, K. O'Hara, N. Shadbolt, and D. J. Weitzner. *A Framework for Web Science*, volume 1. now Publishers Inc., September 2006b. ISBN 1-933019-33-6.

T. Berners-Lee, J. Hollenbach, K. Lu, E. Prud'hommeaux, and M. Schraefel. Tabulator Redux: Writing Into the Semantic Web. Technical Report, November 2007.

C. Bizer, R. Cyganiak, and T. Heath. How to Publish Linked Data on the Web. Technical Report, July 2007. URL http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/20070727/.

A. Blumauer and T. Pellegrini. *Social Semantic Web: Web 2.0 - Was nun?* Springer, 2009. ISBN 978-3-540-72215-1.

P. Bouquet, H. Stoermer, and D. Giacomuzzi. OKKAM: Enabling a Web of Entities. In *Proceedings of the 16th International World Wide Web Conference (Banff, Canada)*. ACM, 2007.

P. Bouquet, H. Stoermer, and C. Niederee. Entity Name System: The Back-bone of an Open and Scalable Web of Data. In *IEEE International Conference on Semantic Computing (ICSC)*, 2008.

J. Breslin, A. Harth, U. Bojars, and S. Decker. Towards Semantically-Interlinked Online Communities. In A. Gómez-Pérez and J. Euzenat, editors, *The Semantic Web: Research and Applications Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29–June 1, 2005, Proceedings*, volume 3532 of *Lecture Notes in Computer Science*, pages 500–514. Springer, May 2005. ISBN 978-3-540-26124-7.

D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, W3C, February 2004. URL `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

D. Brickley and L. Miller. FOAF Vocabulary Specification 0.91. Specification, November 2007. URL `http://xmlns.com/foaf/spec/20071002.html`.

K. G. Clark, L. Feigenbaum, and E. Torres. SPARQL Protocol for RDF. W3C Recommendation, W3C, January 2008. URL `http://www.w3.org/TR/2008/REC-rdf-sparql-protocol-20080115/`.

D. Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C Recommendation, W3C, September 2007. URL `http://www.w3.org/TR/2007/REC-grddl-20070911/`.

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC 5280). Request for comments, May 2008. URL `http://www.ietf.org/rfc/rfc5280.txt`.

D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). Request for comments, July 2006. URL `http://tools.ietf.org/html/rfc4627`.

R. Cyganiak, R. Delbru, and G. Tummarello. Semantic Web Crawling: A Sitemap Ex-

tension. Specification, DERI Galway, July 2007. URL http://sw.deri.org/2007/07/sitemapextension/21112007.html.

F. Dawson and D. Stenerson. Internet Calendaring and Scheduling Core Object Specification (iCalendar). Request for comments, November 1998. URL http://tools.ietf.org/html/rfc2445.

M. Dzbor, J. Domingue, and E. Motta. Magpie - Towards a Semantic Web Browser. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*, pages 690–705. Springer, October 2003. ISBN 78-3-540-20362-9.

R. Fielding. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Dissertation, 2000.

R. Fielding and R. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology (TOIT)*, 2:2:115–150, 2002.

L. Gridinoc and M. d'Aquin. MOAW - URI's Everywhere. In *Proceedings of the 4th Workshop on Scripting for the Semantic Web*, June 2008.

T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 2:199–220, 1993.

P. Hayes. RDF Semantics. W3C Recommendation, W3C, February 2004. URL http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

N. Heino, S. Dietzold, M. Martin, and S. Auer. Developing Semantic Web Applications with the OntoWiki Framework. In T. Pellegrini, S. Auer, K. Tochtermann, and S. Schaffert, editors, *Networked Knowledge - Networked Media, Integrating Knowledge Management, New Media Technologies and Semantic Systems*, volume 221 of *Studies in Computational Intelligence*. Springer, 2009. ISBN 978-3-642-02183-1.

P. Hitzler, M. Krötzsch, S. Rudolph, and Y. Sure. *Semantic Web, Grundlagen*. Springer, 2008. ISBN 978-3-540-33993-9.

D. Huynh, S. Mazzocchi, and D. Karger. Piggy Bank: Experience the Semantic Web inside your web browser. In Y. Gil, E. Motta, V. Benjamins, and M. Musen, editors, *The Semantic Web – ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings*, volume 3729 of *Lecture Notes in Computer Science*, pages 413–430. Springer, November 2007. ISBN 978-3-540-29754-3.

