UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Musikempfehlungen im Semantic Web

Implementation eines Onlineradios
mit Technologien des Semantic Web

Diplomarbeit

Leipzig, 30. November 2009
Betreut und korrigiert durch:
Prof. Dr. Ing. habil. Klaus-Peter Fähnrich
Dipl. Inf. Jens Lehmann

vorgelegt von Steffen Becker
geboren am: 05.04.1981
Studiengang Informatik
Matrikelnummer: 8963902

Abstract

Das Internet bietet heutzutage viele Möglichkeiten neue Musik zu entdecken. Sozial basierte Onlineradios und redaktionell betreute Plattformen stellen Nutzern die Möglichkeit zur Verfügung, sich Vorschläge generieren zu lassen. Die Informationen, die diesen Empfehlungen zugrunde liegen, bleiben jedoch meist im Verborgenen. Das Semantic Web aber bietet die Möglichkeit, vorhandenes Wissen über Musik zu vereinen und dieses auch frei zur Verfügung zu stellen. Dadurch, dass immer mehr Künstler ihre Werke kostenlos im Internet bereitstellen, lässt sich dieses Wissen direkt mit abspielbarer Musik verknüpfen. Durch die Maschinenlesbarkeit der verwendeten Daten können automatisierte Verfahren eingesetzt werden, um so dem Nutzer neue Empfehlungen zu generieren.
Zielsetzung dieser Arbeit ist die Entwicklung einer Webapplikation, die unter Verwendung der Technologien des Semantic Web Benutzern ein Onlineradio bereitstellt, das zugleich auch Musikempfehlungen erstellen kann. Für die Programmierung der Webapplikation werden frei verfügbare Daten und Programme herangezogen, die im weiteren Verlauf dieser Arbeit im Detail erläutert werden. Als Grundlage einer Vorschlagsgenerierung durch maschinelles Lernen wird eine Wissensbasis in Form einer Ontologie erstellt. Abschließend wird die Qualität der Musikempfehlungen evaluiert und Verbesserungsvorschläge werden erbracht. Dabei stellt die Arbeit stets einen Bezug zum aktuellen Stand der Forschung her.

Inhaltsverzeichnis

1 Einleitung

Onlineradios wie last.fm[1] oder pandora[2] ermöglichen es schon seit einigen Jahren Internetnutzern auf der ganzen Welt kostenfrei Musik zu hören und zu entdecken. Durch die Eingabe von Künstlernamen, Songtiteln oder sogenannten *Tags*, die meist zur Klassifizierung des Musikgenres verwendet werden, lassen sich Abspiellisten mit weiteren Titeln erstellen, die dem Benutzer ebenfalls gefallen könnten. Solche Radio-Plattformen sind gerade für weniger bekannte Künstler sehr interessant, da diese eine kostengünstige und bequeme Variante darstellen, die eigenen Werke zu vertreiben und bekannter zu machen.

Die meisten Onlineradios generieren diese Empfehlungen jedoch meist redaktionell oder auf der Grundlage einer Social Community. Das heißt, jeder Nutzer generiert durch aktives Hören und die Verschlagwortung von Titeln und Künstlern indirekt auch Empfehlungen für andere. Die direkten semantischen Zusammenhänge der so entstandenen Empfehlungen sind dadurch nur schwer nachvollziehbar, geschweige denn maschinell reproduzier- oder lesbar. Ein weiteres Problem bei den meisten Plattformen sind die Lizenzrechte der einzelnen Titel – manche Titel sind zwar im System vorhanden, allerdings lässt sich nur ein 30-Sekunden Ausschnitt des Titels abspielen, was für die meisten Nutzer unbefriedigend ist.

An diesem Punkt setzt die Musikplattform Jamendo[3] an, die es Musikern und Künstlern ermöglicht ihre Werke unter einer Creative-Commons Lizenz[4] zu veröffentlichen. Die Musik ist somit frei verfügbar und für jeden Benutzer, auch ohne Anmeldung, kostenlos und legal herunterladbar. Das DBTune-Projekt[5] hat es sich zur Aufgabe

[1] http://last.fm
[2] http://pandora.com
[3] http://jamendo.com
[4] http://creativecommons.org/
[5] http://dbtune.org, ist Teil der Initiative *Linking Open Data on the Semantic Web* (http://linkeddata.org/).

gemacht, musikrelevante Daten zu sammeln und mit Technologien des *Semantic Web* frei verfügbar bereitzustellen, so auch die Daten von Jamendo. Damit gibt es eine Plattform, die Musik in voller Länge, ohne rechtliche Probleme, bereitstellt und eine Schnittstelle, die die dazu vorhandenen Informationen semantisch miteinander verknüpft.

1.1 Zielsetzung der Arbeit

Ziel der Diplomarbeit *Musikempfehlungen im Semantic Web* ist die Entwicklung einer Webapplikation, die mit den Technologien des Semantic Web und gegebenen Nutzerdaten Musikempfehlungen erstellt und diese dann direkt im Browser abspielt. Zur Generierung der Empfehlungen wird das Machine Learning Tool *DL-Learner* der *Research Group for Agile Knowledge Engineering and Semantic Web* verwendet[6]. Als Grundlage für Anfragen und Musikdaten wird die frei verfügbare Musikdatenbank von Jamendo hinzugezogen, die über DBTune[7] bereitgestellt wird. Auf eine funktionsfähige Implementierung der Webapplikation wird ein hoher Wert gelegt. Ziel ist es im theoretischen und praktischen Teil der Diplomarbeit eine Schnittstelle zwischen der Webapplikation und dem DL-Learner zu erarbeiten und diese Schnittstelle auf Performance und Ergebnisgenauigkeit zu untersuchen und gegebenenfalls zu optimieren. Die Entwicklung einer geeigneten Ontologie als Grundlage für den verwendeten Lernalgorithmus ist dabei obligatorisch. Unumgänglich ist die Auseinandersetzung mit den unterschiedlichen zugrunde liegenden Technologien des Semantic Web wie *Resource Description Framework* (RDF), *SPARQL Protocol and RDF Query Language* (SPARQL), *Web Ontology Language* (OWL) und weiteren Konzepten zeitgemäßer Webapplikationen wie JavaScript und *Asynchronous JavaScript and XML* (AJAX).

[6] http://dl-learner.org/Projects/DLLearner
[7] http://dbtune.org/jamendo

1.2 Kapitelübersicht

In Kapitel 2, ab Seite 4, werden wichtige, in dieser Arbeit verwendete Technologien und Konzepte des Semantic Web erläutert, die zur Implementation der Webapplikation notwendig sind. Die im praktischen Teil zum Einsatz kommenden Technologien wie RDF und SPARQL werden vorgestellt und mit entsprechenden Beispielen erläutert.

Kapitel 3, ab Seite 25, behandelt generelle Konzepte zeitgemäßer Webapplikationen, geht auf die verwendeten Technologien und Prinzipien ein und zeigt, wie diese in der Arbeit eingesetzt werden. Des Weiteren werden spezifische Ressourcen wie zum Beispiel die verwendeten Ontologien und RDF-Daten sowie weitere Techniken wie AJAX vorgestellt, die zur programmiertechnischen Umsetzung benötigt werden.

Implementationsspezifische Details und Probleme bei der Umsetzung des praktischen Teils zur Programmierung der Webapplikation werden daraufhin in Kapitel 4, ab Seite 40, behandelt. Dieses Kapitel geht genauer auf die Erstellung der zugrunde liegenden Ontologie und die Anbindung an das verwendete Machine- Learning-Tool DL-Learner ein und beschreibt den generellen Aufbau der Webapplikation.

Nach der erfolgreichen Implementation der Webapplikation wird diese in Kapitel 5, ab Seite 57, auf Performance und Ergebnisqualität untersucht. In diesem Kapitel werden die Optimierungsmöglichkeiten zur Verbesserung des Systems vorgestellt und allgemein beobachtete Probleme während der Implementation und bei der Benutzung der Webapplikation geschildert.

In Kapitel 6, ab Seite 69, wird die Arbeit in den aktuellen Forschungsstand eingeordnet. Gleichzeitig wird das Ergebnis der Arbeit kurz zusammengefasst und ein Ausblick auf eine weitere Entwicklung gegeben.

2 Grundlagen

In diesem Kapitel werden der Begriff 'Semantic Web' und die in dieser Arbeit verwendeten grundlegenden Technologien des Semantic Web kurz erläutert und in Hinblick auf ihre spätere Anwendung bei der Implementierung mit passenden Beispielen erklärt.

2.1 Das Semantic Web

Das *World Wide Web* (Web) ist aus dem heutigen Alltag nicht mehr wegzudenken und für viele Menschen zentrales Instrument zur Kommunikation und zum Informationsaustausch geworden. Durch die rasante Entwicklung des Webs in den letzten Jahrzehnten, sowohl auf infrastruktureller als auch technologischer Basis, ist es heutzutage möglich jedwede Art von Daten oder Informationen – theoretisch von jedem Ort aus – stets aktuell zu halten und diese zur Verfügung zu stellen.
Der weitreichende Ausbau der Infrastruktur des Webs hat diese Entwicklung maßgeblich vorangetrieben, so dass es heute für fast jeden möglich ist mit geringen Kosten an dieser Informationsverbreitung teilzuhaben. Breitbandanschlüsse sind in immer mehr Haushalten verfügbar[8] und auch das mobile Internet wird mit dem stetigen Ausbau von UMTS-Netzen für immer mehr Menschen bezahl- und realisierbar. Das Web wächst stetig an, und damit auch die Menge an verfügbaren Informationen – es wird somit anscheinend immer unüberschaubarer.

Diese Informationen sind hauptsächlich auf den Menschen als Endnutzer ausgerichtet, ohne jegliche direkte semantische Kennzeichnung der Informationen. Des Weiteren gibt es eine nicht überschaubare Anzahl an unterschiedlichen Dateiformaten

[8] Vergleiche Households with broadband access (2004-2008), OECD Broadband Portal, http://www.oecd.org/dataoecd/20/59/39574039.xls.

oder Kodierungen. Für eine Person stellt das in den meisten Fällen kein Problem dar, da sich die Bedeutung der Informationen entweder durch deren Darstellung, den inhaltlichen Kontext oder durch vorhandenes Vorwissen erschließt. So kann ein Mensch zum Beispiel unterscheiden, welche Ergebnisse bei einer Suche[9] nach „korn“ der amerikanischen Rockband oder dem Getreide zuzuordnen und somit für die suchende Person relevant sind. Allerdings wäre es in einem solchen Fall wesentlich sinnvoller eine semantische Suche auszuführen und bereits im Vorfeld die Ergebnismenge auf „Rockband aus Amerika“ einschränken zu können.

An diesem Punkt setzt das Semantic Web an:

> 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. (Berners-Lee et al., 2001)

Die Grundidee des Semantic Web besteht also darin,

> [...] Wege und Methoden zu finden um Informationen so zu repräsentieren, dass Maschinen damit in einer Art und Weise umgehen können, die aus menschlicher Sicht nützlich und sinnvoll erscheint. (Hitzler et al., 2007)

Für die Umsetzung einer solchen Idee bedarf es offener, einheitlicher und flexibler Standards, damit es möglich ist Informationen auszutauschen und diese miteinander in Beziehung zu setzen. Das *World Wide Web Consortium* (W3C)[10] setzt sich seit seiner Gründung 1994 unter anderem dafür ein, ebensolche Standards zu erschaffen und hat mit *eXtensible Markup Language* (XML), RDF, OWL und weiteren Technologien den Grundstein für das Semantic Web gelegt. *Semantik* im Fall des Semantic Web ist die Bedeutung von Zeichenketten oder Worten und deren Beziehung untereinander.

[9] Bei einer der gängigen Suchmaschinen wie http://google.com, http://yahoo.com oder http://bing.com.
[10] http://w3c.org

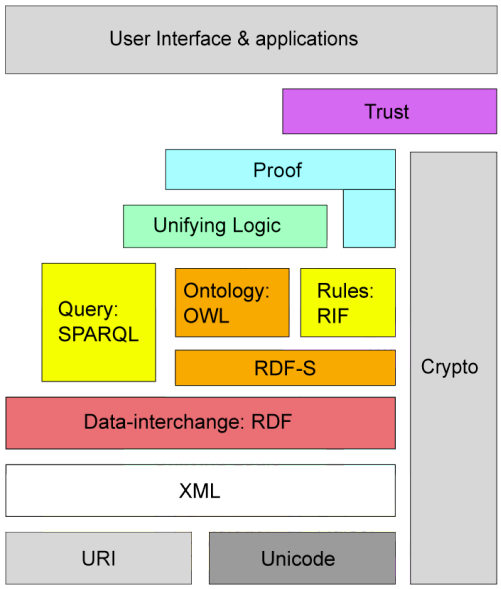

Abbildung 1: Der Semantic-Web-Stack: die Architektur des Semantic Web

Abbildung 1[11] zeigt die Architektur des Semantic Web und beschreibt dessen hierarchische Schichten. Jede obere Schicht verwendet Teilbereiche aus den darunter liegenden Schichten. Die unteren fünf beschreiben Sprachen und Konzepte, die zur Übermittlung und Strukturierung von Informationen eingesetzt werden. Diese werden in der logischen Schicht (*Unifying Logic*) zusammengeführt, woraufhin in der Berechnungsschicht (*Proof*) durch logisches Schlussfolgern Wissen abgeleitet werden kann. In all diesen Schichten muss es möglich sein, Informationen zu signieren, um eine Vertrauenswürdigkeit zu realisieren (*Cryptograhpy*). In der letzten Stufe der Hierarchie (*Trust*) wird dann entschieden, ob dieses Wissen und die Informationen vertrauenswürdig sind, so dass diese in Programmen verwendet werden können (*User interface & applications*).

Die Technologien, die im praktischen Teil dieser Arbeit zum Einsatz kommen, werden nun im Detail vorgestellt.

[11] Quelle: http://upload.wikimedia.org/ wikipedia/commons/5/54/SW_layercake_2006.svg

2.2 XML und JSON als Datenstrukturen

XML ist eine Teilmenge von *SGML* (Standard Generalized Markup Language) und somit eine Auszeichnungssprache oder Markupsprache, mit der sich Daten in Form von Text logisch strukturieren lassen. XML-Dokumente sind menschen- und maschinenlesbar und eignen sich somit hervorragend zum Austausch und Speichern von Daten. Sie setzen sich aus einzelnen Textelementen, den Markup-Tags[12], gekennzeichnet durch < und >, zusammen. Diese Tags können wiederum weitere Tags oder Zeichenketten beinhalten und mit Wert-Attributen versehen werden. Das Last.fm-*API* (Application Programming Interface)[13], das in der Diplomarbeit noch verwendet wird, liefert sämtliche Anfrageergebnisse im XML-Format. Eine Anfrage zum Künstler „Korn“ liefert zum Beispiel folgende (stark gekürzte) XML-Datei:

```
<?xml version="1.0" encoding="utf-8"?>
<artist>
  <name>Korn</name>
  <mbid>ac865b2e-bba8-4f5a-8756-dd40d5e39f46</mbid>
  <url>http://www.last.fm/music/Korn</url>
  <tags>
    <tag>
      <name>alternative metal</name>
      <url>http://www.last.fm/tag/alternative%20metal</url>
    </tag>
  </tags>
</artist>
```

Listing 1: Beispiel-XML-Dokument von last.fm für ‘Korn’

Ein XML-Dokument besitzt genau ein Wurzelelement (in diesem Fall <artist>), das direkt nach den Deklarationen zu Version, Entitäten[14] oder der Enkodierung steht.

[12] Nicht zu verwechseln mit *Tags* im Sinne der Verschlagwortung.
[13] http://last.fm/api
[14] Definierbare Kürzel der Form <!ENTITY [%] Name [SYSTEM|PUBLIC] "Wert"[...] >.

Alle Tags können eine beliebige Bezeichnung erhalten und beliebig tief ineinander verschachtelt werden. Um die genaue Struktur eines solchen XML-Dokuments zu beschreiben oder einzuschränken, bedient man sich sogenannter Schemasprachen. Die zwei bekanntesten hierbei sind *Document Type Definition* (DTD) und *XML Schema Definition* (XSD). DTD beschreibt die strukturellen und syntaktischen Einschränkungen eines XML-Dokuments. So lässt sich mit einer DTD zum Beispiel festlegen, das ein <tag>-Element immer zwei Kindelemente <name> und <url> besitzen muss, die nicht leer sein dürfen. Eine DTD ist selbst kein XML-Dokument, weshalb unter anderem XSD entworfen wurde, das zusätzlich zur Definition von Strukturen in XML selbst ebenfalls ein XML-Dokument ist und auch weitere Vorteile wie zum Beispiel die Einführung von Namensräumen oder vordefinierten und komplexeren Datentypen bietet. Die Namensräume werden dazu benutzt um in einem einzelnen XML-Dokument unterschiedliche XML-Sprachen mischen zu können, diese dabei aber sauber voneinander zu trennen. Es gibt für XML viele Erweiterungen wie *XML Path Language* (XPATH) oder *eXtensible Stylesheet Language Transformation* (XSLT), mit denen es möglich ist, XML-Daten zu transformieren oder gezielt nach Informationen in XML zu suchen.

Um die Grundprinzipien des Semantic Web zu erfüllen reicht XML alleine jedoch nicht aus. Die Bezeichner der einzelnen Tags, die in XML verwendet werden, sind frei wählbar und es erschließt sich kein semantischer Zusammenhang zwischen einzelnen XML-Elementen. XML dient jedoch als weitverbreitetes Datenformat für RDF oder OWL, auf die in Kapitel 2.3, ab Seite 10, und Kapitel 2.4, ab Seite 15, noch weiter eingegangen wird, und ist somit eines der wichtigsten Datenformate des Semantic Web.

Ähnlich wie XML ist *JavaScript Object Notation* (JSON) ein kompaktes, textbasiertes Datenformat, das sowohl menschen- als auch maschinenlesbar ist. Im Gegensatz zu XML ist JSON aber keine Markupsprache sondern lediglich ein Datenaus-

tauschformat. Die Daten werden in JSON kompakter kodiert als in XML, was einen geringeren Overhead, also eine höhere Nutzdatengröße, mit sich bringt. Die JSON-Repräsentation zum XML-Dokument aus Listing 1 sieht wie folgt aus:

```
{ "artist": {
    "name":"Korn",
    "mbid":"ac865b2e-bba8-4f5a-8756-dd40d5e39f46",
    "url":"http:\/\/www.last.fm\/music\/Korn",
    "tags": {
      "tag": {
        "name":"alternative metal",
        "url":"http:\/\/www.last.fm\/tag\/alternative%20metal"
      }
    }
  }
}
```

Listing 2: Beispiel-JSON-Dokument

Jede JSON-Datei beginnt mit einem Objekt, gekennzeichnet durch `{` und `}`. Dieses Objekt enthält eine kommaseparierte Liste an Eigenschaften, nämlich Schlüssel- und Wertpaare, getrennt durch einen Doppelpunkt, wobei der Wert selbst auch wieder ein Objekt sein kann. So lassen sich auch in JSON beliebig tief geschachtelte Datenhierarchien abbilden.

