OTTO-VON-GUERICKE-UNIVERSITÄT MAGDEBURG

FAKULTÄT FÜR INFORMATIK
Institut für Wissens- und Sprachverarbeitung

Diplomarbeit

Ontologien zur semantischen Suche in einem Bestand von Dokumenten

Mirko Otto
26. September 2008

Betreuer
Prof. Dr. Dietmar Rösner

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Universitätsplatz 2
39106 Magdeburg

Abstract

Bestehende Informationssysteme verwalten ihre Daten meist in einer hierarchieartigen Struktur. Wachsende Datenbestände und eine unzureichende Strukturierung machen es dem Nutzer zunehmend schwerer, die gewünschten Informationen zu ermitteln bzw. den Überblick über die Daten zu behalten.

Diese Diplomarbeit beschreibt die Entwicklung eines Konzeptes zur Nutzung von Ontologien in einem Dokumentenbestand. Insbesondere werden dabei die Möglichkeiten für eine semantische Suche herausgestellt. Dafür werden vorhandene E-Learning-Ressourcen aus einem Content Management System auf eine zu entwickelnde Ontologie abgebildet.

Inhaltsverzeichnis

1 Einleitung **17**

1.1 Motivation 17

1.2 Aufbau der Arbeit 18

I Grundlagen **21**

2 Metadaten **23**

2.1 Metadatentypen 24

2.2 Metadatenframework 25

2.3 Metadatenstandards 25

2.3.1 Dublin Core 26

2.3.2 LOM 27

3 Semantic Web **29**

3.1 URI & IRI 30

3.2 Extensible Markup Language 31

3.3 Resource Description Framework 31

3.3.1 RDF-Schema 33

3.4 Anfragesprachen (Query) 33

3.5 Wissensverarbeitung (Logic) 34

3.5.1 Regeln (Rule) 34

3.6 Automatische Beweisführung (Proof) 35

3.7 Vertrauen (Trust) 36

4 Ontologien **37**

4.1 Ontologiebeschreibungssprachen 38

4.1.1 OWL 38

4.2 Entwicklung von Ontologien 43
4.3 Werkzeuge zur Ontologieentwicklung 44
4.3.1 Ontologieframeworks für die Ontologieentwicklung . 45

5 Technologien der semantischen Suche 47
5.1 Suchdienste 47
5.1.1 Human Directed Search 47
5.1.2 Automated Search 48
5.1.3 Meta-Suchmaschinen 49
5.2 Grenzen heutiger Suchtechnologien 49
5.3 Die Rolle der Semantik 50
5.4 Anfragetechniken 51
5.4.1 Semantic Search 51
5.4.2 Semantic Browsing 51
5.5 SPARQL 52
5.5.1 Einfache Graph-Pattern 54
5.5.2 Gruppierende Graph-Pattern 54
5.5.3 Optionale Graph-Pattern 54
5.5.4 Alternative Graph-Pattern 54
5.5.5 Benannte Graph-Pattern 54
5.5.6 Filter 54
5.5.7 Modifikatoren 55
5.6 Faceted Browsing 56

II $PlOSS^X$ 59

6 Konzept 61
6.1 Vorgehensweise 61
6.2 Gesamtkonzept 61
6.3 Beschreibung des Anforderungskontextes 62
6.3.1 Eigenschaften eines ontologiebasierten Anfragesystems 62
6.3.2 Anforderungen 64
6.4 Verwandte Ansätze 64
6.4.1 PloneCollections 65
6.4.2 PloneOntology 65
6.4.3 ContentLicensing 65

7 Erstellung der Ontologie 67
7.1 Wiederverwendung von Ontologien 68
7.2 LOM-Ontologie 68

7.3 CS-Ontologie 69
7.4 CMS-Ontologie 71
7.4.1 ContentTypes 72
7.4.2 Agent 73
7.4.3 DomainConcept 74
7.4.4 Metadata 74
7.5 Erstellen einer Wissensbasis 74
7.6 Anforderungen an eine $PlOSS^X$-konforme Ontologie 75