G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, February 2004. URL `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

M. Krötzsch, D. Vrandecic, and M. Völkel. Semantic MediaWiki. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *The Semantic Web – ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006. Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 935–942. Springer, November 2006. ISBN 3-540-49029-9.

C. Lange. Krextor – An Extensible XML RDF Extraction Framework. In *Proceedings of the 5th Workshop on Scripting for the Semantic Web, 2009*, May 2009.

O. Lassilla and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, W3C, February 1999. URL `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.

B. Leuf and W. Cunningham. *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley, 2001. ISBN 978-0201714999.

F. Manola and E. Miller. RDF Primer. W3C Recommendation, W3C, February 2004. URL `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, W3C, February 2004. URL `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

T. O'Reilly. What Is Web 2.0? Design patterns and business models for the next generation of software. 2005.

A. Passant and P. Laublet. Towards an Interlinked Semantic Wiki Farm. In *Proceedings of the 3rd Semantic Wiki Workshop (SemWiki2008)*, 2008a.

A. Passant and P. Laublet. Meaning Of A Tag: A Collaborative Approach to Bridge the Gap Between Tagging and Linked Data. 2008b.

P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation, W3C, February 2004. URL `http://www.w3.org/TR/2004/REC-owl-semantics-20040210/`.

E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, W3C, January 2008. URL `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.

D. Quan, D. Huynh, and D. Karger. Haystack: A Platform for Authoring End User Semantic Web Applications. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*, pages 738–753. Springer, October 2003. ISBN 78-3-540-20362-9.

B. Quilitz and U. Leser. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008 Proceedings*, volume 5021 of *Lecture Notes in Computer Science*, pages 524–538. Springer, June 2008. ISBN 978-3-540-68233-2.

S. Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *1st International Workshop on Semantic Technologies in Collaborative Applications (STICA'06), Manchester, UK*, June 2006.

A. Seaborne and G. Manjunath. SPARQL/Update: A language for updating RDF graphs. Technical Report, HP Laboratories Bristol, July 2007.

P. Stickler. CBD – Concise Bounded Description. W3C Member Submission, W3C, June 2005. URL `http://www.w3.org/Submission/2005/SUBM-CBD-20050603/`.

H. Story, B. Harbulot, I. Jacobi, and M. Jones. FOAF+SSL: RESTful Authentication for the

Social Web, May 2009.

G. Tummarello, R. Delbru, and E. Oren. Sindice.com: Weaving the Open Linked Data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007, Proceedings*, volume 4825 of *Lecture Notes in Computer Science*, pages 552–565. Springer, November 2007a. ISBN 978-3-540-76297-3.

G. Tummarello, C. Morbidoni, R. Bachmann-Gmür, and O. Erling. RDFSync: Efficient Remote Synchronization of RDF Models. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007, Proceedings*, volume 4825 of *Lecture Notes in Computer Science*, pages 537–551. Springer, November 2007b. ISBN 978-3-540-76297-3.

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Liste der Algorithmen

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
DOAP	Description of a Project
ENS	Entity Name System
Exif	Exchangeable Image File Format
FOAF	Friend of a Friend
GRDDL	Gleaning Resource Descriptions from Dialects of Languages
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IdP	Identity Provider
IFP	Inverse Functional Property
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MOAT	Meaning of a Tag
MOAW	Meaning of a Word
MVC	Model View Controller
N3	Notation 3
OWL	Web Ontology Language
PHP	PHP Hypertext Preprocessor
RDF	Resource Description Framework
RDFS	RDF Schema
REST	Representational State Transfer
SAN	Subject Alternative Name
SCBD	Symmetric Concise Bounded Description
SIOC	Semantically-Interlinked Online Communities
SMW	Semantic Media Wiki
SOAP	Simple Object Access Protocol

SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SSL	Secure Sockets Layer
SyncML	Sync Markup Language
TLS	Transport Layer Security
Turtle	Terse RDF Triple Language
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language