Wie der Name schon vermuten lässt, ist JSON eine Teilmenge von JavaScript, wo es zur Serialisierung von Datenobjekten dient. So lässt sich ein JSON-Dokument mittels der JavaScript-Funktion `eval()` direkt in ein JavaScript-Objekt konvertieren. Dadurch und durch die geringere Komplexität im Vergleich zu XML hat sich JSON gerade für Client-/Serverkommunikation in AJAX-Applikationen etabliert. So lässt sich serverseitig generierter JSON-Code nach der Übermittlung an den Client direkt mit JavaScript weiterverarbeiten, ohne dafür komplizierte oder rechenintensive Aufgaben erledigen zu müssen. Außerdem ist JSON nicht nur auf die Nutzung in JavaScript

beschränkt, denn mittlerweile gibt es in allen gängigen Programmiersprachen Parser und Code-Generatoren[15] mit denen sich JSON weiterverarbeiten oder gar aus XML erstellen lässt[16]. In *PHP* (PHP: Hypertext Preprocessor) lässt sich so seit Version 5.2 eine PHP-Variable direkt mit der Funktionen `json_encode()` in ein JSON-Objekt konvertieren[17].

XML und JSON eignen sich somit hervorragend dazu Wissen und Informationen zu transportieren und interoperabel zu gestalten. JSON legt dabei den Fokus eher auf den reinen Transport der Daten, während XML durch Konzepte wie Namensräume und andere Erweiterungsmöglichkeiten auch als Grundlage für andere Technologien, wie zum Beispiel RDF, dient.

2.3 Ressource Description Framework (RDF)

Ein RDF-Dokument dient zur Beschreibung von gerichteten Graphen, also einer bestimmten endlichen Menge von Knoten, die mit gerichteten Kanten verbunden sind. Diese Knoten und Kanten sind mit eindeutigen Bezeichnern versehen. Die zugrundeliegende Struktur eines RDF-Ausdrucks ist eine Sammlung von Tripeln. Dieses sogenannte RDF-Tripel basiert auf dem linguistischen Prinzip *Subjekt-Verb-Objekt*. Eines oder mehrere dieser Tripel bilden den RDF-Graphen. Abbildung 2 zeigt einen minimalen RDF-Graphen, bestehend aus einem RDF-Tripel. Das Beispiel lässt sich normalsprachlich wie folgt umschreiben: „Peter Schneider ist Autor von Tagebuch“.

Das Subjekt des Tripels ist in diesem Fall die Ressource `http://example.org/PeterSchneider`, das Prädikat die Ressource `http://example.org/istAutor` und `http://example.org/Tagebuch` das Ob-

[15] http://json.org
[16] Vergleiche Nathan et al. (2007).
[17] Und mit `json_decode()` natürlich auch wieder in die andere Richtung.

Abbildung 2: Ein minimaler RDF-Graph

jekt. Eine Ressource ist in einem RDF-Dokument alles, was durch einen eindeutigen Bezeichner im *URI* (Uniform Resource Identifier)-Format gekennzeichnet ist[18]. Eine Ressource muss nicht zwangsweise im Web unter der angegebenen URI erreichbar sein und kann alles mögliche repräsentieren, so zum Beispiel auch Emailadressen (`mailto:schneider@example.org`), generelle Konzepte wie Namen (`http://xmlns.com/foaf/0.1/name`) oder Titel (`http://purl.org/dc/elements/1.1/title`) und natürlich auch eine *URL* (Uniform Resource Locator) zu einer Webseite. Subjekte und Prädikate eines RDF-Tripels sind immer Ressourcen, das Objekt kann auch ein Literal[19] sein, also eine Zeichenkette, die optional anhand eines definierten Datentyps interpretiert werden kann, zum Beispiel Datumsangaben, Zahlen oder Telefonnummern et cetera.

Abbildung 3, basierend auf den Daten von Jamendo, die von DBTune.org bereitgestellt werden[20], zeigt ein solches Beispiel: der Künstler `http://dbtune.org/jamendo/artist/13` hat den Namen 'Echo lali', hat vier verschiedene Alben produziert und kommt aus der Nähe von `http://sws.geonames.org/2991627/`.

Im Gegensatz zu einem rein hierarchisch aufgebauten XML-Dokument, das im Normalfall eine Baumstruktur repräsentiert, ist die Idee hinter RDF die Darstellung von Beziehungen zwischen unterschiedlichen Ressourcen. Einer der Gründe dafür ist, dass sich einzelne Ressourcen nicht immer in eine Hierarchie einordnen lassen, so zum

[18] Vergleiche Berners-Lee et al. (2005).
[19] Oder auch ein sogenannter 'blank node', ein Hilfsknoten, der wiederum auf andere Knoten verweist.
[20] http://dbtune.org/jamendo/

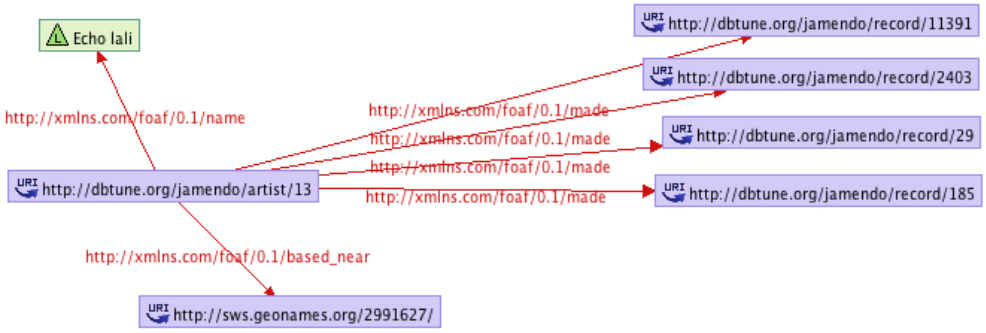

Abbildung 3: RDF-Teilgraph für den Künstler ‘Echo lali’

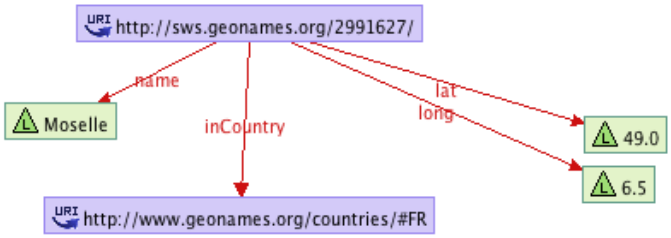

Abbildung 4: RDF-Teilgraph für die Geonames-Ressource ‘2991627’

Beispiel die Ressource `http://xmlns.com/foaf/0.1/name`, die in diesem Fall die Beziehung „hat Namen“ beschreiben soll. Dieses hierarchielose Konzept in RDF hat außerdem den Vorteil, dass sich dadurch Informationen aus unterschiedlichen Quellen kombinieren lassen. Für den Künstler aus Abbildung 3 ist unter anderem die Ressource `http://sws.geonames.org/2991627/` angegeben, eine Anfrage an ebendiese liefert den Teilgraphen aus Abbildung 4.

So lässt sich aus zwei völlig voneinander unabhängigen Quellen anhand der eindeutigen URIs und deren Verknüpfungen untereinander durch implizites Wissen erschließen, dass der Künstler mit dem Namen ‘Echo Lali’ aus ‘Moselle’ in Frankreich[21], Längengrad 6.5 und Breitengrad 49.0, kommt.

[21] http://www.geonames.org/countries/#FR – FR ist das offizielle Länderkürzel für Frankreich nach ISO 3166.

Es gibt mehrere Möglichkeiten, die RDF-Tripel eines Graphen menschen- und maschinenlesbar darzustellen. Eine einfache von Menschen und Maschinen lesbare Syntax ist *Turtle* (Terse RDF Triple Language). Turtle entstand aus *Notation 3* (N3)[22] und bietet im Vergleich dazu eine reduzierte, weniger komplexe Ausdrucksmenge, die aber für die Übermittlung komplexer RDF-Graphen ausreichend ist[23]. Der Graph aus Abbildung 3 sieht in Turtle zum Beispiel wie folgt aus:

```
1 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2 @prefix artist: <http://dbtune.org/jamendo/artist/> .
3
4 artist:13 foaf:name "Echo lali" ;
5           foaf:made <http://dbtune.org/jamendo/record/11391> ;
6           foaf:based_near <http://sws.geonames.org/2991627/> .
```

Listing 3: Turtle-Syntax für RDF-Daten

Ressourcen werden in spitze Klammern gesetzt, Literale in Anführungszeichen und jedes Tripel wird durch einen Punkt abgeschlossen. Die hier verwendete abgekürzte Schreibweise führt die schon aus XML bekannten Namensräume ein, um URIs abzukürzen. Die so abgekürzten URIs werden nicht mehr in spitze Klammern gesetzt, so dass diese nicht mit vollständigen URIs verwechselt werden. Des Weiteren gibt es die Möglichkeit mit dem Einsatz eines Semikolons oder Kommas die Tripel-Schreibweise zu verkürzen. Das Semikolon aus Zeile 4, Listing 3 beendet das Tripel und übergibt in diesen Fall `artist:13` als Subjekt für das nächste Tripel. Mit einem Komma ließe sich auf die gleiche Art und Weise Subjekt *und* Prädikat an das darauf folgende Tripel übergeben.

Die jedoch am weitesten verbreitete Variante in der RDF-Daten ausgeliefert werden ist XML. Dieses Vorgehen hat sich in der Praxis als Standard durchgesetzt, so geben zum

[22] Vergleiche Berners-Lee (2006).
[23] Vergleiche Beckett und Berners-Lee (2008).

Beispiel auch DBTune oder Geonames[24] bei einer Anfrage RDF als XML-Dokument aus. Auch hier werden Namensräume und verkürzte Schreibweisen eingesetzt. Das Listing 3 sieht als XML/RDF-Dokument folgendermaßen aus:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
    <!ENTITY foaf 'http://xmlns.com/foaf/0.1/'>
    <!ENTITY mo 'http://purl.org/ontology/mo/'>
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
    <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
]>
<rdf:RDF xmlns:foaf="&foaf;" xmlns:mo="&mo;" xmlns:rdf="&rdf;" xmlns
    :rdfs="&rdfs;" xmlns:xsd="&xsd;">
  <mo:MusicArtist rdf:about="http://dbtune.org/jamendo/artist/13">
    <rdf:type rdf:resource="&rdfs;Resource"/>
    <foaf:name rdf:datatype="&xsd;string">Echo lali</foaf:name>
    <foaf:made rdf:resource="http://dbtune.org/jamendo/record
        /11391"/>
    <foaf:based_near rdf:resource="http://sws.geonames.org
        /2991627/"/>
  </mo:MusicArtist>
</rdf:RDF>
```

Listing 4: RDF/XML-Syntax-Ausschnitt für den Künstler `http://dbtune.org/jamendo/artist/13`

Das Wurzelelement eines RDF/XML-Dokuments ist ein Knoten vom Typ `rdf:RDF`. Dies hat sich als Standardbezeichnung durchgesetzt. Bedingt durch den hierarchischen Aufbau eines XML-Dokuments müssen auch die Tripel derart kodiert werden, diese werden dabei meist nach Subjekten geordnet, in diesem Fall `mo:MusicArtist`.
`rdf:about` gibt den Bezeichner des Subjekts, also die Ressource an. Bei Prädikat und Objekt geschieht dies über `rdf:resource`. Eine tiefere Schachtelung ist möglich, so

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

ließe sich zum Beispiel für den Knoten `foaf:made` zusätzlich der Name des Albums angeben[25].
Im Vergleich zur Turtle-Syntax können außerdem noch Datentypen für Literale definiert werden (hier: `rdf:datatype="&xsd;string"`). Eine verkürzte Schreibweise für Literale als Attributwerte für Elemente ist aber auch möglich. Die Zeilen 10-12 aus Listing 4 würden sich wie folgt zusammenfassen lassen:

```
1 <mo:MusicArtist rdf:about="http://dbtune.org/jamendo/artist/13" foaf
      :name="Echo lali">
```

Listing 5: RDF/XML-Kurzschreibweise

Die Möglichkeiten der Darstellung in XML sind sehr vielfältig und auf unterschiedliche Art und Weise zu lösen und in jedem Fall ausreichend einen RDF-Graphen komplett zu enkodieren.

2.4 Einfache Ontologien in RDF Schema

Der Begriff Ontologie kommt ursprünglich aus der theoretischen Philosophie und beschäftigt sich mit dem 'Sein' und zugrunde liegenden Strukturen der Realität. In der Informatik dienen Ontologien dazu, Wissensbestände aus unterschiedlichsten Bereichen zu strukturieren. Eine Ontologie besteht damit aus Taxonomien, also Klassifikationen, und Relationen, die bestimmte Begriffe oder Dinge in Gruppen sortieren und diese miteinander in Beziehungen setzen.
Wie in Kapitel 2.3, ab Seite 10, gezeigt können mit RDF bestimmte Aussagen zu unterschiedlichen Ressourcen gemacht werden. Individuen lassen sich untereinander verknüpfen und können unterschiedlichen simplen Datentypen zugeordnet werden. Allerdings sieht RDF dies nicht für Gruppen von Ressourcen vor. So wäre es zum

[25] Die verwendeten Präfixe `foaf` und `mo` aus Listing 4 stehen im Zusammenhang mit den Ontologien *FOAF* und *MusicOntology* auf die in Kapitel 3.3, ab Seite 34, genauer eingegangen wird.

Beispiel wünschenswert nicht nur für einzelne Individuen, sondern für ganze Klassen von Individuen Beziehungen einzuführen.

Mit Hilfe von *RDF Schema* (RDFS) lassen sich solche zusätzlichen Informationen, das nach Hitzler et al. (2007, Kapitel 3.4) „sogenannte terminologische Wissen oder auch Schemawissen“, über ein Vokabular definieren. Gruppen von Ressourcen nennt man Klassen, einzelne Ressourcen sind die sogenannten Instanzen oder Individuen einer Klasse. Klassen können ineinander verschachtelt und kombiniert werden, ein Beispiel zeigt Listing 6[26]:

```
1 ex:Waldi rdf:type ex:Dackel .
2 ex:Dackel rdf:type rdfs:Class .
3 ex:Dackel rdfs:subClassOf ex:Hund .
```

Listing 6: RDFS-Beispiel

Die Ressource `ex:Waldi`[27] ist vom Typ `ex:Dackel`. Aus dieser Anweisung ist jedoch noch nicht klar, ob `ex:Dackel` ebenfalls ein einzelnes Objekt oder eine Klasse ist, weshalb diese URI in der zweiten Zeile von Listing 6 als RDFS-Klasse spezifiziert wird. `ex:Dackel` selbst ist eine Unterklasse von `ex:Hund`. Diese Unterklassenbeziehung ist transitiv und reflexiv, es lässt sich somit schlussfolgern, dass `ex:Waldi` ebenfalls der Klasse `ex:Hund` zugeordnet ist. Durch die Reflexivität lassen sich mehrere Klassen auch gleichsetzen, so dass klar wird, dass diese die gleichen Individuen enthalten:

```
1 ex:Dog rdfs:subClassOf ex:Hund .
2 ex:Hund rdfs:subClassOf ex:Dog .
```

Listing 7: Reflexivität von Klassen in RDFS

In RDFS ist es des Weiteren möglich für die Prädikate eines RDF-Tripels, die sich semantisch meist schwer einer Klasse oder einem Individuum zuordnen lassen, soge-

[26] Der Einfachheit halber werden in den folgenden Beispielen Kopfdaten wie Prefixe und Versionsdefinitionen nicht aufgeführt.

[27] `ex:` steht für einen Beispielnamensraum wie http://example.org/.

nannte *Propertys* anzugeben. Propertys können, wie auch Klassen, Unterpropertys enthalten, die mit `rdf:subPropertyOf` definiert werden:

```
1 ex:Waldi ex:istLieblingsHundVon ex:Peter .
2 ex:istHundVon rdf:type rdf:Property .
3 ex:istLieblingsHundVon rdf:subPropertyOf ex:istHundVon .
4 ex:istLieblingsHundVon rdfs:domain ex:Hund .
5 ex:istLieblingsHundVon rdfs:range ex:Herrchen .
```

Listing 8: RDFS-Property-Beispiel

Mit `rdfs:domain` lässt sich die Zuweisung zu Subjekten festlegen und mit `rdfs:range` die Zuweisung der Propertys zu Objekten. In Listing 8 legt diese Einschränkung des Wertebereichs fest, dass das Subjekt eines Tripels mit dem Prädikat `ex:istLieblingsHundVon` der Klasse `ex:Hund` und das Objekt der Klasse `ex:Herrchen` zugeordnet sein muss um die Sinnhaftigkeit der Aussage zu gewährleisten.

RDFS bietet also die Möglichkeit einfache Ontologien zu modellieren und darzustellen. Um komplexere Klassenbeziehungen und Zusammenhänge darzustellen reicht diese Sprache jedoch nicht aus. Das Modellieren von Formulierungen wie zum Beispiel „Waldi ist der einzige Hund von Peter“ oder „Ein Musikalbum hat mindestens einen Titel“ bedarf der Prädikatenlogik erster Stufe, auf der die Ontologiesprache OWL basiert.

2.5 Ontologien in OWL

OWL unterscheidet sich in drei verschiedene Teilsprachen: *OWL Full*, *OWL DL*[28] und *OWL Lite*. Letztere wiederum ist eine Teilsprache von OWL DL und OWL Full mit einer stark reduzierten Ausdrucksstärke, die vor allem zur Entwicklung einfacher

[28] DL steht für Description Logic.

Taxonomien und leicht ableitbarer Ontologien verwendet wird. OWL DL enthält OWL Lite und ist selbst Teilsprache von OWL Full und wie auch OWL Lite entscheidbar. OWL DL ist die am weitesten verbreitete und unterstützte Teilsprache von OWL. Die ausdrucksstärkste der drei Teilsprachen und ist OWL Full und durch ihre Komplexität unentscheidbar.
Es gilt: OWL Lite $\subset$ OWL DL $\subset$ OWL Full.

Klassen und Propertys, die schon in RDFS vorhanden sind, bilden auch in OWL die Grundbestandteile der Ontologie. In OWL gibt es jedoch eine Vielzahl an weiteren vordefinierten Beziehungen. Die Syntax von OWL lässt sich wie auch schon bei RDF in unterschiedlichen Formaten wie XML/RDF oder Turtle darstellen. Die folgenden Beispiele sind alle in Turtle-Syntax verfasst[29].

Klassen werden in OWL mit `owl:Class` definiert[30], zur Unterklassendefinition kommt das bereits bekannte `rdfs:subClassOf` zum Einsatz. Klassen können zusätzlich mit `owl:equivalentClass` und `owl:disjointClass` als gleich oder disjunkt zueinander deklariert werden. Des Weiteren gibt es in OWL zwei vordefinierte Klassen: `owl:Thing` und `owl:Nothing`. Erstere ist die allgemeinste Klasse in OWL die alle Individuen enthält und somit die Oberklasse aller Klassen ist, letztere ist eine Unterklasse aller Klassen, die keine Individuen enthält.