8 Semantische Suchanfragen 77
8.1 Anfrageform 77
8.2 Negation 78
8.3 SPARQL 78
8.3.1 SPARQL-Prolog 78
8.3.2 Einfache Anfrage 78
8.3.3 Oder-Verknüpfung 79
8.3.4 Und-Verknüpfung 80
8.3.5 Komplexe Anfragen 80
8.3.6 Negation in SPARQL 81
8.3.7 Negation by divide 82
8.3.8 Beschränkungen von SPARQL 83
8.4 Faceted Browsing 83
8.5 Konjunktive Anfragen und Regeln 85

III Zusammenfassung und Ausblick 89

9 Zusammenfassung 91
9.1 Einordnung von $PlOSS^X$ 92
9.2 Abdeckung der Anforderungen 92

10 Ausblick 95

A Taxonomie der CMS-Ontologie 97

B Instanzen der Ontologien 99

C Software zur Arbeit 103

Abbildungsverzeichnis

3.1 Schichtenmodell der Semantic Web Architektur 30
3.2 Grafische Darstellung einer Aussage in RDF-Grammatik . . . 32
4.1 OWL Sprachebenen 42
7.1 Import-Relationen (owl:imports) der Teilontologien 68
7.2 Taxonomie der LOM-Ontologie 69
7.3 Taxonomie der CS-Ontologie 70
7.4 Taxonomie der CMS-Ontologie 72
7.5 Taxonomie der Superklassen der CMS-Ontologie 72
8.1 Faceted Browsing 84
8.2 Faceted Browsing Selected 85
9.1 Architektur von $PlOSS^X$ 91
A.1 Komplette Taxonomie der CMS-Ontologie 98

Tabellenverzeichnis

7.1 Eigenschaften der Klasse ContentTypes 73
7.2 Eigenschaften der Klasse Person 73
7.3 Eigenschaft der Klasse ComputerScience 74
7.4 Eigenschaften der Klasse LOM 74
B.1 Instanzen der CS-Ontologie 99
B.2 Instanzen der Inhaltstypen der CMS-Ontologie 101

Listings

3.1 Beispiel einer Aussage in XML-Syntax 32
4.1 XML-Namespace Deklaration und OWL-Header Definition . . 40
4.2 OWL-Class Definition 40
4.3 OWL-Property Definition 40
4.4 OWL-Individual Definition 41
5.1 einfache SPARQL-Anfrage 53
5.2 RDF Daten 53
5.3 Ergebnis der einfachen SPARQL-Anfrage 53
5.4 SPARQL-Anfrage mit Filter 55
5.5 Ergebnis der SPARQL-Anfrage mit Filter 55
8.1 Präfixe der verwendeten Namespaces 78
8.2 Subklassen von ConentTypes 78
8.3 Ergebnismenge: Subklassen von ConentTypes ohne Reasoner . 79
8.4 Ergebnismenge: Subklassen von ConentTypes mit Reasoner . 79
8.5 Anfrage mit Oder-Verknüpfung 79
8.6 Ergebnismenge der Anfrage mit Oder-Verknüpfung 80
8.7 Anfrage mit Und-Verknüpfung 80
8.8 Ergebnismenge der Anfrage mit Und-Verknüpfung 80
8.9 Anfrage mit Oder-Und-Verknüpfung 81
8.10 Ergebnismenge der Anfrage mit Oder-Und-Verknüpfung . . . 81
8.11 Anfrage nach Inhaltstypen mit FP-Konzepten 81
8.12 Ergebnismenge der Anfrage nach Inhaltstypen mit FP-Konzepten 82
8.13 Negation As Failure 82
8.14 Negation by divide 83
8.15 Negation by divide - Ergebnismengen 83
8.16 Anfrage nach Anwendung einer SWRL-Regel 86

8.17 SWRL-Regel 87

Abkürzungsverzeichnis

CMS	Content Management System
DC	Dublin Core
DCMI	Dublin Core Metadata Initiative
DTD	Document Type Definition
IEEE	Institute of Electrical and Electronics Engineers
IRI	Internationalized Resource Identifiers
LOM	Learning Objects Metadata
OWA	Open World Assumption
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF-Schema
RIF WG	Rule Interchange Format Working Group
SGML	Standard Generalized Markup Language
SPARQL	SPARQL Protocol and RDF Query Language
SW	Semantic Web
SWRL	Semantic Web Rules Language
UID	Unique ID
URI	Uniform Resource Identifier
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

1 Einleitung

Heutige Informationssysteme verwalten ihre Daten meist in modernen Datenbanksystemen. Wachsende Datenbestände und eine unzureichende Strukturierung machen es dem Nutzer zunehmend schwerer die gewünschten Informationen zu ermitteln bzw. den Überblick über die Daten zu behalten. Typischerweise besitzen Datenbanksysteme eine Indexsuche die eine Suche über Schlüsselwörter bereitstellt. Aber beispielsweise die Suche nach Informationen über den Ersteller eines Dokumentes wird keine Antwort liefern können, da hierfür Hintergrundwissen über das Dokument notwendig ist.