```
1 ex:Herrchen rdf:type owl:Class .
2 ex:Hund rdf:type owl:Class .
3 ex:Hund owl:disjointClass ex:Herrchen .
4 ex:Dog owl:equivalentClass ex:Hund .
```

Listing 9: Äquivalente und disjunkte Klassen in OWL

Aus dem Beispiel aus Listing 9 lässt sich über diese einfachen Klassenbeziehungen aus Inferenz schlussfolgern, dass `ex:Herrchen` und `ex:Dog` disjunkte Klassen sind.

[29] Ein OWL Syntax Converter ist zu finden unter http://owl.cs.manchester.ac.uk/converter/.
[30] Der Namesraum von OWL ist http://www.w3.org/2002/07/owl#.

Schlussfolgerungen in OWL basieren auf der *Open-World-Assumption*, das heißt, es wird davon ausgegangen, dass Aussagen, die nicht direkt aus einer Ontologie abgeleitet werden können, nicht unbedingt falsch sein müssen[31]. Klassen können mit Hilfe logischer Konstruktoren wie Konjunktion, Disjunktion und Komplement zu komplexeren Klassen verbunden werden. Die Sprachelemente dafür sind: `owl:intersectionOf`, `owl:unionOf` und `owl:complementOf`.

In OWL gibt es, wie auch in RDFS, die Möglichkeit, für Prädikate, die auch Rollen genannt werden, Einschränkungen oder Eigenschaften und Beziehungen wie die schon genannten `rdfs:range` und `rdfs:domain` zu vergeben. Zusätzlich zu diesen lassen sich zum Beispiel mit dem Sprachelement `owl:someElementsFrom` Beziehungen wie „hat mindestens ein“ oder mit `owl:cardinality` die genaue Anzahl von Unterklassen oder Ressourcen angeben. Rollenbeziehungen sind mit den Sprachelementen `rdfs:subPropertyOf` oder auch `rdfs:inverseOf` realisierbar.

```
1  ex:Herrchen owl:equivalentClass [
2    rdf:type            owl:Class ;
3    owl:intersectionOf  ( ex:Mann ex:Hundebesitzer )
4  ] .
5  ex:Hund owl:equivalentClass  [
6    rdf:type            owl:Restriction ;
7    owl:onProperty      ex:istHundVon ;
8    owl:someValuesFrom  ex:Herrchen
9  ] .
10 ex:istHundVon owl:inverseOf ex:istHerrchenVon .
```

Listing 10: Ontologie-Beispiel mit komplexen Klassen und Rollenrestriktionen

Die Ontologie aus Listing 10 lässt sich in natürlichsprachlich wie folgt formulieren: Ein Herrchen ist ein Mann und gleichzeitig ein Hundebesitzer. Ein Hund hat mindestens

[31] Aus der Aussage „Waldi ist ein Hund von Peter.“ lässt sich nach der Open-World-Assumption nicht schlussfolgern, ob Bello auch ein Hund von Peter ist, es kann aber nicht gesagt werden, dass Bello *kein* Hund von Peter ist.

ein Herrchen und ‘ist Herrchen von’ ist das Gegenstück zu ‘ist Hund von’.

OWL bietet des Weiteren auch vordefinierte Sprachelemente an, um einzelne Individuen in einer Ontologie miteinander in Verbindung zu setzen. So lassen sich nicht nur ganze Klassen, sondern auch Instanzen einer Klasse mit `owl:sameAs` gleichsetzen oder mit `owl:differentFrom` voneinander abgrenzen.

OWL macht es also möglich komplexe Ontologiestrukturen zu formulieren, die sowohl maschinen- als auch menschenlesbar sind. Die Komplexität lässt sich durch die drei Teilsprachen von OWL im Vorfeld festlegen, wobei OWL DL in der Anwendung und somit auch in dieser Arbeit die größte Bedeutung zukommt.

2.6 Die Anfragesprache SPARQL

Mit den vorgestellten Technologien RDF, RDF-Schema und OWL ist es möglich, Informationen maschinenlesbar zu machen. Um nun gezielt bestimmte Informationen aus vorhandenen RDF-Graphen extrahieren zu können, bedarf es noch einer Anfragesprache. SPARQL, eine W3C-Recommendation[32], erfüllt eben diesen Zweck. SPARQL ist nur als Anfragesprache für RDF gedacht und unterstützt weder RDF-Schema noch OWL. Eine beispielhafte SPARQL-Anfrage zeigt Listing 11:

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 SELECT ?name ?records
3 WHERE {
4   <http://dbtune.org/jamendo/artist/13> foaf:name ?name ;
5                                         foaf:made ?records .
6 }
```

Listing 11: SPARQL-Anfrage für den Künstler `http://dbtune.org/jamendo/artist/13`

[32] Vergleiche Prud’hommeaux und Seaborne (2007).

Die Syntax erinnert an *SQL* (Structured Query Language), jedoch verwendet SPARQL für die eigentlichen Anfragen RDF-Graphen-ähnliche Muster in Turtle-Syntax[33]. Eine Anfrage setzt sich zusammen aus dem `PREFIX-Teil`, in dem, wie auch in RDF, die unterschiedlichen Namensräume definiert werden, dem `SELECT`-Teil, in dem das Ausgabeformat und die Bezeichner für gewünschte Rückgabewerte definiert werden und dem `WHERE`-Statement, in dem die eigentliche Anfrage formuliert wird. Die Bezeichner, sind frei wählbare Variablen[34], die mit `$` oder wie in diesem Fall mit `?` deklariert werden, und werden nach einer erfolgreichen Anfrage mit Werten belegt. Diese Variablen können in der Anfrage sowohl für Subjekt und Prädikat, als auch, für das Objekt eingesetzt werden. Die Anfrage aus Listing 11 sucht also nach allen Namen und Alben, die dem angegebenen Künstler zugeordnet sind und speichert diese in den entsprechenden Variablen. Mit den RDF-Daten aus Abbildung 3 würde diese Anfrage das Ergebnis aus Tabelle 1 liefern:

name	records
“Echo lali”	http://dbtune.org/jamendo/record/11391
“Echo lali”	http://dbtune.org/jamendo/record/185
“Echo lali”	http://dbtune.org/jamendo/record/2403
“Echo lali”	http://dbtune.org/jamendo/record/29

Tabelle 1: Ergebnis für SPARQL-Anfrage aus Listing 11

Die resultierende Ergebnistabelle enthält nur die Bezeichner, die in der `SELECT`-Anweisung angegeben wurden[35], und führt diese, um die tabellenartige Struktur beizubehalten auch mehrfach auf. SPARQL unterstützt weiterhin Gruppierungen, die durch geschweifte Klammen gekennzeichnet sind. Mit den Schlüsselwör-

[33] Vergleiche Listing 3, Kapitel 2.3, ab Seite 10,.
[34] Zeichenketten aus Zahlen, Buchstaben und einigen Sonderzeichen.
[35] Ein Anfrage der Form `SELECT *` liefert in der Ergebnistabelle alle benutzten Variablen.

tern `OPTIONAL` und `UNION` lassen sich diese Gruppierungen mit optionalen oder alternativen Anfragen kombinieren. Die Anfrage aus Listing 12 lässt sich somit wie folgt formulieren: finde maximal zwei voneinander verschiedene Alben von Künstler `http://dbtune.org/jamendo/artist/13`, die entweder mit 'chanson' oder 'polka' getagged sind[36], und falls diese einen Titel haben, so zeige diesen ebenfalls an.

```
1  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2  PREFIX tag: <http://dbtune.org/jamendo/tag/>
3  PREFIX tags: <http://www.holygoat.co.uk/owl/redwood/0.1/tags/>
4  SELECT DISTINCT * WHERE {
5    { <http://dbtune.org/jamendo/artist/13> foaf:made ?records . }
6    { { ?records tags:taggedWithTag tag:polka . }
7        UNION
8      { ?records tags:taggedWithTag tag:chanson . }
9    }
10   OPTIONAL { ?records dc:title ?nameOfAlbum . }
11 } SORT BY ?nameOfAlbum LIMIT 2
```

Listing 12: SPARQL-Anfrage mit optionalen und alternativen Gruppierungen

SPARQL unterstützt, wie auch SQL und andere Anfragesprachen, Parameter wie `LIMIT`, `ORDER` und `DISTINCT` zur Einschränkung oder genauer Ausgabebestimmung der gewünschten Resultate. Mit der zusätzlichen Angabe eines Filters lassen sich die erwünschten Ergebnisse weiter einschränken. `FILTER(?anzahl > 10)` würde nur Suchergebnisse anzeigen, bei denen die Variable `anzahl` Werte größer zehn hat, die Angabe von `FILTER (regex(...))` erlaubt hierbei sogar die Verwendung regulärer Ausdrücke zur Filterung der Ergebnisse.

[36] Kein exklusives oder.

Mit diesen und weiteren Ausgabemöglichkeiten und Operatoren[37] existiert mit SPARQL eine sehr umfangreiche Anfragesprache, die eine leicht zu benutzende Schnittstelle bietet, um bestimmte Informationen aus unterschiedlichen RDF-Graphen zu extrahieren und zu kombinieren.

2.7 Machine Learning in OWL

Machine Learning (maschinelles Lernen) ist ein Teilgebiet der künstlichen Intelligenz. Maschinelles Lernen legt sein Hauptaugenmerk auf die automatische Erkennung von Gesetzmäßigkeiten oder sogenannten *Patterns* aus gegebenen Daten. Das lernende System kann somit auch Daten beurteilen, die ihm unbekannt sind. Ein Bereich des maschinellen Lernens ist das *überwachte* Lernen. Der zugrunde liegende Lernalgorithmus bekommt dabei Ein- und Ausgabepaare vorgegeben, er lernt sozusagen aus vorhandenen Beispielen.
Im konkreten Fall des maschinellen Lernens in einer *Description Logic*, also einer Sprache zur Repräsentation von Wissen, auf der OWL DL und OWL Lite basieren, lässt sich das eigentliche Lernproblem wie folgt beschreiben:

Gegeben sei eine Wissensbasis in Form modellierter Ontologien in OWL. Gegeben seien weiterhin Individuen aus dieser Wissensbasis: positive und negative Beispiele. Ziel ist es nun, einen Ausdruck zu finden, so dass möglichst wenige der negativen und möglichst viele der positiven Beispiele in den durch den Ausdruck beschriebenen Klassen enthalten sind. Der so entstandene Klassenausdruck, meist eine logische Verknüpfung mehrerer in der Ontologie enthaltener Klassen[38], sollte die Eigenschaft haben, auch solche Instanzen korrekt klassifizieren zu können, die nicht Teilmenge der positiven oder negativen Beispiele sind[39]. Eine Anwendung für ein solches Lernproblem stellt

[37] Vergleiche Hitzler et al. (2007, Kapitel 7).
[38] Vergleiche Kapitel 2.5, ab Seite 17, und Motik et al. (2009).
[39] Vergleiche Lehmann und Hitzler (2008) und Lehmann (2009).

zum Beispiel die Suche nach Chemikalien, die Krebs verursachen können, dar[40].

Die verwendeten Algorithmen reichen von Brute-Force- oder Zufallsalgorithmen bis hin zu genetischen oder heuristischen Algorithmen. Zu der letzten Kategorie hört auch der in dieser Arbeit verwendete Algorithmus *Class Expression Learning for Ontology Engineering* (CELOE), auf den in Kapitel 4.4, ab Seite 54, noch näher eingegangen wird.

2.8 Zusammenfassung

Das Semantic Web und die mit ihm verbundenen Technologien machen es möglich, Wissen und Informationen maschinen- und menschenlesbar zu strukturieren und zu übermitteln. Die hierfür entwickelten Standards stellen Methoden zur Speicherung, Strukturierung und Übermittlung dieser Informationen dar. Mit maschinellem Lernen ist es möglich, aus diesem semantisch aufbereiteten Wissen neue Erkenntnisse zu gewinnen.

Die in diesem Kapitel vorgestellten Technologien bilden die Grundbausteine des Semantic Web. Diese und weitere Technologien werden stetig weiterentwickelt, weshalb diese Arbeit nur einen Teil des aktuellen Stands repräsentieren kann[41]. Für eine detailliertere Ausführung der hier aufgeführten Technologien sei auf Hitzler et al. (2007) und Berners-Lee et al. (2001) oder die offiziellen Dokumentationen und Spezifikationen des W3C[42] verwiesen.

[40] Vergleiche dazu http://dl-learner.org/wiki/Carcinogenesis.
[41] Vergleiche http://www.w3.org/TR/owl2-overview/ oder http://www.w3.org/TR/2009/WD-sparql11-query-20091022/.
[42] http://www.w3.org/standards/semanticweb/

3 Entwurf der Webapplikation moosique.net

Der Hauptbestandteil dieser Arbeit ist es eine Webapplikation zu entwickeln, mit der es möglich sein soll, in unterschiedlichen Kategorien nach Musik zu suchen, sich die Ergebnisse direkt im Webbrowser anzeigen zu lassen und vor allem auch dort anzuhören. Eine direkte Anbindung des DL-Learner-Webservices an die Webapplikation, zur Suche und Vorschlagsgenerierung aus gehörten Musikstücken, ist dabei unabdinglich. In diesem Kapitel wird nun auf die unterschiedlichen Aspekte und Probleme des Entwurfs einer derartigen Webapplikation eingegangen.

3.1 Konzepte zeitgemäßer Webapplikationen

3.1.1 Reaktionszeiten

Eines der Hauptziele einer zeitgemäßen Webapplikation sollte es sein, dem Benutzer die Bedienung dieser so einfach und intuitiv wie möglich zu machen. Wie der verwendete Begriff Applikation nahelegt, soll beim Benutzer der Eindruck entstehen, er würde in der Tat eine Desktopanwendung bedienen, und nicht auf eine Webseite zugreifen. Einer der Unterschiede zwischen einer nativen Anwendung und einer Webapplikation ist die Reaktionszeit des Programms.

Um die Reaktionszeit der in dieser Arbeit entstandenen Webapplikation zu verkürzen, wurden für den Frontendbereich modernste Technologien wie *Hyper Text Markup Language* (HTML)[43] in Version 5 zum Aufbau der Grundstruktur der Applikation und *Cascading Style Sheets level 3* (CSS3)[44] zur Gestaltung dieser eingesetzt. Diese Technologien sind zwar selbst noch in der Entwicklung und werden bisher nicht in allen

[43] http://dev.w3.org/html5/spec/Overview.html
[44] http://www.w3.org/Style/CSS/current-work#CSS3

gängigen Browsern vollständig unterstützt, doch die aktuell verfügbaren Versionen der plattformunabhängigen Browserengines Webkit[45] und Gecko[46] implementieren die verwendeten Features jedoch mittlerweile ausreichend. Mit diesen Technologien ist es möglich, den produzierten Code und die damit einhergehende Datengröße erheblich zu reduzieren. So erlaubt es CSS3 zum Beispiel visuelle Elemente und Effekte einzuführen, die sonst nur durch den Einsatz zusätzlicher Grafiken möglich wären.

Durch die Einhaltung des Prinzips der Trennung von Layout, Inhalt und Funktion lassen sich benötigte Serveranfragen und Downloadzeiten auf ein Minimum beschränken. Durch den Einsatz erweiterter Frontendtechniken wie CSS-Sprites[47] oder der Komprimierung und Zusammenfassung der eingebundenen CSS- und JavaScript-Dateien vor der Auslieferung an den Browser können die Seitenaufbauzeiten der Webapplikation weiter optimiert werden, was zu kürzeren Wartezeiten auf Benutzerseite führt.

3.1.2 Interfacedesign

Im Laufe der Arbeit entstanden mehrere Anforderungen an die Webapplikation moosique.net, die sich aus Gründen der Funktionalität und der Benutzerfreundlichkeit ergaben:

1. Suchfunktion:
   Dem Benutzer soll die Möglichkeit gegeben werden, in unterschiedlichen, vordefinierten Kategorien, wie zum Beispiel Künstlernamen oder Songtiteln nach Musik zu suchen. Des Weiteren soll dem Benutzer die Möglichkeit gegeben werden, sich durch die Angabe seines (falls vorhandenen) last.fm-Benutzernamens Vorschläge auf Basis seiner Profildaten anzeigen zu lassen.

[45] http://webkit.org/
[46] https://wiki.mozilla.org/Firefox3/Gecko_Feature_List
[47] Vergleiche Shea (2004).

2. Medienplayer-Funktionalitäten:
   Der Benutzer soll volle Kontrolle über die von ihm hinzugefügten Titel und Alben haben. Die Möglichkeit die Wiedergabe zu pausieren oder abzubrechen, bestimmte Titel zu überspringen oder auszuschließen, sowie die Unterstützung eines kontinuierlichen Abspielens mehrerer Titel nacheinander sind Pflichtkriterien.

3. Vorschlagsgenerierung und Anzeige:
   Dem Benutzer muss die Möglichkeit gegeben werden, die ihm generierten Vorschläge einzusehen und auswählen zu können. Dabei sollen zusätzliche Informationen angezeigt werden, die begründen, warum gerade diese Vorschläge generiert wurden.

Um diese Anforderungen in der Praxis umzusetzen wurde auf bewährte Interfaceelemente wie zum Beispiel Tabs[48] oder die international verständlichen Symbole für 'play' und 'pause' bei Medienplayern zurückgegriffen. Diese Elemente sind von anderen Webseiten und Desktopanwendungen bekannt, so dass die Bedienung der Applikation intuitiv erfolgen sollte und somit auch die Erwartungshaltung des Benutzers erfüllt werden kann[49]. Die Suche und die Bedienelemente zur Steuerung der Wiedergabe sowie Informationen zum laufenden Musikstück bilden dabei zentrale Elemente und werden immer im oberen Bereich angezeigt. Gleiches gilt für das Hauptmenü im rechten oberen Bereich der Seite, welches mit Tabs die Umschaltung des Inhaltsbereiches ermöglicht. Die Abbildungen 5, 6, 7 und 8 zeigen Screenshots der Interfaceumsetzung der finalen Applikation.

[48] Registerkarten
[49] Vergleiche Wirth (2004, Kapitel 5 und Kapitel 12).

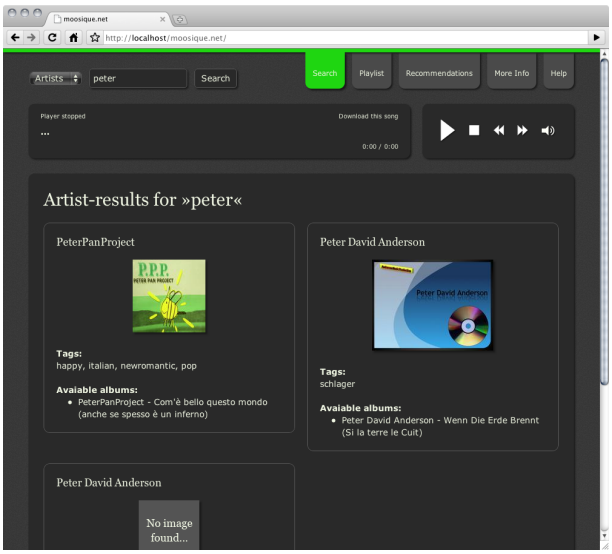

Abbildung 5: Suchergebnissanzeige

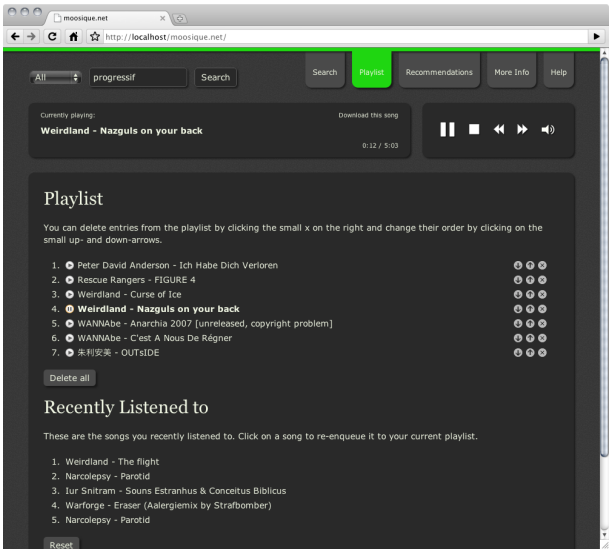

Abbildung 6: Medienplayer und Wiedergabeliste

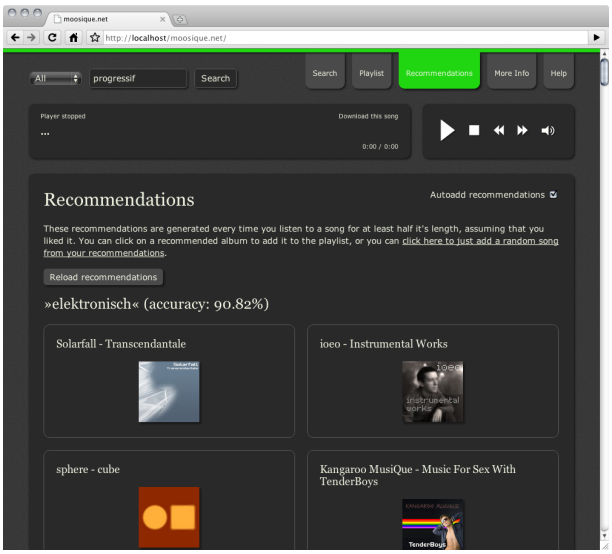

Abbildung 7: Anzeige der generierten Vorschläge

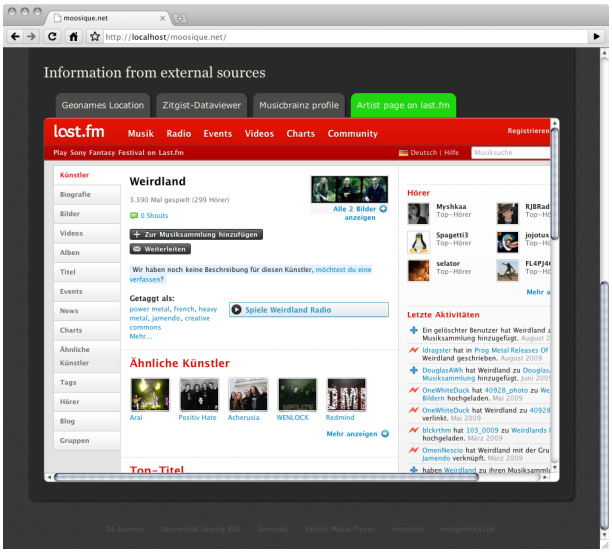

Abbildung 8: Zusätzliche Informationen

3.1.3 JavaScript und AJAX

Um die in Kapitel 3.1.2, ab Seite 26, erarbeiteten Anforderungen zu erfüllen und die Dynamik der Webapplikation erheblich zu verbessern, ist es unumgänglich JavaScript einzusetzen. JavaScript ist ein Dialekt von ECMAScript und somit eine objektbasierte Skriptsprache, die hauptsächlich auf Client-Seite in Webbrowsern dazu eingesetzt wird, den Aufbau und die Gestaltung einer Webseite zur Laufzeit zu verändern. Durch die Manipulierung des sogenannten *Document Object Model* (DOM) mit Hilfe von JavaScript lassen sich die einzelnen Elemente einer statischen HTML-Seite jederzeit ändern. Durch den Einsatz von *DOM-Events* ist es so zum Beispiel möglich, komplette Seitenbereiche durch Nutzeraktionen ausblenden oder neu generieren zu lassen, was in der Webapplikation moosique.net unter anderem für die Funktionalität der Tabs verwendet wird.

Um die Anforderung der kontinuierlichen Wiedergabe einzelner Musikstücke gewährleisten zu können, verwendet die Applikation eine AJAX-Schnittstelle. Damit ist es möglich Anfragen vom Browser aus an den Server zu stellen, ohne dass die eigentliche Webseite neu geladen werden muss. Das Abspielen eines Musikstücks durch den Browser wird dadurch nicht unterbrochen, da die Anfragen sozusagen 'im Hintergrund' asynchron stattfinden. Die Anfrage wird über ein *XMLHttpRequest*-Objekt per JavaScript an den Server übermittelt. Sobald der Server die Anfrage bearbeitet hat, gibt er die Antwort wieder an das Objekt zurück, woraufhin die Antwort weiterverarbeitet werden kann. Die Antwort wird aufgrund der leichteren Verwendung in JavaScript meist als XML oder JSON an den Client übermittelt. Ebenfalls bewährt hat sich die Übermittlung von vorbereiteten, serverseitig generierten HTML-Bausteinen, da diese ohne weiteren Rechenaufwand auf Clientseite direkt in das bestehende DOM eingebunden werden können.

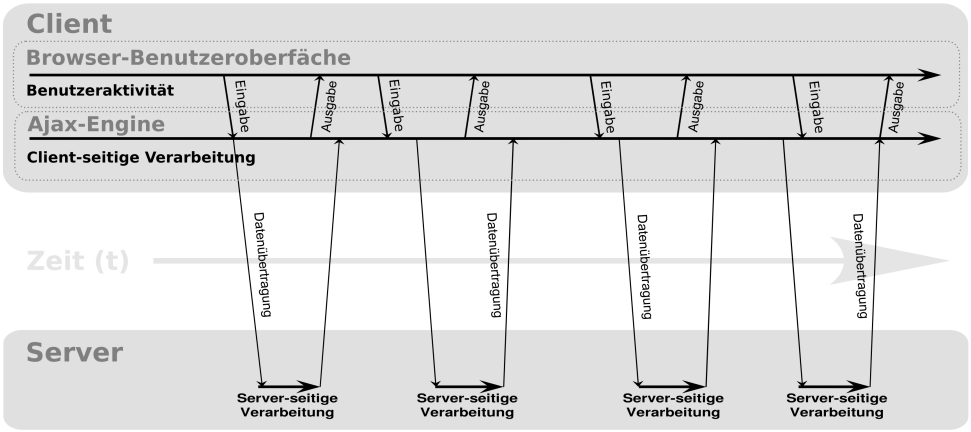

Abbildung 9: Asynchrone Client-/Server-Kommunikation mit AJAX

Das Funktionsprinzip einer AJAX-Applikation wird in Abbildung 9[50] dargestellt. Der Benutzer kann somit jederzeit Suchanfragen oder Anfragen zur Generierung von Vorschlägen durchführen und trotzdem weiterhin Musik hören - ohne ein neues Browserfenster aufmachen zu müssen. Die Vorteile von AJAX liegen damit klar auf der Hand, allerdings hat der Einsatz dieser Technologie auch Nachteile. Benutzer, deren Browser keine JavaScript-Unterstützung bietet, haben keine Möglichkeit eine AJAX-Applikation, sofern diese nur als solche konzipiert und programmiert wurde, zu benutzen. Des Weiteren ist die Umsetzung barrierefreier Applikationen unter dem Einsatz von AJAX nur schwer zu realisieren, da die meisten Screenreader[51] die per AJAX sukzessiv nachgeladenen Inhalte nicht erkennen und wiedergeben können. Da die in dieser Arbeit entstandene Applikation, unter anderem wegen der Möglichkeit Audiodateien direkt im Browser abspielen zu können, jedoch zwingend JavaScript voraussetzt[52], wird eine weitere Untersuchung dieser Problematiken nicht vorgenommen[53].

[50] Quelle: http://upload.wikimedia.org/wikipedia/commons/4/4a/Prozessfluss-ajax.svg

[51] Screenreader sind Programme, die Blinden und Sehbehinderten eine alternative Ausgabe der Bildschirminhalte ermöglichen.

[52] Vergleiche Kapitel 3.2, ab Seite 33,.

[53] Für beide Probleme gibt es dennoch Lösungen und Ansätze, vergleiche dazu Keith (2007) und Craig et al. (2009).

Um die Implementierung der JavaScript- und AJAX-Funktionen zu erleichtern, empfiehlt sich der Einsatz eines *Frameworks*. Das in dieser Arbeit verwendete Framework *mootools*[54] ist ein clientseitiges JavaScript-Framework, das viele der benötigten Funktionen wie objektorientierte Programierung, DOM-Manipulation oder AJAX als Bibliothek vordefinierter Objekte und Klassen bereitstellt[55]. Der Einsatz eines solchen Frameworks bringt viele Vorteile mit sich. Die Implementation einer AJAX-Schnittstelle gestaltet sich in mootools relativ einfach, ein Beispiel sei in Listing 13 gegeben:

```
var request = new Request({
  method: 'get', url: 'request.php',
  onSuccess: function(response) {
    $('antwort').set('text', response);
  },
  onFailure: function() {
    $('antwort').set('text', 'Fehler passieren...');
  }
}).send('anfrage=mehrInfo');
```

Listing 13: AJAX-Request mit mootools

Es wird ein neues `Request`-Objekt erstellt und mit den Parametern `method` und `url` wird der Anfragetyp und die angesprochene URL definiert. Bei einer erfolgreichen Anfrage mit dem Get-Paramenter `anfrage=mehrInfo` wird die Antwort des Servers direkt an das HTML-Element `antwort` übergeben, bei einem Fehler wird der Text 'Fehler passieren...' angezeigt. Einer der größten Vorteile bei der Verwendung dieser Methode ist, dass sie browserunabhängig funktioniert. Die unterschiedlichen Implementierungen der Browser für das *XMLHttpRequest*-Objekt[56] spielen hier keine Rolle, da das Framework diese Unterscheidung übernimmt.

[54] http://mootools.net
[55] Vergleiche Proietti et al. (2009) und Newton (2008).
[56] Vergleiche Garrett (2005).

Viele der häufig in JavaScript verwendeten Funktionen werden durch den Einsatz eines Frameworks erleichtert. Sowohl die Wartbarkeit als auch die Erweiterbarkeit des entstehenden Quellcodes können, auch gerade durch das objektorientierte Klassenprinzip, das in mootools sehr gut umgesetzt wird, erheblich verbessert werden.

Ein genauer Vergleich verschiedener client- und serverseitiger Frameworks würde den Rahmen dieser Arbeit sprengen. Für eine Liste der verbreitesten clientseitigen JavaScript-Frameworks und ein Vergleich der Funktionalitäten sei auf Schütz (2009) verwiesen.

3.2 Audiowiedergabe in aktuellen Browsern

Ein weiterer Bestandteil der Entwurfsphase war die Auswahl einer geeigneten Methode, das Abspielen von Audiodateien im Browser zu ermöglichen[57]. Dabei gibt es mehrere Ansätze, um in einem modernen Webbrowser Audiodateien abzuspielen. Der einfachste Ansatz ist, die gewünschte Audiodatei direkt mit den HTML-Elementen `<embed>` oder `<object>` einzubinden. Dies erweist sich in der Praxis und für diese Arbeit jedoch nicht als sinnvoll, da je nach Betriebssystem und Browser (und eventuell installierten Plugins) eine Wiedergabe nicht gewährleistet werden kann. Des Weiteren fehlt hierbei die Möglichkeit Abspiellisten anzulegen.

Die Wahl fiel auf den *Yahoo Media Player*, der kostenfrei von Yahoo! bereitgestellt wird[58]. Der Yahoo Media Player wird per JavaScript in den Browser eingebunden und auf vielen Plattformen unterstützt[59]. Er bietet eine umfangreiche API, mit der sich

[57] Die auf Jamendo hinterlegte Musik ist im weit verbreiteten Audioformat *MP3* (MPEG-1 Audio Layer 3) und im quelloffenen Ogg-Vorbis-Format verfügbar.
[58] http://mediaplayer.yahoo.com/
[59] Vergleiche http://developer.yahoo.com/yui/articles/gbs/.

die Wiedergabe der Musikstücke beeinflussen lässt. Das API lässt sich per JavaScript ansprechen. So ist es möglich, sämtliche Playerfunktionen zur Laufzeit zu steuern und neue Musikstücke zu einer vorhandenen Abspielliste hinzuzufügen. Die Installation eines zusätzlichen Browser-Plugins wie Flash[60] oder Quicktime[61] ist bei diesem Player im Vergleich zu anderen frei verfügbaren Playern[62] also nicht nötig.

Die gewünschte Methode wäre die Verwendung des HTML5-Elements `<audio>`, jedoch wird diese native Form der Einbindung von Audiodateien zumindest bisher nicht ausreichend in den gängigen Browsern implementiert[63] und ist damit für diese Webapplikation nicht geeignet.

3.3 Verwendete Ontologien

In Kapitel 2, ab Seite 4, wurden in den verwendeten Beispielen Namensräume für Ontologien verwendet, die in dieser Arbeit benutzt werden. Die wichtigste hierbei verwendete Ontologie ist die *Music Ontology* von Raimond et al. (2007). Diese Ontologie stellt ein großes Vokabular an Konzepten und Eigenschaften zur Beschreibung musikrelevanter Daten bereit. So lassen sich von Künstlern und Komponisten, bis hin zu deren Werken und Auftritten sogar Konzepte wie Veröffentlichungsdatum eines Tonträgers, Signalqualität der Aufnahmen oder Songtexte bestimmter Titel beschreiben[64]. Die in dieser Arbeit und auch von DBTune verwendeten Konzepte der Music Ontology für die verwendeten RDF-Daten sollen zum besseren Verständnis kurz vorgestellt werden:

[60] http://www.adobe.com/se/products/flashplayer/
[61] http://www.apple.com/quicktime/download/
[62] Zum Beispiel der JW FLV Player (http://longtailvideo.com) oder der XSPF Web Music Player (http://musicplayer.sourceforge.net).
[63] Vergleiche http://html5doctor.com/native-audio-in-the-browser/.
[64] Für eine komplette Spezifikation der Music Ontology sei auf Giasson und Raimond (2007) verwiesen.

- `mo:MusicArtist`: ein Künstler, Sänger, Künstlergruppe et cetera
- `mo:Record`: ein veröffentlichtes Album eines Künstlers
- `mo:Track`: die einzelnen Titel auf einem veröffentlichen Album
- `mo:available_as`: die Verfügbarkeit eines Albums oder Titels als Medienquelle; als Ressource ist ein entsprechender Downloadlink hinterlegt
- `mo:image`: das Titelbild eines Albums
- `mo:track_number`: Nummer eines Titels auf dem verknüpften Album
- `mo:license`: Lizenz eines bestimmten Titels[65]
- `mo:published_as`: Veröffentlichungsart eines Titels

Die Music Ontology verwendet zusätzlich noch die Ontologien *FOAF* (Friend of a Friend)[66], *DC* (Dublin Core)[67] und die *Tag Ontology*[68]. Diese Ontologien stellen allgemeine Konzepte und Metadaten wie zum Beispiel 'Name' oder 'in der Nähe von' bereit. Dabei sind sie so einfach wie möglich gehalten, um die Verwendung in unterschiedlichen Arten von Dokumenten zu ermöglichen. Die verwendeten Bezeichner und deren konkrete semantische Bedeutung in den von DBTune vorliegenden Daten und damit auch in dieser Arbeit sind:

- `dc:title`: Beschreibung des Titels eines Albums
- `tags:taggedWithTag`: Schlagworte, die für ein bestimmtes Album vergeben wurden
- `foaf:made` und `foaf:maker` Beschreibung der Relation zwischen veröffentlichen Alben und Künstlern
- `foaf:name`: der Künstlername

[65] Zum Beispiel http://creativecommons.org/licenses/by-nc-sa/3.0/.
[66] Vergleiche Brickley und Miller (2007).
[67] Vergleiche Newman (2005).
[68] Vergleiche Kokkelink und Schwänzl (2002).

- `foaf:img`: ein Bild des Künstlers
- `foaf:homepage`: die Homepage oder der Webauftritt des Künstlers
- `foaf:based_near`: Herkunft des Künstlers[69]

Diese Klassen und Relationen bilden die Basis der RDF-Daten von DBTune und werden in den implementierten SPARQL-Anfragen verwendet, auf die in Kapitel 4.3, ab Seite 50, näher eingegangen wird.

3.4 Das Lernproblem

Die von einem Benutzer gehörten Musikstücke werden als positive Beispiele betrachtet. Ein Musikstück gilt dabei als gehört, wenn der Benutzer mindestens die Hälfte der Spieldauer erreicht hat, ohne das Abspielen vorher abzubrechen. Diese Methode wird auch beim bei last.fm eingesetzt und hat sich bewährt. Die aktuelle Abspielzeit eines gerade gehörten Musikstücks lässt sich mit dem API des verwendeten Yahoo Media Players jederzeit bestimmen, weshalb dieses Verfahren in der Webapplikation umgesetzt wird. Insgesamt werden bis zu maximal zehn positive Beispiel in Betracht gezogen, die von der Webapplikation in einem Cookie gespeichert werden.

Die Grundannahme ist dabei die, dass wenn ein Benutzer ein Musikstück für länger als 50% seiner Abspieldauer gehört hat, er dieses danach auch nicht abbrechen, sondern zu Ende anhören wird, und somit indirekt als ‘gut’ bewertet.
Eine Aussage über negative Beispiele kann hierbei jedoch nicht gemacht werden, da nicht angenommen werden kann, dass ein Benutzer ein Musikstück als negativ bewertet, wenn er das Abspielen vorzeitig abbricht. Dies könnte auch andere Gründe haben, wie zum Beispiel schlechte Enkodierungsqualität oder stückhaft unterbrochene Wiedergabe wegen eines langsamen Streamingservers.

[69] Eine eindeutige Ressource, über die weitere Informationen hinterlegt sein können.

Durch dieses Verfahren schränkt sich das in Kapitel 2.7, ab Seite 23, vorgestellte Lernproblem auf 'nur positive Beispiele' ein, das Lernproblem der Webapplikation lässt sich somit vereinfacht formulieren als: „Finde einen Klassenausdruck, so dass möglichst alle positiven Beispiele in den durch den Ausdruck beschriebenen Klassen enthalten sind."

3.5 Das Machine-Learning-Tool DL-Learner

Zur Anbindung des SPARQL-Endpoints von DBTune an die Webapplikation moosique.net dient das OpenSource-Tool *DL-Learner* der *Research Group for Agile Knowledge Engineering and Semantic Web*. Der DL-Learner ist ein frei verfügbares Framework zum überwachten maschinellen Lernen in OWL und Beschreibungslogiken[70]. Er bietet mehrere unterschiedliche Komponenten zur Einbindung von sogenannten *Knowledge Sources* (Wissensbasen), verschiedenen *Reasonern* und Algorithmen zum überwachten Lernen.

Knowledge Sources können als Ontologie in OWL, als KB-Datei[71] oder als SPARQL-Endpoint angegeben werden, was die Einbindung externer Wissensquellen erheblich erleichtert. Die in dieser Arbeit verwendeten RDF-Daten von Jamendo können somit direkt über den SPARQL-Endpoint von DBTune[72] bereitgestellt werden.

Das in Kapitel 3.4, ab Seite 36, vorgestellte, dieser Arbeit zugrunde liegende Lernproblem entspricht dem von DL-Learner unterstützten Lernproblem `posOnlyLP`. Der DL-Learner bietet für die unterschiedlichen Lernprobleme mehrere Algorithmen an, für das hier behandelte Lernproblem kommen nur die Algorithmen *Brute Force*, *Random Guesser* und *CELOE* in Frage. Der Algorithmus CELOE liefert hier, im Vergleich

[70] Vergleiche Lehmann (2009).
[71] Ein internes DL-Learner-Format zur Repräsentation von Wissen.
[72] http://dbtune.org/jamendo/sparql

zu den beiden anderen relativ einfachen Algorithmen, die besten Ergebnisse, weshalb auch nur dieser eingesetzt wird.