Soll dieses Wissen effektiv eingesetzt und genutzt werden, ist es nicht ausreichend, es digital, aber unstrukturiert bereitzustellen. Vielmehr werden Methoden zur Strukturierung von Wissen benötigt. Die Schlüsselrolle soll hierfür der Einsatz einer formalen Wissensbeschreibung – eine sog. *Ontologie* – einnehmen. Sie bietet sich als ein natürliches Mittel an, um Wissen strukturiert mittels Informationstechnologie bereitzustellen.

Weltweites Interesse finden Ontologien in der heutigen Zeit aufgrund der Semantic Web Initiative des WWW-Schöpfers Tim Berners-Lee (vgl. [BLF99]). Sie beruht auf der Grundidee, Web-Dokumente mit „Semantik“ in Form von Metadaten zu versehen, die ihren Inhalt näher beschreiben, und diese durch Ableitungsregeln miteinander zu verknüpfen. Damit sollen Suchmaschinen und andere Anwendungen unterstützt werden, benötigte Informationen gezielt und effizient zu finden und miteinander zu verbinden.

Aber auch beinahe 10 Jahre nach der Idee des Semantic Web bleibt eine semantische Suche in einem *Content Management System (CMS)* nur Vision.

1.1 Motivation

Seit dem Wintersemester 2003/2004 werden von der AG *Wissensbasierte Systeme und Dokumentverarbeitung (WDOK)*[1] im Institut für Wissens- und Sprachverarbeitung an der Otto-von-Guericke-Universität Magdeburg multimediale Lehrangebote in einer E-Learning-Umgebung in der universitären Lehre eingesetzt. Die Bereitstellung dieser Lehr- und Lernmaterialien erfolgt in einem Content Management System.

1. `http://wdok.cs.uni-magdeburg.de/` (Abruf: 26. September 2008)

In dieser Arbeit wird untersucht, welche Möglichkeiten es gibt, diese multimedialen Lehrangebote um ontologiebasierte Metadaten zu erweitern und wie eine semantische Suche auf Basis dieser Metadaten erfolgt. Des Weiteren wird gezeigt, wie eine semantische Suche mit einem Content Management System zusammengeführt werden kann. Die sich daraus ergebenden Erkenntnisse zum Aufbau einer semantischen Suchtechnologie werden in einer möglichst abstrakten Ontologiestruktur und in einem Konzept eines semantischen Anfragesystems umgesetzt. Anhand von Anwendungsfällen wird erläutert, wie semantische Suchanfragen mit den Technologien der Semantic Web Architektur des W3C realisiert werden können.

1.2 Aufbau der Arbeit

In Teil I werden die theoretischen Grundlagen der behandelten Themenbereiche vorgestellt.

Zu Beginn geht Kapitel 2 auf Metadaten ein und stellt bestehende Konzepte und Standards vor.

Kapitel 3 gibt einen Überblick über die Technologien des Semantic Web. Die Schwerpunkte liegen dabei auf den Standards XML, RDF und RDF-Schema, wobei die Abschnitte zu den einzelnen Standards entsprechende Handbücher nicht ersetzen sollen. Vielmehr sollen dort die Zusammenhänge zwischen den verschiedenen Standards gezeigt werden.

In Kapitel 4 werden Ontologien und deren Bestandteile erklärt. In diesem Zusammenhang wird die zur formalen Ausgestaltung einer Ontologie einsetzbare Ontologiesprache OWL vorgestellt. Darauf folgend wird ein mögliches Konzept für die Entwicklung einer Ontologie gezeigt. Der letzte Abschnitt des Kapitels widmet sich den verschiedenen Werkzeugen zur Ontologieentwicklung.