Als Reasoner, eine Softwarekomponente, die zusätzliches Wissen durch logisches Schlussfolgern oder Inferenz ableiten kann, stehen ebenfalls mehrere Komponenten im DL-Learner bereit. Zur Implementierung wird der im DL-Learner als Standard implementierte *Fast Instance Checker* verwendet, da dieser am zügigsten arbeitet, was eine Vorraussetzung nach Kapitel 3.1.1, ab Seite 25, ist.

Der DL-Learner bietet außerdem die Möglichkeit, SPARQL-Anfragen an definierte Wissensbasen zu senden, und sich diese als XML oder JSON ausliefern zu lassen. Die dabei erhaltenen Daten lassen sich in einem Cache zwischenspeichern, so dass eine erneute Anfrage gleichen Inhalts sehr schnell Resultate liefert. Diese Komponente wird hauptsächlich für die Suchfunktion der Webapplikation eingesetzt. Durch den Einsatz des Caches lassen sich so häufig gestellte Suchanfragen, zum Beispiel zu den meistverwendeten Tags, direkt beantworten, ohne dass der Benutzer lange auf Ergebnisse warten muss.

Die Anbindung der genannten Komponenten an die eigentliche PHP-Webapplikation wird über den Webservice des DL-Learners vorgenommen. Dieser basiert auf *JAX-WS* (Java API for XML Web Services) und setzt Java in Version 6 voraus. Mit diesem Webservice ist es möglich über eine SOAP-Schnittstelle, die auch in PHP unterstützt wird[73], die vom Webservice bereitgestellten Funktionen für SPARQL-Anfragen oder Lernalgorithmen zu nutzen. Für eine komplette Funktionsübersicht des Webservice sei auf die Dokumentation des DL-Learners verwiesen[74].

[73] PHP muss dafür mit `-enable-soap` compiliert werden.
[74] http://dl-learner.org/javadoc/org/dllearner/server/DLLearnerWS.html

3.6 Zusammenfassung

Die in diesem Kapitel vorgestellten Technologien und Konzepte bilden die Grundlage der zu implementierenden Webapplikation. Der Fokus wurde dabei auf eine möglichst einfache Bedienung der Webapplikation durch den Benutzer gelegt. Die vorgestellten Ontologien formieren die Basis für die verwendeten RDF-Daten die zur Suche und der Generierung von Vorschlägen hinzugezogen werden. Zur eigentlichen Berechnung der Vorschläge kommt der DL-Learner zum Einsatz, der eine weitreichende Vielfalt an Algorithmen Komponenten bietet, um Wissen aus diesen Daten abzuleiten.

4 Implementation

In diesem Kapitel wird der generelle Aufbau der während der Arbeit entstandenen Webapplikation beschrieben. Weiterhin wird erläutert, wie aus den vorliegenden Daten eine Ontologie erstellt wurde, so dass diese später vom Lernalgorithmus CELOE verwendet werden kann. Dazu war eine Evaluation der Daten notwendig, die in Kapitel 4.2, ab Seite 42, beschrieben wird. Zu dem wird auf die Implementation der SPARQL-Schnittstelle zur Suche und Extraktion zusätzlicher Informationen eingegangen, und welche Aspekte dabei beachtet werden mussten. Der Algorithmus CELOE musste, um ausreichende Resultate liefern zu können, entsprechend konfiguriert werden, was in Kapitel 4.4, ab Seite 54, näher erläutert wird.

4.1 Aufbau der Webapplikation

Die im Verlauf der Arbeit entwickelte Webapplikation setzt sich aus mehreren Bestandteilen zusammen. Diese lassen sich in vier Bereiche unterscheiden: das clientseitige Frontend, die serverseitige PHP-Applikation, der DL-Learner und externe Wissensquellen.

Das Frontend ist der Bereich in dem der Benutzer agiert. Von dort aus werden sämtliche Anfragen über die in JavaScript implementierte AJAX-Schnittstelle an die PHP-Applikation weitergegeben. Dies betrifft sowohl die Such- als auch die Lernanfragen. Auch Playlist-Anfragen aufgrund des Hinzufügens von Alben oder Musikstücken sowie die Anzeige der zusätzlichen Informationen werden mittels AJAX an die PHP-Applikation gesendet. Die Liste der positiven Beispiele, also die gehörten Alben, werden in einem Cookie gespeichert, so dass sowohl die PHP-Applikation, als auch das Frontend mit Hilfe von JavaScript Zugriff auf diese hat. Durch die Speicherung in einem Cookie hat der Benutzer die Möglichkeit auch nach Beenden des Browsers zu

einem späteren Zeitpunkt den gleichen Status seiner gehörten Alben beizubehalten. Die Verwendung eines Cookies hat zudem die Vorteile, das weniger Daten zwischen Client und Server übertragen werden müssen.

Die nach objektorientierten Grundsätzen programmierte PHP-Applikation auf Serverseite ist für die Auswertung der Anfragen von Clientseite zuständig. Angeforderte Playlisten werden aus Dateien im Format *XSPF* (XML Shareable Playlist Format), die Jamendo bereitgestellt, in vorbereitete HTML-Konstrukte zur direkten Einbindung in den Yahoo Media Player konvertiert[75]. Suchanfragen oder Anfragen zu zusätzlichen Informationen eines Künstlers führen wiederum zur Generierung von SPARQL-Anfragen durch die PHP-Applikation. Sowohl SPARQL-Anfragen als auch die Ausführung des Lernalgorithmus erfolgen über den Webservice des DL-Learners. Die vom DL-Learner benutzten Algorithmen verwenden wiederum als Wissenquelle den SPARQL-Endpoint von DBTune. Der Lernalgorithmus verwendet dabei zusätzlich noch die in dieser Arbeit erstellte OWL-Ontologie, die sich in einer OWL-Datei im Root-Pfad der Webapplikation befindet.

Vom DL-Learner empfangene Antworten werden von der PHP-Applikation aufbereitet und als fertiges HTML über AJAX an das Frontend zurückgegeben. Die eigentlichen Musikdaten in diesem Fall im Format MP3, da der Yahoo Media Player das Format Ogg nicht unterstützt, werden von Jamendo-Servern direkt als Stream zur Verfügung gestellt.

[75] Der Yahoo Media Player unterstützt diese Format zwar nativ, bei Verwendung dieser tritt allerdings ein Fehler in der Abspielreihenfolge der Stücke auf, weshalb die XSPF-Datei nicht direkt an den Player übermittelt wird.

4.2 Erstellen einer Musik-Ontologie

Eines der Kernstücke dieser Arbeit ist es eine geeignete Ontologie in OWL auf Grundlage der benutzten RDF-Daten zu erstellen, so dass diese später vom benutzten Lernalgorithmus verwendet werden kann. Nach einer Evaluation der vorhandenen RDF-Daten, die vom DBTune-Projekt bereitgestellt werden[76], wurde im Verlauf der Arbeit klar, dass die Ontologie auf Basis der sogenannten Tags erstellt werden muss. Diese Tags werden zur Verschlagwortung der unterschiedlichen verfügbaren Alben von Künstlern verwendet. Zum Zeitpunkt der Evaluation[77] der Daten waren im benutzen RDF-Dump von DBTune insgesamt 53.438 Tags 5.329 verfügbaren Alben zugeordnet[78], so dass sich eine durchschnittliche Anzahl von 10 Tags pro Album ergab. Von diesen 53.438 Tags waren 9.236 wohlunterschieden und 5.990 dieser wohlunterschiedenen Tags kamen nur ein einziges mal vor.

Das Verhältnis der meistverwendeten Tags zu denen der am wenigsten verwendeten entspricht dem einer Exponentialverteilung, so zu sehen auf Abbildung 10. Die 50 meistverwendeten Tags kommen insgesamt 19.175 mal vor und machen somit mehr als ein Drittel aller verwendeten Tags aus, die 250 meistverwendeten Tags mit einer Anzahl von 31.934 ca. 60%. Allein die Summe der vier am häufigsten verwendeten Tags ‘electro’, ‘rock’ ‘ambient’ und ‘pop’, die insgesamt 5804 mal vorkommen, entsprechen ca. 10% aller Tags, was statistisch gesehen letztendlich bedeutet, dass jedes verfügbare Album mit einem dieser vier Top-Tags verschlagwortet ist.

Alle vergebenen Tags sind Eingaben einzelner Worte, die durch die Künstler beim Hinzufügen des Albums in der Jamendo-Datenbank gemacht wurden. Dabei wurden sehr unterschiedliche Tags verwendet, um die Musik zu beschreiben. Nach einer Un-

[76] http://dbtune.org/jamendo/sparql
[77] Juli 2009
[78] Alben ohne Tags sind hier nicht mit eingerechnet.

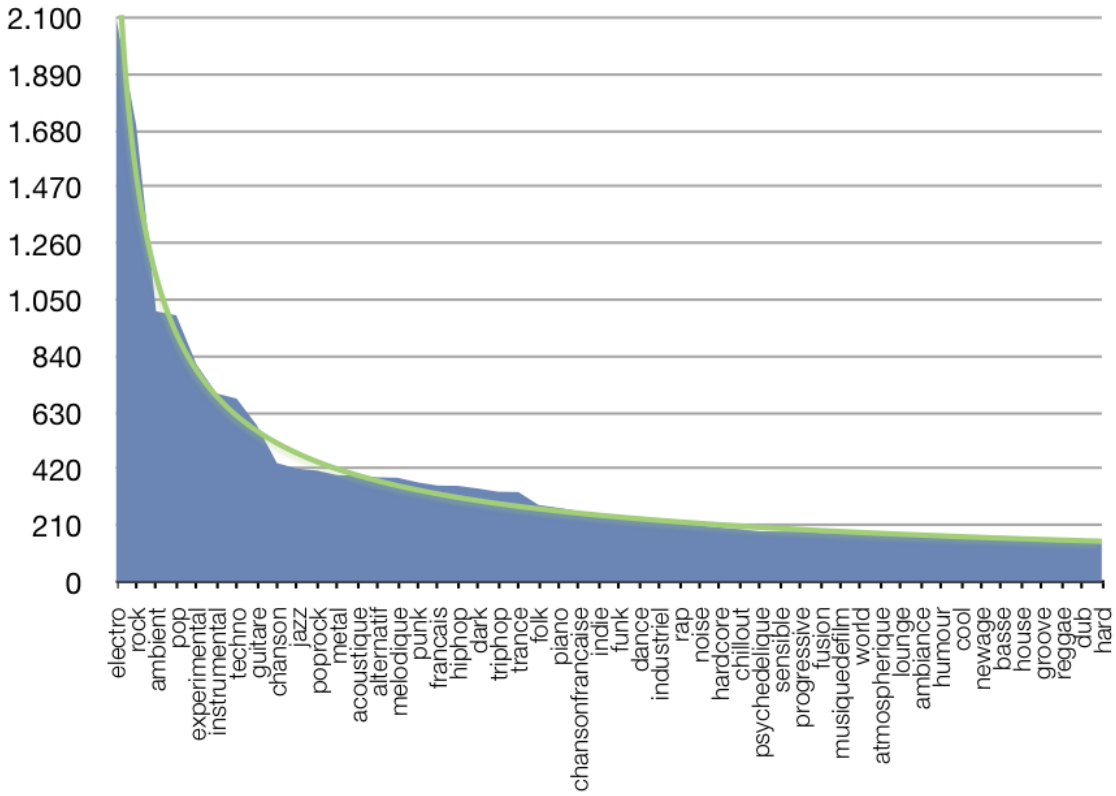

Abbildung 10: Verteilung der 50 meistverwendeten Tags

tersuchung der einzelnen Tags ergaben sich unterschiedliche Kategorien, in die sich die vergebenen Tags eingliedern lassen:

- Musikgenres:
  Ein Großteil der verwendeten Tags beschreibt das zugeordnete Musikgenre, so zum Beispiel 'rock', 'reggae', 'acidjazz' oder 'grindcore' etc. Diese Tags bilden die Grundlage der zu erstellenden Ontologie und werden in dieser entsprechend hierarchisch eingeordnet.
- Musikinstrumente:
  Hier lassen sich allgemeine wie spezielle Musikinstrumente einordnen, zum Beispiel 'ukulele', 'saxophone' oder 'flute', aber auch Klassen von Musikinstrumenten wie zum Beispiel 'brass'[79] oder auch spezielle Tags wie 'whawha'[80].
- Sprach- oder Herkunftsbezeichnungen:
  Diese Tags beschreiben entweder die Sprache des jeweiligen Musikstücks oder auch die Herkunft des Künstlers, wie zum Beispiel 'russian' oder 'berlin'. Hier treten sowohl Tags zum Landes- oder Städtenamen als auch Tags zur Sprache oder der ethnischen Abstammung auf.
- stimmungsbeschreibende Tags:
  Einige der vergebenen Tags beschreiben die generelle subjektive Stimmung oder Wirkung eines Musikstücks, so zum Beispiel 'sad', 'melancholic' oder auch 'relaxing' und 'easylistening'.
- Sonstige:
  Hierzu zählen Tags, die sich nicht in eine der oben genannten Kategorien einordnen lassen, jedoch durchaus relevant sein können, zum Beispiel Tags zur Bezeichnung des Plattenlabels ('17sonsrecords') oder anderen Schlagworten wie zum Beispiel 'conceptalbum', 'unplugged' oder 'seenlive'.

[79] Blechblasinstrumente
[80] Ein elektronisches Effektgerät für E-Gitarren.

Die völlig freie Eingabe der Tags durch die Künstler kann aber natürlich auch zu Problemen führen. Die Untersuchung der Daten zeigte folgende Auffälligkeiten:

- Äquivalente Tags:
  Für einige der vorhandenen Tags sind mehrere Schreibweisen vorhanden. So gibt es zum Beispiel für den Tag 'electro' elf unterschiedliche verwendete Schreibweisen wie 'elektro', 'elctro' oder 'elektronisch' etc. Ursache dafür ist meist die unterschiedliche sprachliche Herkunft des Tags oder schlicht und ergreifend Tippfehler, die die Künstler bei der Eingabe der Tags gemacht haben.
  Das Redundanz-Problem lässt sich in der zu erstellenden Ontologie jedoch relativ einfach dadurch lösen, dass für jede ähnliche Schreibweise das OWL-Property *owl:equivalentClass* eingeführt wird. Durch eine Deklaration wie in Listing 14 wird die Klasse 'ambient' somit äquivalent zu 'ambiental' und zu allen anderen Klassen ähnlicher Schreibweise.
- Allgemeine Worte oder Bindeworte:
  Es existieren Tags die sich in keiner Art und Weise einem Musikgenre oder einer der anderen Kategorien zuordnen lassen. Diese Tags sind wahrscheinlich dadurch entstanden, dass bei der Eingabe der Tags durch die Künstler komplette beschreibende Sätze als einzelne Tags gespeichert wurden, so zum Beispiel Bindeworte wie 'and', 'for', 'of', 'en' oder 'sans' und normale Wörter wie zum Beispiel: 'song', 'musik', 'sound', 'day' oder 'street' – diese Tags finden keine Verwendung in der Ontologie.
- Subjektive Bezeichnungen:
  Viele Künstler haben bei der Vergabe der Tags ihre persönliche Meinung zu ihrer Musik vergeben, weswegen es eine Vielzahl an Tags wie zum Beispiel 'good', 'realgood', 'magnifique', 'genial' oder auch 'excellent' gibt, die ebenfalls keine Verwendung in der Ontologie finden.
- Nicht klassifizierbar:

Tags, die sich in der Ontologie nicht einordnen lassen oder deren Sinnhaftigkeit sich nicht erschließen lässt, wie zum Beispiel ‘1’, ‘i’, ‘mp3’, ‘yesss’, ‘jamendo’, oder auch ‘badlytaggedfiles’ werden ebenfalls nicht mit in die Ontologie übernommen.

- Einzigartige Tags:
  Insgesamt gibt es, wie eingangs schon erwähnt, eine Liste von 5990 Tags, die nur ein einziges Mal vorkommen. Diese werden ebenfalls aus der Liste der in die Ontologie aufzunehmenden Tags ausgeschlossen, da es wenig Sinn macht, diese Tags zur Vorschlagsgenerierung zu verwenden, da es sonst leicht vorkommen kann, dass immer nur das gleiche, dem Tag entsprechende Album, als Resultat in Betracht gezogen wird.

Nach dieser Datenevaluation wurden die verbliebenen Tags mit Hilfe des Ontologie-Editors Protégé[81] in eine OWL-Ontologie überführt. Aus Aufwands- und Relevanzgründen wurden Tags, die seltener als vier mal vergeben wurden, sowie Tags aus der genannten Kategorie ‘Sonstige’ nicht mit in die Ontologie übernommen.

```
:ambient rdf:type owl:Class ;
  owl:equivalentClass
    :ambiental ,
    :ambiente ,
    :downtempo ,
    :planant ,
    :tacambient ;
rdfs:subClassOf :electro .
```

Listing 14: Auszug aus moosique.owl mit äquivalenten Tags

Die genannten Kategorien Musikinstrumente, Sprach- oder Herkunftsbezeichnungen und stimmungsbeschreibende Tags ließen sich anhand bereits vorhandener Ontologien beschreiben und wurden in die entsprechenden Klassen `mo:Instrument`, `dc:language`

[81] http://protege.stanford.edu

und `mo:Mood` einsortiert. Innerhalb dieser Klassen konnte die weitere Klassifizierung intuitiv vorgenommen werden, da die zugrunde liegenden Hierarchien einer flachen Baumstruktur entsprechen, oder wie im Fall der Musikinstrumente bestehende Klassifizierungen einfach übernommen werden konnten[82].

Schwieriger war jedoch die Klassifizierung der Musikgenres, da es kein generelles Verfahren zur Klassifizierung derselben gibt. Dies hängt zum einen damit zusammen, dass viele Musikgenres miteinander verwandt sind und sich nicht voneinander trennen oder in Taxonomien einordnen lassen. Zum anderen gibt es verschiedene Ansätze, die Vielzahl an Musikgenres zu sortieren: nach geschichtlichem Ursprung, nach verwendeten Instrumenten oder nach allgemeinen Stilrichtungen. Eine intuitive Klassifizierung ist hier ebenfalls schwierig zu realisieren, da die Objektivität der entstehenden Ontologie damit nicht unbedingt gegeben ist. Die in dieser Arbeit erstellte Ontologie orientiert sich an den Prinzipien von Pachet und Cazaly (2000) und versucht eine Einordnung nach Häufigkeit des Auftretens eines genrebeschreibendem Tags und stilistischem Zusammenhang vorzunehmen. Häufig verwendete Tags, wie zum Beispiel ‘rock’ oder ‘electro’ bilden dabei sogenannte Metaklassen, von denen viele andere Klassen wie zum Beispiel ‘metal’ oder ‘techno’ abgeleitet und als Unterklassen definiert werden können. Die einzelnen Klassen werden dann, nachdem sie als Unterklasse definiert wurden, mit weiteren, ähnlichen Klassen verknüpft. Ähnlich bedeutet hierbei, dass ein semantischer Zusammenhang aufgrund stilistischer oder geschichtlicher Hintergründe besteht.

Ein weiteres Problem bei der Erstellung der Ontologie waren die vorliegenden Daten. Die von Jamendo benutzten Tags bestehen immer nur aus einem Wort. Allerdings gibt es viele Genres, die aus mehreren Wörtern bestehen, wobei es hierfür auch keine einheitliche Schreibweise gibt. So lässt sich zum Beispiel ‘classic rock’ auch als ‘classicrock’ schreiben. Bei der Vergabe der Tags durch die Künstler entstanden aus der

[82] Vergleiche Valentin et al. (2004).

ersten Schreibweise jedoch bei der Datenübernahme in Jamendo zwei Tags, nämlich ‘classic’ und ‘rock’. Der Tag ‘classic’ kann ohne einen weiteren semantischen Zusammenhang jedoch auch als „klassische Musik“ interpretiert werden, die allerdings nicht mehr viel mit Rockmusik gemein hat. Für solche kritischen Tags wie ‘classic’ musste in einer weiteren Datenevaluation untersucht werden, wie häufig ein gemeinsames Auftreten mit anderen Tags eventuell das gemeinte Genre beschreibt, um diese richtig klassifizieren zu können.

Da viele Musikgenres sich an Stilistiken anderer Genres bedienen und in vielen Musikrichtungen diese Stile miteinander kombiniert werden, war es dadurch auch nicht möglich, die einzelnen Klassen voneinander abzugrenzen. Sogar bei anscheinend offensichtlicher Unterschiedlichkeit, wie zum Beispiel bei den Genres ‘metal’ und ‘techno’, konnte die Relation `owl:disjointWith` nicht verwendet werden, da es durchaus Musikgenres gibt, die beiden Klassen zugeordnet werden können[83]. Durch Inferenz der beiden übergeordneten Klassen wäre es durch den Einsatz disjunkter Klassen somit zu einem logischen Widerspruch in der Ontologie gekommen.

Die entstandene Ontologie besteht somit nur aus Klassen, die mit den Prädikaten `owl:equivalentClass` und `rdfs:subClassOf` miteinander in Beziehung gesetzt wurden. Als Klassen wurden nur in Jamendo vorhandene Tags und die beschriebenen Hauptkategorien verwendet. Einen Ausschnitt zeigt Abbildung 11.

Die auf die beschriebene Art und Weise implementierte Ontologie beinhaltet damit ungefähr 500 der meistverwendeten, sinnvollen Tags. Die Abdeckung für alle vergebenen Tags liegt damit bei über 80%, womit die Ontologie für den später verwendeten Lernalgorithmus eine ausreichende Grundlage bildet. Zu vermerken ist hierbei allerdings noch, dass der verwendete SPARQL-Endpoint von DBTune, als auch die bereitgestellten RDF/XML-Daten nicht aktuell sind. Auf der offiziellen Webpräsenz

[83] Zum Beispiel ‘breakcore’ oder ‘industrialmetal’.

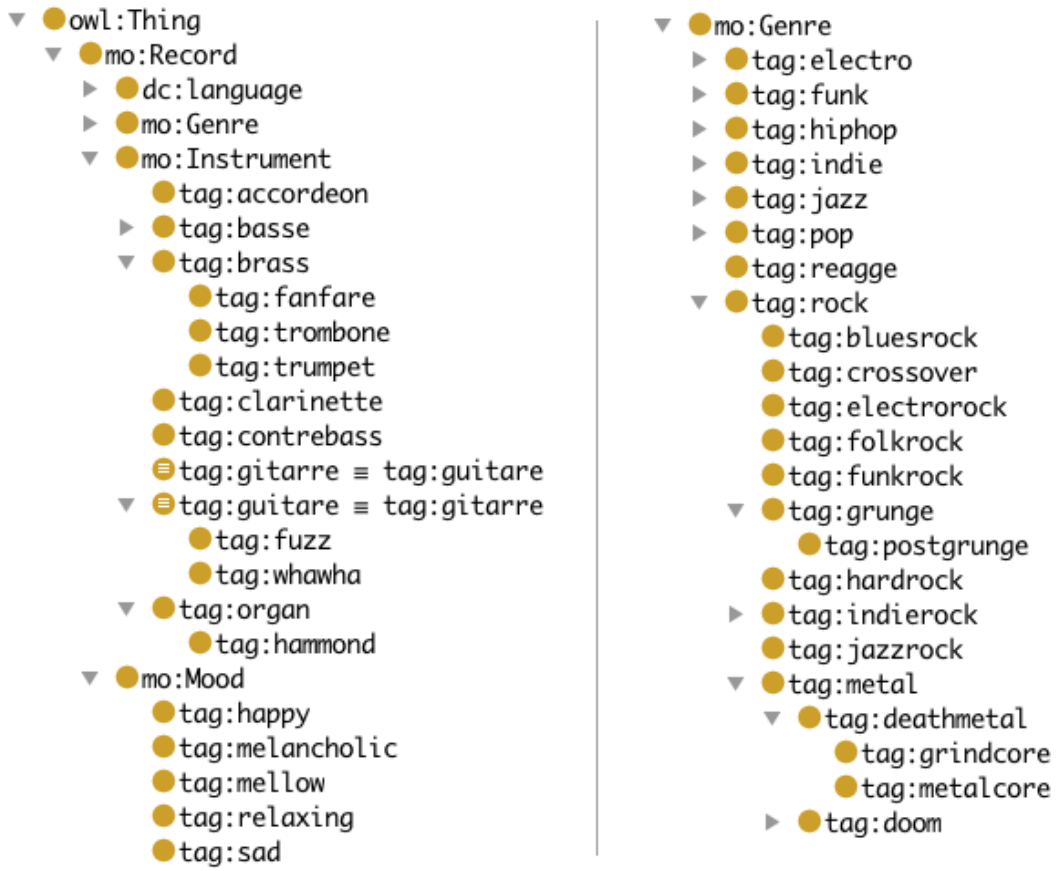

Abbildung 11: Ausschnitt aus der Ontologie zur Tag-Klassifizierung

von Jamendo ist die Rede von über 27.000 Alben, im SPARQL-Endpoint sind gerade einmal 5.768 Alben als Ressourcen hinterlegt.

4.3 Die SPARQL-Schnittstelle

Zur Umsetzung der Suchanfragen, die durch einen Benutzer gestellt werden können, musste eine entsprechende Schnittstelle zum SPARQL-Endpoint von DBTune geschaffen werden. Die Begriffe, nach denen der Benutzer sucht, werden per AJAX an die serverseitig laufende PHP-Applikation weitergegeben. Abhängig von der gewählten Kategorie wird aus dem Suchbegriff durch die PHP-Klasse `SparqlQueryBuilder` eine Suchanfrage generiert, die dann über die SPARQL-Schnittstelle[84] des DL-Learners an DBTune geschickt wird. Dabei gab es folgende Fälle zu unterscheiden:

- Suche mit einem Wort:
  Gibt ein Benutzer als Suchbegriff ein zusammenhängendes Wort ein, so kann dieses direkt als Suchbegriff für die SPARQL-Anfrage benutzt werden. Dabei wird durch Angabe eines Filters in der SPARQL-Anfrage nach entsprechenden Künstlernamen, Tags oder Song- und Albumtiteln gesucht. Bei einer Suche nach Tags werden dann zum Beispiel für den Suchbegriff ‘rock’ auch Ergebnisse gefundener Alben für ‘punkrock’ oder ‘indierock’ angezeigt. Eine Suche nach Künstlern mit dem Suchbegriff ‘fish’ würde die SPARQL-Anfrage aus Listing 15 generieren.
- Suche mit mehreren Wörtern:
  Gibt ein Benutzer eine Zeichenkette als Suchbegriff ein, die aus mehreren Worten besteht[85] erfolgt eine Fallunterscheidung nach ausgewählter Suchkategorie: für eine Suche nach Künstlernamen, Song- und Albumtiteln kann der Suchbe-

[84] Die vom Webservice implementierte Funktion `sparqlQuery()`.
[85] Zeichenketten die durch Leerzeichen unterbrochen sind.

griff wie ein Wort aufgefasst werden, da diese Knoten von Literalen repräsentiert werden. Das Prädikat ist in diesem Fall `rdf:datatype=''&xsd;string''`. In einem Literal sind Leerzeichen erlaubt, weshalb eine Suche nach Künstlern mit dem Suchbegriff 'Echo lali' unter Verwendung eines Filters ein Ergebnis liefert[86]. Bei einer Suche nach Tags und mehreren Worten würde ein Filter nicht die gewünschten Resultate liefern. Ein Tag ist eine Ressource und wird deshalb von einem URI repräsentiert, in welchem keine Leerzeichen erlaubt sind. Bei einer Suche nach 'classic rock' würde die Angabe von `FILTER (regex(str(?tag), ''classic rock'', ''i'')) .` ein leeres Ergebnis zur Folge haben.

Die Problematik mit Suchanfragen, die aus mehreren Worten bestehen, wurde schon beim Erstellen der Ontologie in Kapitel 4.2, ab Seite 42, festgestellt. Im Fall der Suche nach Tags lässt sich das Problem durch eine SPARQL-Abfrage wie in Listing 16 jedoch leicht umgehen, indem die Suchwörter als einzelne Tags interpretiert werden. Die Suche erfolgt dabei über die Angabe dieser als Objekte in der SPARQL-Anfrage.

```
1  SELECT DISTINCT * WHERE {
2    ?artist rdf:type mo:MusicArtist ;
3            foaf:name ?artistName ;
4            foaf:made ?record .
5    ?record rdf:type mo:Record ;
6            dc:title ?albumTitle ;
7            mo:available_as ?playlist ;
8            tags:taggedWithTag ?tag .
9    OPTIONAL { ?artist foaf:img ?image . }
10   OPTIONAL { ?artist foaf:homepage ?homepage . }
11   FILTER (regex(str(?artistName), "fish", "i")) .
12   FILTER (regex(str(?playlist), "xspf", "i")) . }
```

Listing 15: SPARQL-Anfrage für die Suche nach Künstlern mit dem Suchbegriff 'fish'

[86] Vergleiche Kapitel 2.3, ab Seite 10,, Abbildung 3.

```
SELECT DISTINCT * WHERE {
  ?artist rdf:type mo:MusicArtist ;
          foaf:name ?artistName ;
          foaf:made ?record .
  ?record rdf:type mo:Record ;
          dc:title ?albumTitle .
          mo:available_as ?playlist ;
          tags:taggedWithTag ?tag .
          tags:taggedWithTag <http://dbtune.org/jamendo/tag/classic
              >'
          tags:taggedWithTag <http://dbtune.org/jamendo/tag/rock>'
  OPTIONAL {
    ?record mo:image ?cover .
    FILTER (regex(str(?cover), "1.100.jpg", "i")) .
  }
  FILTER (regex(str(?playlist), "xspf", "i")) .
}
```

Listing 16: SPARQL-Anfrage für die Suche nach Tags mit mehreren Suchwörtern: 'classic rock'

Zu dem wurde die Möglichkeit implementiert, durch die Angabe des Benutzernamens von last.fm eine Suchanfrage zu senden. Diese Suchanfrage benutzt das last.fm-API und versucht die jeweils meistgebrauchten Tags des Benutzers zu extrahieren. Aus den gefundenen Tags werden dann mehrere SPARQL-Anfragen generiert, je nach Anzahl der Worte des vergebenen Tags analog zu Listing 15 oder 16. Bei all diesen Suchanfragen ist zu beachten, dass diese nur Ergebnisse zurückliefern, bei denen das verknüpfte Album mindestens mit einem Tag in Beziehung gesetzt wurde, da diese später die Grundlage für den verwendeten Lernalgorithmus bieten.

Die Performance der SPARQL-Schnittstelle und somit auch die der Suche ist stark davon abhängig, wie viele Ergebnisse zu einer bestimmten Anfrage existieren. Für die Suche nach dem Tag 'rock' ist die Antwort vom DBTune-Endpoint 2,43 Megabyte groß

und beinhaltet 3.195 Ergebnisse. Um die Geschwindigkeit einer Suche zu verbessern gibt es deshalb die Möglichkeit, die Anzahl der Suchergebnisse einzuschränken. In SPARQL ist dies durch die Angabe von `LIMIT`[87] möglich. Möchte man nun die Anzahl der gefundenen Ergebnisse auf beispielsweise 20 beschränken, wird an die SPARQL-Anfrage `LIMIT 20` angehängt.

In der Praxis ist dies allerdings nur bedingt empfehlenswert. Die SPARQL-Anfrage aus Listing 15 liefert Ergebnisse für zwei Künstler: 'Fishbone Rocket' und 'Fish Man'. Insgesamt besteht das Resultat der Suche allerdings aus 29 Ergebnissen, da dem ersten Künstler 24 und dem zweiten fünf Tags zugeordnet sind. Würde man die Ergebnismenge nun mit `LIMIT 20` beschränken, würde nur noch der Künstler 'Fishbone Rocket' mit nunmehr 20 Tags als Ergebnis zurückgeliefert werden. Die Suchergebnisse wären dadurch verfälscht.
Ein weiteres Problem beim Einsatz von `LIMIT` ist, dass die eingeschränkte Ergebnismenge immer die gleiche ist. Es wäre wünschenswert, wenn sich die Ergebnismenge zufällig wählen lassen würde, so dass ein Benutzer nicht immer die gleichen Resultate angezeigt bekommt. In SQL lässt sich dies durch die Angabe von `ORDER BY RAND()` realisieren. In SPARQL ist diese Funktion allerdings nicht implementiert. Aus diesem Grund wurde bei der Implementation der Webapplikation auf den Einsatz von `LIMIT` verzichtet.

Um trotzdem die Anzeige der Suchergebnisse für den Benutzer übersichtlich zu gestalten und diese zu beschleunigen, wurde eine Kompromisslösung implementiert. Die Ergebnisse einer SPARQL-Anfrage lassen sich vom DL-Learner im JSON-Format zurückliefern und somit mit der PHP-Funktion `json_decode` in ein PHP-Objekt konvertieren. Dieses Objekt lässt sich dann mit den eigens dafür programmierten Funktionen `object2array()` und `mergeArray()`[88] in einen Array konvertieren. Dieser Array kann nun mit Hilfe der PHP-eigenen Funktionen `array_rand()` und `array_slice()` auf

[87] Siehe Listing 12.
[88] Zu finden unter /src/moosique.net/moosique/classes/DataHelper.php.

eine bestimmte Anzahl an Ergebnissen reduziert werden, wobei diese auch zufällig ausgewählt werden können.
Dieses Vorgehen verringert zwar die Größe der Daten, die vom Server an den Browser geschickt wird, an den Daten die zwischen dem SPARQL-Endpoint und der Webapplikation übertragen werden, ändert sich jedoch nichts. Allerdings können dadurch, dass die Ergebnisse einer SPARQL-Anfrage vom DL-Learner in einem Cache zwischengespeichert werden, selbst große Ergebnismengen, sofern diese bereits gecached sind, zügig an die PHP-Applikation weitergegeben werden.

Ein Benutzer hat die Möglichkeit sich zusätzliche Informationen zum Künstler des aktuell abgespielten Musikstücks anzeigen zu lassen. Dafür wird ebenfalls eine SPARQL-Anfrage generiert, die alle eventuell vorhandenen, zusätzlichen Informationen unter der Verwendung von `OPTIONAL` vom SPARQL-Endpoint abfragt. Diese zusätzlichen Informationen sind die Homepage des Künstlers `foaf:homepage`, die Herkunft des Künstlers `foaf:based_near` und die sogenannte *Musicbrainz-ID* über die Relation `owl:sameAs`. Diese ID ist ein eindeutiger Bezeichner für einen Künstler, der auch auf anderen Plattformen verwendet wird, unter anderem musicbrainz[89] oder auch last.fm. Anhand dieser ist es möglich weitere Informationen von anderen Plattformen einzubinden, was in der Webapplikation über eine direkte Einbindung mit Iframes geschieht. Allerdings besitzen nur 119 Künstler, die auf Jamendo eingetragen sind, eine solche ID, weshalb die Zusatzinformationen externer Quellen nicht immer angezeigt werden.

4.4 Der Learnalgorithmus CELOE

Für das in Kapitel 3.4, ab Seite 36, vorgestellte Lernproblem dieser Arbeit bietet der DL-Learner die genannten drei Algorithmen Brute-Force, Random Guesser und CE-

[89] http://musicbrainz.org

LOE. Die ersten beiden Algorithmen verfolgen einen relativ simplen Lösungsansatz, indem diese versuchen durch reines Ausprobieren einen passenden Klassenausdruck zu finden. Der Algorithmus CELOE basiert auf einem heuristischen Ansatz und bedient sich dabei fortgeschrittenerer Maßnahmen zur Verwendung des bereitgestellten Hintergrundwissens. Dieser Algorithmus benötigt, wie bereits in Kapitel 2.7, ab Seite 23, und Kapitel 3.5, ab Seite 37, beschrieben, positive Beispiele als Wissensgrundlage, die aus den gehörten Musikstücken abgeleitet werden. Da Musikstücke in Jamendo nie mit Tags verknüpft werden, die die Grundlage der Ontologie bilden, ist das eigentliche positive Beispiel das zugrunde liegende Album, also eine Ressource der Form `http://dbtune.org/jamendo/record/11391`. Eine solche Ressource ist immer vom Typ `mo:Record` und hat immer ein Tripel der Form `tags:taggedWithTag rdf:resource="http://dbtune.org/jamendo/tag/TAG"/>`. Somit ist jedes Album als Startinstanz für den Lernalgorithmus nutzbar.

Da als Klassenausdruck nur eine Kombination aus Tags gewünscht ist, um daraus weitere SPARQL-Anfragen generieren zu können und so dem Benutzer weitere Alben vorzuschlagen, ist die Wahl der nicht-positiven Instanzen trivial[90]. Diese werden benötigt, um den Algorithmus mit zusätzlichem Wissen zu versorgen und werden zufällig aus einer Liste aller Alben gewählt, die mit mindestens einem Tag versehen sind. Die Anzahl der Instanzen kann dabei abhängig von der Anzahl positiver Beispiele gewählt oder auf einen festen Wert gesetzt werden. Bei einer maximalen Anzahl zehn positiver Beispiele konnte als Optimum ein Wert zwischen drei und fünf für die zufällig gewählten Instanzen ermittelt werden. Die Anzahl der zufällig gewählten Instanzen hat direkten Einfluss auf die Performance des Lernalgorithmus, was in Kapitel 5.1.2, ab Seite 61, noch genauer behandelt wird, und sollte deshalb so gering wie mögliche gehalten werden.

Daraus ergaben sich folgende Konfigurationsparameter für die Fragmentextraktion

[90] Vergleiche Hellmann et al. (2009).

und den eingesetzten Algorithmus:

```
problem = "posOnlyLP"
algorithm = "celoe"
maxExecutionTimeInSeconds = 3
useHasValueConstructor = true
valueFrequencyThreshold = 2
reasoner = "fastInstanceChecker"
recursionDepth = 1
saveExtractedFragment = true
instances = 4
```

Listing 17: Konfiguration für den Lernalgorithmus

Die bei einem erfolgreichen Lernvorgang gefundenen Klassenausdrücke in KB-Syntax können dann, mit der durch den Webservice bereitgestellten Funktion `kbToSqarql()`, in SPARQL-Fragmente konvertiert werden. Diese Fragmente werden schließlich an eine SPARQL-Anfrage (ähnlich der Anfrage zur Suche nach Alben) angehängt, um aus den so gefunden Klassenausdrücken dem Benutzer Vorschläge in Form einer Auflistung zufällig gewählter Alben anzuzeigen. Diese können dann zur seiner Abspielliste hinzufügt werden.