Das Kapitel 5 gibt einen Überblick über die Technologien der semantischen Suche. Zunächst werden vorhandene Konzepte von Suchdiensten behandelt und deren prinzipielle Funktionsweise erläutert. Im Anschluss daran werden die Grenzen heutiger Suchtechnologien aufgezeigt und welche Rolle die Semantik in diesem Zusammenhang spielt. Abschließend werden Anfragetechniken für eine semantische Suche bzw. eine semantische Navigation vorgestellt.

In Teil II erfolgt eine Beschreibung der konkreten Vorgehensweise bezüglich der Umsetzung der Problemstellung.

Zunächst wird in Kapitel 6 das Gesamtkonzept des zu entwickelnden semantischen Anfragesystems beschrieben. Anschließend folgt eine Beschreibung des Anforderungskontextes. Dies umfasst insbesondere die grundlegenden Eigenschaften eines ontologiebasierten Anfragesystems und die sich daraus ergebenen Anforderungen. Im letzten Abschnitt des Kapitels werden verwandte Ansätze betrachtet, die sich mit der Integration von Metainformationen in das CMS Plone auseinandersetzen.

Kapitel 7 beschreibt die Erstellung der Ontologiestruktur, welche dem hier zu entwickelnden semantischen Anfragesystem zugrunde liegt. Dabei wird insbesondere auf die Wiederverwendung von Ontologien eingegangen. Im Anschluss daran folgt die Entwicklung der erforderlichen Ontologien. Abschließend werden Anforderungen an eine dem Anfragesystem entsprechende Ontologie aufgezeigt.

In Kapitel 8 wird zunächst der Begriff „semantische Suche“ definiert, so wie dieser im Kontext dieser Arbeit verstanden werden soll. Im Anschluss daran folgt ein Überblick zu möglichen Anfrageformen an eine Ontologie im Rahmen der Semantic Web Architektur des W3C. Dabei wird insbesondere die Anfrageformulierung in der Anfragesprache SPARQL diskutiert und an Beispielen veranschaulicht. Weiterhin wird mit dem Konzept des Faceted Browsing ein visuelles Benutzerinterface zur Anfrageformulierung angeboten. Abschließend wird die Formulierung konjunktiver Anfragen mittels Regeln erläutert.

In Teil III werden Resümee und Ausblick zu dieser Arbeit gegeben.

Kapitel 9 fasst die erreichten Ergebnisse zusammen, ordnet den konzipierten Ansatz anhand von Merkmalen einer semantischen Suche ein und legt die Abdeckung der gestellten Anforderungen dar.

Zum Abschluss der Arbeit enthält Kapitel 10 einen Ausblick auf Aspekte, welche durch weiterführende Aktivitäten untersucht werden sollten.

Teil I

Grundlagen

2 Metadaten

Metadaten haben eine lange Tradition in den Informations- und Dokumentationswissenschaften sowie im Bibliothekswesen (vgl. [Sch04]).

In der Informationsgesellschaft sollen Metadaten dabei helfen, die jeweils relevante Information zu finden. Die Notwendigkeit von Metadaten ist ganz erheblich davon abhängig, wie die Suche ausgeführt werden soll. Das gezielte Suchen, z. B. nach einem bestimmten Titel eines Dokumentes, erfordert weniger Metadaten, als das strukturierte Suchen, z. B. nach einem Grundlagenwerk der Informatik. Soll eine Suche im zweiten Fall erfolgreich sein und nicht mehrere hundert Ergebnisse liefern, so müssen ausreichend Metadaten zur Verfügung stehen, die solche Dokumente inhaltlich und formal beschreiben.

Der Begriff „Metadaten“ selbst stammt aber aus dem Bereich der Informatik. Dort wird das Präfix „meta“ im Allgemeinen in der Bedeutung „about“ benutzt. So ist beispielsweise eine Metasprache eine Sprache, welche andere Sprachen beschreibt und Metadaten sind Daten zur Beschreibung von Daten. Die erste Publikation, welche den Begriff Metadaten im Sinn von Daten über Daten benutzte, war die erste Edition des NASA Directory Interchange Format Manual im Jahr 1988 (vgl. [Cap03] S. 1).

Nach [Cap03] ist es klar, dass es keine richtige oder falsche Interpretation des Metadatenbegriffs gibt. Wird der Begriff Metadaten aber genutzt, so sollte sichergestellt sein, wie er verstanden und in welchem Kontext er eingesetzt wird.