4.5 Zusammenfassung

Insgesamt haben sich bei der Implementation keine größeren Probleme ergeben. Es hat sich jedoch in der im Vorfeld durchgeführten Datenevaluation bereits ergeben, dass die von Jamendo bereitgestellten RDF-Daten wenig zusätzliche Informationen zu einzelnen Künstlern, Musikstücken oder Alben bereitstellen. Inwieweit diese vorhandenen Daten Einfluss auf die Qualität der Ergebnisse haben, wird in Kapitel 5, ab Seite 57, behandelt. Der Quellcode der in dieser Arbeit entstandenen Webapplikation ist im Sourceforge-Repository des DL-Learners unter `/src/moosique.net/` zu finden.

5 Evaluation

Nach der erfolgreichen Implementation der Webapplikation und den Schnittstellen zum DL-Learner werden in diesem Kapitel Untersuchungen bezüglich der Performance der eingesetzten Algorithmen sowie der Relevanz der resultierenden Ergebnisse durchgeführt und die Ergebnisse vorgestellt. Zusätzlich zu dieser Evaluation werden mögliche Vorgehensweisen einer Optimierung aufgezeigt.

5.1 Performanceuntersuchungen

Alle Untersuchungen verwenden den bereits genannten unter DBTune bereitgestellen SPARQL-Endpoint für Jamendo. Die Untersuchungen wurden, sofern nicht explizit anderweitig angegeben, unter nachfolgender Systemkonfiguration durchgeführt:

Hardware:

- Macbook 2GHz Intel Core Duo (32 bit)
- Mac OS X 10.6.2
- 2GB 667 MHz DDR2 SDRAM
- 5400rpm Western Digital WD5000BEVT
- 16.000/1.024 kbit/s DSL

Software:

- Java Version 1.6.0_03-p3 (soylatte/openJDK)
- Apache 2.2.14 / PHP Version 5.2.11

Alle Lernanfragen wurden pro Durchlauf insgesamt 100 Mal aufgerufen, um für die entstehenden Werte einen repräsentativen Mittelwert berechnen zu können. Als posi-

tive Beispiele wurden bis zu zehn Alben eines gleichen Genres gewählt. Die Instanzen des Lernproblems waren vier zufällig gewählte, zu den positiven Beispielen verschiedene Alben und wurden pro Lernanfrage neu generiert. Die Ergebnisse der folgenden Tabellen stellen immer den berechneten Mittelwert eines Durchlaufs dar.

Im vorgestellten Testverfahren werden hierbei sowohl die Laufzeit und Performance der Fragmentextraktion, als auch die des Lernalgorithmus untersucht. Das dazu programmierte Skript ist im DL-Learner-Repository zu finden unter `/src/moosique.net/moosique/testing/learnPerformanceTest.php`.

5.1.1 Lernalgorithmus

Der verwendete Lernalgorithmus CELOE lässt sich, wie schon in Kapitel 4.4, ab Seite 54, beschrieben, in der maximalen Laufzeit beschränken. Die Laufzeit des Algorithmus hat starken Einfluss auf die Genauigkeit und Relevanz der Ergebnisse. Ziel der Evaluation des Lernalgorithmus war es, die Laufzeit so zu bestimmen, dass die resultierenden Ergebnisse zufriedenstellend sind. Ein Ergebnis ist für die Webapplikation dann zufriedenstellend, wenn die berechnete Genauigkeit größer als 80% ist und der berechnete Klassenausdruck nur Tags als Klassen beinhaltet. Tabelle 2 stellt die Anzahl getesteter Klassenausdrücke in Abhängigkeit zur Laufzeit des Algorithmus in Sekunden dar.

Zeit in Sekunden	Anzahl getesteter Klassenausdrücke
1	44.584
3	78.121
5	98.603
10	176.630
30	349.817

Tabelle 2: Anzahl getesteter Klassenausdrücke in Abhängigkeit von der maximalen Ausführungszeit des Lernalgorithmus CELOE

Die Ergebnisse hatten in allen Fällen eine Treffergenauigkeit von über 80%, jedoch variierten die erwarteten Ergebnisse der Klassenausdrücke in den Bereichen zwischen 1 und 3 Sekunden stark. So kam es bei einer Laufzeit von einer Sekunde häufiger zu unbrauchbaren Ergebnissen wie in Listing 18, die zwar eine gute Trefferquote haben, jedoch für eine Vorschlagsgenerierung nicht verwendbar sind.

```
1 (http://purl.org/ontology/mo/Genre and http://www.w3.org/2000/01/rdf
      -schema#Resource) 93,85%
2 (http://purl.org/ontology/mo/Genre and http://www.holygoat.co.uk/owl
      /redwood/0.1/tags/Tag) 93,85%
3 http://www.holygoat.co.uk/owl/redwood/0.1/tags/Tag 92,26%
```

Listing 18: Unerwünschte Ergebnisse (in KB-Syntax)

Ab circa drei Sekunden waren die Ergebnisse jedoch zufriedenstellend, so dass im Normalfall sogar mindestens ein Ergebnis mit einer Trefferquote von 100% berechnet werden konnte. Für Laufzeiten länger als fünf Sekunden konnte jedoch keine Verbesserung der Ergebnisqualität festgestellt werden. Die Resultate des Lernalgorithmus bei einer Laufzeit von fünf Sekunden und einer Laufzeit von 30 Sekunden waren somit nahezu identisch. Das optimale Maximum zu testender Klassenausdrücke liegt somit bei etwa 90.000.

Das Resultat ist weiterhin stark von der Anzahl der positiven Beispiele abhängig. Bei einem einzigen positiven Beispiel entsprechen die resultierenden Klassenausdrücke meist den vergebenen Tags des Albums, die Trefferquoten sind dadurch sehr hoch ($>$ 80%, 100% nicht unüblich). Zusammengesetzte Klassenausdrücke wie in Listing 18 konnten bei zunehmender Anzahl positiver Beispiele gehäuft beobachtet werden, jedoch konnte auch bei der maximalen Anzahl von zehn positiven Beispielen gleichen Genres immer mindestens ein zufriedenstellendes Ergebnis berechnet werden.

Der Einfluss der Anzahl der zufällig gewählten Instanzen auf die Ergebnisse des Algorithmus konnte dahingehend beobachtet werden, dass bei zunehmender Anzahl die gefundenen Klassenausdrücke häufig zusammengesetzte Klassenausdrücke waren. Da diese gefundenen Ausdrücke jedoch in ungefähr der Hälfte aller Fälle nicht zufriedenstellend waren, und damit keinen wesentlichen Einfluss auf die Qualität der Ergebnisse hatten, wurde aus Performancegründen ein fester Wert für die Anzahl der Instanzen vergeben[91].

Bei der Evaluation wurde des Weiteren festgestellt, dass die vorgegebenen maximalen Laufzeiten des Algorithmus vom System eingehalten wurden und es eine Abweichung von bis zu maximal 30ms nach oben gab. Diese Abweichung entsteht aus der internen Implementierung, die den Algorithmus nicht nach exakt der angegeben Zeit terminiert, sondern noch solange wartet, bis die aktuelle Berechnung vollendet ist.

Insgesamt war die Ergebnisqualität des verwendeten Algorithmus CELOE mehr als zufriedenstellend. Durch Performance- und Langzeittests bei der Benutzung der Webapplikation haben sich als optimalen Werte für den Parameter `maxExecutionTimeInSeconds` Zeiten zwischen drei und fünf Sekunden ergeben. Dieser Wert ist jedoch selbstverständlich nur unter der gegebenen Systemkonfiguration als optimal anzusehen und sollte bei schnelleren oder langsameren Computersystemen entsprechend konfiguriert werden.

[91] Vergleiche Kapitel 5.1.2, ab Seite 61,.

5.1.2 Fragmentextraktion

Fragmentextraktion beschreibt den Teil des Lernverfahrens, der für die gegebenen positiven Beispiele und zufälligen Instanzen aus den gegebenen Wissensbasen weitere Informationen extrahiert. Es wird für jede gegebene Instanz versucht zusätzliche Informationen aus dem SPARQL-Endpoint von DBTune zu extrahieren. Dabei lässt sich mit dem Parameter `recursionDepth` die Rekursionstiefe der zu extrahierenden Informationen angeben. In der Praxis hat sich dabei eine Tiefe von eins bis zwei als bestes Mittelmaß bewährt. Eine Tiefe von eins ist für die implementierte Webapplikation ausreichend und liefert die gewünschten Informationen zu den vergebenen Tags. Eine Rekursionstiefe höher als zwei ist in dieser Applikation nicht nötig, da dadurch keine signifikante Verbesserung der Ergebnisse erzeugt werden konnte.

Der DL-Learner bietet standardmäßig, die extrahierten Fragmente der Instanzen lokal in einem *Cache* zwischenzuspeichern, um eine wiederholte Extraktion der gleichen Instanz erheblich zu beschleunigen. Um die Dauer einer solchen Extraktion unverfälscht zu bestimmen wurde zunächst für die folgenden Performancetests der Cache deaktiviert. Tabelle 3 zeigt einen Vergleich der benötigten Laufzeiten in Abhängigkeit der Rekursionstiefe bei deaktiviertem Cache[92]:

	recursionDepth = 1	recursionDepth = 2
posExamples = 1 / instances = 4	19,464s	44,083s
posExamples = 3 / instances = 4	26,220s	67,374s
posExamples = 7 / instances = 4	35,127s	98,765s
posExamples = 10 / instances = 4	49,020s	113,502s

Tabelle 3: Laufzeit in Abhängigkeit der Rekursionstiefe bei deaktiviertem Cache

[92] Von den Zeiten, die durch das Skript berechnet wurden, wurde die angegebene Laufzeit des Lernalgorithmus, in diesem Fall drei Sekunden, abgezogen.

Bei einer Rekursionstiefe von zwei konnte die Laufzeit durch Exklusion bestimmter Klassen noch erheblich verbessert werden. Dabei wurden Knoten ausgeschlossen, die keine Einwirkung auf die Ergebnisse des Lernalgorithmus haben. Tabelle 4 zeigt ein Vergleich bei Rekursionstiefe zwei und Exklusion der Knoten `mo:image`, `mo:available_as`, `mo:track`, `serql:directType` und `dc:title`.

	ohne Exklusion	mit Exklusion
posExamples = 1 / instances = 4	44,083s	21,746s
posExamples = 3 / instances = 4	67,374s	27,980s
posExamples = 7 / instances = 4	98,765s	35,956s
posExamples = 10 / instances = 4	113,502s	48,741s

Tabelle 4: Exklusion bestimmter Knoten bei Rekursionstiefe zwei

Durch die Exklusion der angegebenen Knoten konnten bei einer Rekursionstiefe von zwei die Laufzeiten der Fragementextraktion wesentlich verkürzt werden. Die so entstandenen Laufzeiten entsprechen somit denen der Rekursionstiefe von eins. Die durchschnittliche Dauer, die der Algorithmus unter den gegebenen Umständen zur Fragmentextraktion pro Instanz benötigt, beträgt damit ungefähr 3,6 Sekunden.

In der Praxis ist es jedoch sehr zu empfehlen, den Cache des DL-Learners nicht zu deaktivieren, da sich durch Benutzung des Caches die benötigte Laufzeit drastisch verkürzen lässt. Einen Vergleich bietet Tabelle 5:

Auffällig hierbei ist, dass die benötigten Laufzeiten bei aktiviertem Cache mit circa fünf Sekunden nahezu konstant bleiben. Das liegt daran, dass bei diesem Performancetest die gewählten positiven Beispiele immer die selben waren, und somit nach einmaliger Extraktion im ersten Durchlauf im Cache gespeichert wurden. Für alle darauf folgenden Durchläufe musste die Fragmentextraktion dann nur noch für die vier zu-

	Cache deaktiviert	Cache aktiv
posExamples = 1 / instances = 4	19,464s	4,922s
posExamples = 3 / instances = 4	26,220s	5,179s
posExamples = 7 / instances = 4	35,127s	5,425s
posExamples = 10 / instances = 4	49,020s	5,338s

Tabelle 5: Performance mit (de-)aktiviertem Cache, Rekursionstiefe eins

fällig gewählten Instanzen durchgeführt werden, die immer neu generiert werden. Die Zeit die benötigt wird, um eine bereits im Cache vorhandene Instanz zu untersuchen ist mit weniger als 20 Millisekunden verschwindend gering und wirkt sich somit nicht auf die Gesamtlaufzeit aus.

Dieses Verhalten konnte auch beim Einsatz der Webapplikation beobachtet werden, obwohl die positiven Beispiele hier nicht immer die selben sind. Die Erklärung hierfür ist folgende:
Jedes mal, wenn ein Benutzer ein Musikstück bis zur Hälfte gehört hat, wird das Album, das mit dem Musikstück verknüpft ist, zur Liste positiver Beispiele hinzugefügt. Sollte das Album bereits in der Liste vorhanden sein, wird dieses nicht nochmals zur Liste hinzugefügt. Das heißt, dass sich pro Lernanfrage die positiven Beispiele im Vergleich zur vorigen Lernanfrage um höchstens ein Element unterscheiden. Die Gesamtanzahl der zu extrahierenden Fragmente, die nicht bereits im Cache zwischengespeichert wurden, ist somit maximal gleich der Anzahl der zufällig gewählten Instanzen plus eins.

Die damit benötigte Zeit zur Ausführung eines kompletten Lernvorgangs, bestehend aus Fragmentextraktion und Durchlauf des Lernalgorithmus, ist damit konstant. Als optimale maximale Laufzeit des Lernalgorithmus hat sich ein Wert von drei Sekunden ergeben. Die Ausführungszeit einer Fragmentextraktion von ungefähr fünf Sekunden

ist direkt abhängig von der Anzahl der zufällig gewählten Instanzen, die in diesem Fall vier beträgt. Die obere Zeitschranke, die zur Vorschlagsgenerierung notwendig ist, beträgt damit ungefähr zehn Sekunden.

5.2 Lokaler SPARQL-Endpoint

Da die bisherigen Performancetests alle den SPARQL-Endpoint von DBTune verwendeten, war es notwendig einen Vergleich mit einem lokalen Endpoint herzustellen, um abschätzen zu können, wie die Anbindungsgeschwindigkeit an den Endpoint von DBTune die Ergebnisse beeinflusst. Die Vermutung, dass sich eine schlechtere Anbindungsgeschwindigkeit direkt auf die Performance des Algorithmus auswirkt, liegt nahe. Um einen lokalen SPARQL-Endpoint bereitzustellen wurde die universelle Serverplattform Virtuoso[93] in der Open Source Edition[94] installiert und anschließend die RDF-Daten von DBTune importiert[95]. Für optimale Ergebnisse wurde die Konfiguration des Virtuoso-Servers wie in Listing 19 angepasst:

```
1 MaxQueryExecutionTime         = 600
2 MaxCheckpointRemap            = 1000000
3 NumberOfBuffers               = 360000
4 MaxMemPoolSize                = 0
5 StopCompilerWhenXOverRunTime = 1
```

Listing 19: Konfiguration für Virtuoso

Tabelle 6 zeigt einen Vergleich zwischen dem mit Virtuoso realisierten lokalen SPARQL-Endpoint und dem von DBTune:

[93] http://www.openlinksw.com/virtuoso/
[94] Version 5.0.12.3041-pthreads
[95] http://moustaki.org/resources/jamendo-rdf.tar.gz

	Virtuoso (lokal)	DBTune (extern)
posExamples = 1 / instances = 4	3,367s	4,922s
posExamples = 3 / instances = 4	3,535s	5,179s
posExamples = 7 / instances = 4	3,883s	5,425s
posExamples = 10 / instances = 4	3,540s	5,338s

Tabelle 6: Performancevergleich zwischen lokalem und externen SPARQL-Endpoint mit aktiviertem Cache, Rekursionstiefe eins

Auch hier ist wieder zu beobachten, dass die benötigten Laufzeiten unabhängig von der Anzahl positiver Beispiele nahezu konstant bleiben. Allerdings benötigt der Algorithmus bei Verwendung des lokalen SPARQL-Endpoints im Durchschnitt circa 1,7 Sekunden weniger Zeit zur Fragmentextraktion. Auch die Anzahl getesteter Klassenausdrücke war bei einer maximalen Ausführungszeit des Lernalgorithmus von drei Sekunden bei einer durchschnittlich getesteten Anzahl von 97.400 Ausdrücken um etwa 24% höher als bei der Verwendung des Endpoints von DBTune.

Diese Differenz hat wahrscheinlich mehrere Faktoren. Zum einen ist es möglich, dass der SPARQL-Endpoint von DBTune zeitweise ausgelastet ist und die Anfragen damit nur langsam abarbeiten kann. Ein weiterer Grund könnte die in den Tests verwendete Internetanbindung sein, die sich sowohl auf die Antwortzeiten als auch auf die Übertragungsgeschwindigkeit der Daten auswirken kann.

Ein Vergleich zu anderen RDF-Speichersystemen mit SPARQL-Endpoint-Implementation wie zum Beispiel Jena/Joseki[96] oder Sesame[97] entfällt damit an dieser Stelle, da mit diesem Performancetest bereits gezeigt werden konnte, dass der Einsatz eines lokalen SPARQL-Endpoints für ein produktives System

[96] http://jena.sourceforge.net/
[97] http://www.openrdf.org/

sinnvoll sein kann. Für einen Performancevergleich der unterschiedlichen verfügbaren Speichersysteme sei auf Becker (2008) und Bizer und Schultz (2009) verwiesen.

5.3 Ergebnisrelevanz

Die nächste Untersuchung beschäftigte sich mit der Relevanz der durch den Lernalgorithmus gefundenen Ergebnisse für den Benutzer. Bedingt durch die statistische Verteilung der Tags sind nach einer längeren Benutzung der Webapplikation zwei Fälle zu beobachten:

1. Durch die Häufigkeit der meistverwendeten Tags wie zum Beispiel ‘rock’ oder ‘electro’ kommt es oft vor, dass die generierten Vorschläge aus einer Kombination nur dieser Tags bestehen. Wie in Kapitel 4.2, ab Seite 42, beschrieben, ist durchschnittlich jedes Album mit einem der vier am häufigsten auftretenden Tags verknüpft, so dass ein Klassenausdruck wie „electro OR rock“ für eine Vielzahl von positiven Beispielen zufriedenstellend ist.
2. Werden nur spezifische, eher selten vorkommende Stilrichtungen oder Genres vom Benutzer bevorzugt, so sind die daraus entstehenden Resultate meist wieder Klassenausdrücke, die aus den seltenen Tags bestehen. Dies konnte mit den gewählten Beispielen des in Kapitel 5.1, ab Seite 57, verwendeten Skripts beobachtet werden.

Im ersten Fall kann es vorkommen, dass der semantische Zusammenhang zwischen gehörten und vorgeschlagenen Alben für den Benutzer der Webapplikation nicht unbedingt nachvollziehbar ist. Das liegt daran, dass Tags wie ‘rock’ oder ‘electro’ viel zu vage formuliert sind und eine große Anzahl an Subgenres beherbergen, die zumindest nach dem subjektiven Empfinden eines Benutzer nicht untereinander korrelieren.

Im zweiten Fall ist die Relevanz der Ergebnisse für den Benutzer meist relativ hoch, da dieser sozusagen „im Genre bleibt“. Dabei kann es allerdings häufig zu immer den gleichen Vorschlägen kommen. Ein weiteres Problem in diesem Fall ist, dass wenn der Benutzer nun Musikstücke eines völlig anderen Genres durch Hören zu den positiven Beispielen hinzufügt, keine hinreichend zufriedenstellende Klassenausdrücke mehr berechnet werden können.

Lösungsansätze zu diesen und weiteren Problemen der vorliegenden Arbeit werden im folgenden Kapitel erläutert.

5.4 Optimierbarkeit

Wie sich in der Evaluation gezeigt hat, besteht der größte Optimierungsbedarf an den bereitgestellten Daten selbst. Die Verwendung einzelner Worte als Tags ist zur Bezeichnung von Genres mit Problemen behaftet, die Einführung von Ressourcen wie `http://dbtune.org/jamendo/tag/indie+pop` oder `http://dbtune.org/jamendo/tag/death+metal` könnte dieses Problem zumindest in Ansätzen lösen. Die von der Musicontology definierte Klasse `mo:Genre` könnte dafür explizit eingesetzt werden, diese taucht allerdings in den verwendeten RDF-Daten an keiner Stelle auf. Dies würde die Relevanz der vom DL-Learner berechneten Vorschläge erheblich verbessern. Wünschenswert wäre ebenso eine Erweiterung der Daten mit Klassen oder Relationen wie `mo:biography` oder `mo:Lyrics` um dem Benutzer nützliche Zusatzinformationen über den Künstler oder aktuell gespielte Titel anzeigen zu können.

Eine weitere Möglichkeit zur Lösung der in Kapitel 5.3, ab Seite 66, angesprochenen Probleme ließe sich dadurch erreichen, für Alben mit wenig aussagekräftigen Tags wie ‘hard’, die sowohl ‘rock’ als auch ‘core’ zugeordnet werden können, eine automatische Untersuchung der weiteren vergebenen Tags, und der Korrelation dieser zueinander,

vorgenommen wird. Durch eine Gewichtung dieser Tags würden sich daraufhin die einzelnen Alben besser einem Genre zuordnen lassen, wodurch bessere Empfehlungen unterbreitet werden könnten. Die Programmierung einer solchen Funktionalität würde jedoch den Rahmen dieser Arbeit sprengen.

Hinsichtlich der Performance und der Ergebnisqualität der verwendeten Algorithmen und Komponenten des DL-Learners ist im Laufe der Arbeit kein Optimierungsbedarf entstanden. Die Webapplikation selbst lässt sich jedoch durchaus noch erweitern. Mögliche Verbesserungsmöglichkeiten wären zum Beispiel der Einsatz von *RDFa* (Resource Description Framework - in - attributes), so dass die semantischen Hintergrundinformationen direkt im Browser eingebettet werden können. Des Weiteren wäre eine Erweiterung der Applikation nach den Richtlinien barrierefreier Webentwicklung möglich[98], so dass diese einem größeren Benutzerkreis zugänglich wären. Auch eine Suche nach Herkunftsort des Künstlers über die bereitgestellten Informationen von Geonames könnte für den Benutzer eine Bereicherung darstellen.

Die Ontologie könnte außerdem mit den bisher vernachlässigten Tags[99] erweitert werden, so dass die generierten Vorschläge unter Umständen für den Benutzer relevanter werden könnten.

5.5 Zusammenfassung

Die Untersuchung der Webapplikation auf Performance und Ergebnisgenauigkeit hat positive Ergebnisse aufgezeigt. Die vom DL-Learner verwendeten Algorithmen zur Generierung der Vorschläge haben sich als eine solide Grundlage erwiesen, eine Optimierung dieser ist für die entstandene Applikation nicht nötig. Anders verhält es sich jedoch mit den verwendeten Daten, die durchaus noch optimiert werden können.

[98] Vergleiche http://www.w3.org/TR/WCAG/.
[99] Zum Beispiel solche, die weniger als vier mal vorkommen.

6 Fazit

Dieses Kapitel fasst die in dieser Arbeit erreichten Ergebnisse zusammen und beleuchtet sie kritisch. Es wird ein Vergleich mit anderen Arbeiten herangezogen um diese Arbeit besser in den aktuellen Stand der Forschung einordnen zu können.

6.1 Zusammenfassung und Vergleich mit anderen Arbeiten

Es gibt zahlreiche Plattformen, die es sich zur Aufgabe gemacht haben, Musikliebhabern auf Basis ihres persönlichen Geschmacks Hörempfehlungen zu generieren, wie zum Beispiel last.fm oder die relativ junge Plattform spotify[100]. Diese stellen ihre Daten und das damit verbundene Wissen jedoch nicht durch Verwendung der Technologien des Semantic Web, sondern durch eigene Schnittstellen bereit. Dies macht ein maschinelles Lernen weitgehend unmöglich. Die Musikempfehlungen, die diese Plattformen generieren, sind in den meisten Fällen nicht direkt abspielbar, da diese den Lizenzrechten des jeweiligen Künstlers oder Labels unterliegen.

Die in dieser Arbeit entstandene Webapplikation entspricht zwar im Umfang nicht den oben genannten Plattformen, grenzt sich aber dadurch ab, dass die Empfehlungen durch maschinelles Lernen mit Hilfe einer Ontologie und Techniken des Semantic Web erstellt werden. Außerdem sind die verwendeten Musikdaten alle unter einer OpenSource-Lizenz veröffentlicht worden, so dass nicht nur Vorschläge generiert werden, sondern diese dem Benutzer auch direkt zur Verfügung gestellt werden können.

Die Webapplikation legt dabei großen Fokus auf Bedienbarkeit durch ein simples Interface. Unter Einsatz aktueller Webstandards konnten die Reaktions- und Seitenaufbauzeiten minimiert werden. Die konsequente Benutzung von Webstandards hat

[100] http://www.spotify.com/

außerdem den Vorteil, dass eine saubere Trennung zwischen Layout, Struktur und Funktionalität gewährleistet werden kann.
Die Applikation gliedert sich in vier Teile: das Frontend auf Clientseite, das über eine AJAX-Schnittstelle asynchron mit der objektorientierten PHP-Applikation auf Serverseite kommuniziert, die für die Auswertung aller Anfragen und Datenaufbereitung verantwortlich ist; der DL-Learner, der Komponenten und Algorithmen zur Suche per SPARQL und zum Generieren der Vorschläge bietet und letztlich die externen Wissensquellen von DBTune und die Musikdaten von Jamendo.

Für die Generierung der Musikempfehlungen wurde eine Ontologie erstellt, die in den RDF-Daten häufig benutzte, sinnvolle Tags zueinander in Beziehung setzt. Um eine Eingliederung der Tags in die vier gefundenen Klassen Musikgenre, Musikinstrumente, Herkunft und Stimmung zu realisieren, wurden weitere Ontologien hinzugezogen. Als wichtigste sei hierbei die Musicontology genannt. Diese bildet auch die Basis für die SPARQL-Anfragen, wobei diese wiederum in literale und ressourcenbezogene Anfragen für Tags getrennt wurden.

Das den Musikempfehlungen zugrunde liegende Lernproblem besteht darin, bis zu zehn vom Benutzer gehörte Stücke als positive Beispiele zu verwenden und auf Basis dieser und der erstellten Ontologie zufriedenstellende Ergebnisse mit Hilfe des CELOE-Algorithmus zu berechnen. Aus diesen werden dann Vorschläge generiert, aus denen der Benutzer Musikstücke zum direkten Abspielen im Browser auswählen kann. In einer Evaluation wurde die Performance der verwendeten Algorithmen zur Fragementextraktion und zum Lernen untersucht. Diese wurden dann so konfiguriert, dass die sich ergebende Laufzeit so niedrig wie möglich und in diesem Fall sogar konstant gehalten werden konnte.

Im Bereich der Forschung gibt es einige Arbeiten, die sich mit dem Generieren von Musikempfehlungen beschäftigen. Celma (2008) behandelt unter anderem das Problem, auf welcher Basis solche Empfehlungen Sinn machen und wie unterschiedliche Be-

nutzer gleichartige Empfehlungen bewerten. Die Plattform, die durch die Arbeit von Celma (2008) entstanden ist[101], bietet die Möglichkeit, Wissen aus unterschiedlichen Quellen zu vereinen und verwendet dazu Technologien des Semantic Web. Allerdings fehlt auch hier die Möglichkeit, die generierten Vorschläge direkt anzuhören.

Neben diesen rein semantischen Ansätzen gibt es außerdem technische Ansätze, die versuchen, durch die Extraktion des Audiosignals eines Musikstücks dieses zu klassifizieren und daraufhin Vorschläge zu generieren. Die Arbeit von Abdallah et al. (2006) verwendet zudem weitere semantisch verknüpfte Hintergrundinformationen, um die Empfehlungsqualität zu verbessern. In der Praxis ist dieser Ansatz, zumindest als Webapplikation, nicht ohne weiteres umsetzbar, da die Auswertung des Audiosignals eine erhöhte Laufzeit erfordert.

Vielversprechende Konzepte liefern Song et al. (2009) und Passant und Raimond (2008), die versuchen die Vorteile einer auf sozialem Wissen basierenden Plattform, wie zum Beispiel last.fm, mit vorhandenem Wissen und den Technologien des Semantic Web zu vereinen. Projekte wie DBTune oder die Musicontology bilden dabei wichtige Grundlagen und liefern ausreichende Informationen, weshalb sie auch in dieser Arbeit eine wichtige Rolle spielen.

Die von DBTune bereitgestellten Daten sind jedoch mit Problemen behaftet. Eines der größten Probleme dieser Arbeit war es, die Tags, die nur aus einem Wort bestehen, in der Ontologie sinnvoll einzuordnen. Diese Tags verursachen auch Probleme bei der Relevanz der gefundenen Musikempfehlungen, da es durch deren statistische Verteilung zu ungeneuen oder viel zu speziellen Empfehlungen kommen kann. Eine Lösung für diese Probleme konnte in der Arbeit nicht vorgestellt werden, es wurde aber eine Erweiterung der Daten um zusätzliche Informationen vorgeschlagen, mit denen sich die Relevanz der Ergebnisse verbessern lassen würde.

[101] http://foafdevel.searchsounds.net

6.2 Schlusswort

Die Entwicklung des Semantic Web schreitet stetig voran und wird bereits in vielen Bereichen eingesetzt[102] und wird somit wohl auch weiterhin eine wichtige Rolle im Web der Zukunft einnehmen. Das Semantic Web eignet sich durch seine Grundsätze hervorragend dazu, Informationen zu Musik und bestimmten Künstlern zu suchen. Durch die verwendeten eindeutigen Bezeichner und Ressourcen ist es möglich, Wissen aus verschiedenen Quellen zu vereinen. Mit diesem Wissen und den geeigneten Programmen und Schnittstellen, die in dieser Arbeit vorgestellt wurden, lässt sich neues Wissen ableiten, woraufhin Empfehlungen generiert werden können. Allerdings ist die Qualität dieser Empfehlungen stark von den vorhandenen Daten abhängig.
Plattformen wie Jamendo erfreuen sich eines stetigen Zuwachses, da mittlerweile viele Musiker neue Wege beschreiten und ihre Musik kostenlos im Internet zur Verfügung stellen. Die Kombination aus frei verfügbarem Wissen und Musik eröffnet damit die Möglichkeit, Benutzern Musikvorschläge nicht nur zu unterbreiten, sondern auch gleich das passende Musikstück zum Download oder Hören anzubieten.

Im praktischen Teil der Arbeit ist eine benutzbare Webapplikation entstanden, die es möglich macht, die frei verfügbare Musik von Jamendo in einem personalisierten Radiostream zu genießen und sich gleichzeitig über die Künstler zu informieren. Eine Webapplikation wie diese ist jedoch in den meisten Fällen „work in progress“ und wie bereits in Kapitel 5.4, ab Seite 67, erwähnt worden ist, gibt es durchaus noch Verbesserungs- und Erweiterungsmöglichkeiten. Durch das objektorientierte Konzept der Arbeit lässt sich diese so auch in der Zukunft leicht erweitern. Eigens dafür wurde die Domain moosique.net registriert. Nach erfolgreicher Konfiguration des zugrunde liegenden Webservers wird die aus dieser Diplomarbeit entstandene Webapplikation dort ihr berechtigtes Zuhause finden.

[102] http://www.w3.org/2001/sw/sweo/public/UseCases/

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CELOE Class Expression Learning for Ontology Engineering

CSS3 Cascading Style Sheets level 3

DC Dublin Core

DTD Document Type Definition

DOM Document Object Model

FOAF Friend of a Friend

HTML Hyper Text Markup Language

JSON JavaScript Object Notation

JAX-WS Java API for XML Web Services

MP3 MPEG-1 Audio Layer 3

N3 Notation 3

OWL Web Ontology Language

PHP PHP: Hypertext Preprocessor

RDF Resource Description Framework

RDFa Resource Description Framework - in - attributes

RDFS RDF Schema

SGML Standard Generalized Markup Language

SPARQL SPARQL Protocol and RDF Query Language

SQL Structured Query Language

Turtle Terse RDF Triple Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

W3C World Wide Web Consortium

XML eXtensible Markup Language

XPATH XML Path Language

XSD XML Schema Definition

XSLT eXtensible Stylesheet Language Transformation

XSPF XML Shareable Playlist Format

Literatur

Samer A. Abdallah, Yves Raimond, und Mark Sandler. An ontology-based approach to information management for music analysis systems. Technical report, Audio Engineering Society, May 2006.

Christian Becker. *RDF Store Benchmarks with DBpedia*. Freie Universität Berlin, Januar 2008. http://www4.wiwiss.fu-berlin.de/benchmarks-200801/.

David Beckett und Tim Berners-Lee. *Turtle - Terse RDF Triple Language*. W3C, Januar 2008. http://www.w3.org/TeamSubmission/turtle/.

Tim Berners-Lee. Linked data - design issues. Juni 2009. http://www.w3.org/DesignIssues/LinkedData.html.

Tim Berners-Lee. *Notation 3 - A readable language for data on the Web*, März 2006. http://www.w3.org/DesignIssues/Notation3.html.

Tim Berners-Lee, James Hendler, und Ora Lassila. The semantic web. *Scientific American*, Mai 2001. http://scientificamerican.com/article.cfm?id=the-semantic-web.

Tim Berners-Lee, Roy T. Fielding, und Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. W3C/MIT and Day Software and Adobe Systems, Januar 2005. http://tools.ietf.org/html/rfc3986section-3.1.

Chris Bizer und Andreas Schultz. *Berlin SPARQL Benchmark Results*. Freie Universität Berlin, März 2009. http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/index.html.

Dan Brickley und Libby Miller. Foaf vocabulary specification 0.91. 11 2007. http://xmlns.com/foaf/spec/20071002.html.

Òscar Celma. *Music Recommendation and Discovery in the Long Tail*. PhD thesis,

Universitat Pompeu Fabra, Barcelona, Spain, 2008. URL http://mtg.upf.edu/~ocelma/PhD/doc/ocelma-thesis.pdf.

James Craig, Michael Cooper, Lisa Pappas, Rich Schwerdtfeger, und Lisa Seeman. *Accessible Rich Internet Applications (WAI-ARIA) 1.0*. W3C, Februar 2009. http://www.w3.org/TR/wai-aria/.

Cristian Darie, Bogdan Brinzarea, Filip Chereches-Tosa, und Mihai Bucica. *AJAX and PHP: Building Responsive Web Applications*. Packt Publishing, Oktober 2006.

Jesse James Garrett. Ajax: A new approach to web applications. *Adaptive Path*, Februar 2005. http://www.adaptivepath.com/ideas/essays/archives/000385.php.

Frédérick Giasson und Yves Raimond. Music ontology specification. 2 2007. http://purl.org/ontology/mo/.

Sebastian Hellmann, Jens Lehmann, und Sören Auer. Learning of owl class descriptions on very large knowledge bases. *International Journal On Semantic Web and Information Systems*, 2009.

Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, und York Sure. *Semantic Web: Grundlagen*. Springer, Berlin, Oktober 2007.

Jeremy Keith. *Bulletproof Ajax*. New Riders, illustrated edition edition, Februar 2007.

Graham Klyne und Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*, Februar 2004. http://www.w3.org/TR/rdf-concepts/.

Stefan Kokkelink und Roland Schwänzl. *Expressing Qualified Dublin Core in RDF / XML*, Mai 2002. http://dublincore.org/documents/2002/05/15/dcq-rdf-xml/.

Peter Lavin. *Object-Oriented PHP - Concepts, Techniques, And Code*. William Pollock, No Starch Press San Fransisco, 2006.

Jens Lehmann. *DL-Learner Manual*. Universität Leipzig, April 2009.

Jens Lehmann und Pascal Hitzler. A refinement operator based learning algorithm for the alc description logic. In Hendrick Blockeel, Jude W. Shavlik, und Prasad

Tadepalli, editors, *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2008.

Boris Motik, Peter F. Patel-Schneider, und Bijan Parsia. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. W3C, Oktober 2009. http://www.w3.org/TR/owl-syntax/.

Senthil Nathan, Edward J Pring, und John Morar. Convert xml to json in php. *IBM developerWorks*, Januar 2007. http://www.ibm.com/developerworks/xml/library/x-xml2jsonphp/.

Richard Newman. *Tag ontology*, Dezember 2005. http://www.holygoat.co.uk/owl/redwood/0.1/tags/.

Aaron Newton. *MooTools Essentials: The Official MooTools Reference for JavaScript^{TM} and Ajax Development*. Apress, September 2008.

Natalya F. Noy und Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University, Stanford, CA, 94305, 2000.

François Pachet und Daniel Cazaly. A taxonomy of musical genres. April 2000.

Alexandre Passant und Yves Raimond. Combining social music and semantic web for music-related recommender systems. *CEUR-ws.org, Vol. 405*, Oktober 2008. http://apassant.net/publications.

Valerio Proietti et al. *MooTools Docs*, 2009. http://mootools.net/docs/.

Eric Prud'hommeaux und Andy Seaborne. *SPARQL Query Language for RDF*, November 2007. http://www.w3.org/TR/rdf-sparql-query/.

Yves Raimond, Samer Abdallah, Mark Sandler, und Frederick Giasson. The music ontology. *Austrian Computer Society (OCG).*, page 6, 2007.

Matthias Schütz. *JavaScript Framework Matrix*, September 2009. http://matthiasschuetz.com/javascript-framework-matrix/de/.

Dave Shea. Css sprites: Image slicing's kiss of death. *A List Apart*, No. 173, März 2004. http://www.alistapart.com/articles/sprites.

Seheon Song, Minkoo Kim, Seungmin Rho, und Eenjun Hwang. Music ontology for mood and situation reasoning to support music retrieval and recommendation. pages 304–309, Februar 2009. http://dx.doi.org/10.1109/ICDS.2009.50.

Erich Valentin, Franz A. Stein, und Christine Weiss. *Handbuch der Musikinstrumentenkunde*. Bosse, (gebundene, völlig neu erarbeitete ausgabe) edition, März 2004.

Dr. Thomas Wirth. *Missing Links - Über gutes Webdesign*. Carl Hanser Verlag München Wien, 2., erweiterte auflage edition, 2004.

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

Danksagung

Bedanken möchte ich mich bei meinem Betreuer Jens Lehmann, der mich im Laufe der Arbeit immer tatkräftig unterstützt hat. Ohne ihn wäre diese Arbeit wohl nie zustande gekommen. Für seelische Unterstützung und Korrekturlesen möchte ich meinen guten Freunden und Mitbewohnern Julian, Florian und Elisa danken. Außerdem danke ich meiner Familie für die ausdauernde finanzielle und geistige Unterstützung, die mir dieses Studium ermöglicht hat. Besonderen Dank gebührt meiner Freundin Antje, die mich stets motiviert hat und mir immer mit der nötigen Geduld zur Seite stand.

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich diese Diplomarbeit selbstständig ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe.

Diese Arbeit ist bislang keiner anderen Prüfungsbehörde vorgelegt und auch nicht auf sonstigem Wege veröffentlicht worden. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Steffen Becker