Die *Dublin Core Metadata Initiative* beschreibt Metadaten wie folgt:

> *„In general, "data about data;" functionally, "structured data about data." Metadata includes data associated with either an information system or an information object for purposes of description, administration, legal requirements, technical functionality, use and usage, and preservation. In the case of Dublin Core, information that expresses the intellectual content, intellectual property and/or instantiation characteristics of an information resource. . . . “*[1]

1. `http://dublincore.org/documents/usageguide/glossary.shtml` (Abruf: 26. September 2008)

Tim Berners-Lee, der Erfinder des World Wide Web, definiert in [BL97] Metadaten kurz und knapp als *„. . . machine understandable information about web resources or other things.“*

Ergänzend dazu definiert er die folgenden Axiome:

1. *„Metadata is data.“* und *„Metadata can describe metadata.“*
2. *„The architecture is of metadata represented as a set of independent assertions.“*

In [Sta02] beschreibt Staab Metadaten als Daten, die ausgewählte Aspekte anderer Daten beschreiben und unterscheidet nach formalen, semiformalen oder informalen Daten. So sind im typischen Fall einer Anwendung die Daten informal, z. B. Freitext, und die Metadaten semiformal, z. B. ein Dublin Core Feld für den Titel, oder formal, z. B. ein Dublin Core Feld für die Sprache einer Ressource, welches dann entsprechend der Konvention für Sprachbezeichner[2] interpretiert werden kann.

Schmidt weist in [Sch04] darauf hin, dass der Begriff Metadaten von der Betrachtungsweise und vom Standpunkt abhängt: So kann zwischen Objektdaten, d. h. Daten die die Ressource ausmachen, und Metadaten nicht immer klar unterschieden werden. So sind in einem Bibliothekskatalog Metadaten verzeichnet. Sie beschreiben die in der Bibliothek vorhanden Bücher und Publikationen. Für ein Programm, welches diese Metadaten verwaltet, werden diese Daten zu Objektdaten.

Daher definiert Schmidt Metadaten in [Sch04]

> *„. . . als alles, was über eine Informationsressource gesagt werden muss, damit diese unter einem bestimmten Aspekt vollständig repräsentiert ist. Unter einer Informationsressource ist dabei alles zu verstehen, was als Einheit von einem Menschen oder einer Maschine adressiert und verarbeitet werden kann.“*

2.1 Metadatentypen

Der weitreichende Gebrauch von Metadaten hat nach [Cap03] und [PRR04] zu einer Typologie von Metadaten in deskriptive, administrative und strukturelle Metadaten geführt. Diese Kategorien stehen nicht für die Qualität der Metadatenelemente selbst, da laut der Metadatendefinition sämtliche Metadaten etwas beschreiben, sondern für die funktionalen Aspekte der Metadaten.

- **Deskriptive** oder **beschreibende Metadaten** sollen dem Auffinden und der Identifizierung von Ressourcen dienen und liefern Antworten auf die Fragen **Wer**, **Was**, **Wann** und **Wo** in Bezug auf Ressourcen (vgl. [PRR04] S. 7).
- **Administrative Metadaten** sind Informationen, welche das Management von Ressourcen erleichtern sollen. Sie können Informationen enthalten, wie z. B. wann und wie wurde eine Ressource erstellt oder welche Zugriffsbeschränkungen für eine Ressource gelten. Dabei lässt sich

2. Tags for Identifying Languages – `ftp://ftp.rfc-editor.org/in-notes/rfc4646.txt` (Abruf: 26. September 2008)

eine Grenze zwischen deskriptiven und administrativen Metadaten nicht immer exakt ziehen und hängt oft von der Perspektive der Benutzer ab. Beispielsweise kann eine eindeutige Identifikationsnummer dem Administrator dabei helfen, die Zugriffsrechte auf die Ressource zu definieren. Die Identifikationsnummer könnte aber auch in den beschreibenden Metadaten dem Benutzer zur Identifikation der Ressource dienen.

- **Strukturelle Metadaten** können nach [Cap03] als der Kleber betrachtet werden, der verbundene Objekte zusammenhält. Sie werden z. B. benötigt, um die Verknüpfungen zwischen physikalischen Dateien und Seiten, zwischen Seiten und Kapiteln und zwischen Kapiteln und dem Buch als Ganzes auszudrücken. Der Unterschied zu deskriptiven Metadaten liegt darin, dass strukturelle Metadaten im Allgemeinen in der automatischen Verarbeitung genutzt werden.

2.2 Metadatenframework

Ein *Metadatenframework* ist eine allgemeingültige, an Regeln und Richtlinien orientierte Empfehlung. Mit einem Metadatenframework wird festgelegt, das und vor allem wie Daten möglichst plattformunabhängig erstellt und in andere Systeme importiert und integriert werden können.

Ein Metadatenframework umfasst dazu folgende Aspekte:

- **Semantik:** Die Semantik beschreibt die Bedeutung, die normalerweise von Normierungs-Gremien festgelegt wird (z. B. Dublin Core).

- **Datenmodell:** Das Datenmodell legt fest, welche Grammatik und Struktur die Metadaten besitzen können. Datenmodelle für Metadaten können beispielsweise einfache Attribut/Wert-Kombinationen oder Sätze mit Subjekt, Prädikat und Objekt (z. B. Tripel in RDF) sein.

- **Syntax:** Die Syntax dient dazu, die entsprechend dem Datenmodell generierten Aussagen zu repräsentieren (z. B. XML).

- **Identifikation:** Um Metadaten verschiedener Quellen sinnvoll verarbeiten zu können, muss eindeutig gekennzeichnet werden, um welche Semantik, welches Datenmodell und welche Syntax es sich handelt. Dafür ist ein Identifikationsmechanismus erforderlich, der orthogonal zu den anderen drei Schichten liegt.

Die Ausprägung eines konkreten Metadatenframework ist RDF, auf welches in Abschnitt 3.3 näher eingegangen wird.

2.3 Metadatenstandards

Durch den Einsatz von Metadatenstandards können Ressourcen einheitlich beschrieben werden. Die Verarbeitung – beispielsweise das Auffinden von Dokumenten – ist einfacher als die inhaltliche Verarbeitung von beschriebenen Dokumenten.

2.3.1 Dublin Core

Urheber der *Dublin Core (DC)* Spezifikation ist die *Dublin Core Metadata Initiative(DCMI)*. Die Ziele der DCMI sind die Entwicklung und Verbreitung von internationalen domänenunabhängigen Standards und Vokabulare für Metadaten (vgl. [Sta02]).

Dublin Core ist ein Standard, mit dem unterschiedlichste Informations-Ressourcen im Internet beschrieben werden können. Er besteht aus einer Menge von 15 Hauptelementen, der sogenannten Metadatenelementmenge *(Dublin Core Metadata Element Set, Version 1.1)*, die vom DCMI empfohlen wird (vgl. [DCM08b]). Die *DCMI Metadata Terms* empfehlen zusätzliche und detailliertere Felder, mit denen eine exaktere Beschreibung bzw. Kategorisierung der Ressource erfolgen kann (vgl. [DCM08a]). Alle Felder sind optional, können mehrfach auftreten und im Gegensatz zu anderen Metadatenstandards in beliebiger Reihenfolge stehen. Im Folgenden werden die 15 Metadatenelemente mit einer kurzen Erläuterung aufgeführt (vgl. [DCM08b]):

1. **contributor:** eine Person, eine Organisation oder ein Dienst, die sich an der Erstellung der Ressource beteiligt hat
2. **coverage:** das räumliche oder zeitliche Thema der Ressource, die räumliche Anwendbarkeit der Ressource oder der Rechtsraum, für den die Ressource gilt
3. **creator:** eine Person, eine Organisation oder ein Dienst, die wesentlich für die Erstellung der Ressource verantwortlich ist
4. **date:** Zeitpunkt oder eine Zeitspanne im Zusammenhang mit einem Ereignis im Entwicklungsprozess der Ressource
5. **description:** Beschreibung der Ressource
6. **format:** Dateiformat, der Datenträger oder der Umfang der Ressource
7. **identifier:** eine eindeutige Referenz auf die Ressource innerhalb eines gegebenen Kontexts
8. **language:** Sprache der Ressource
9. **publisher:** eine Person, eine Organisation oder ein Dienst, die für die Verfügbarkeit der Ressource verantwortlich ist
10. **relation:** eine verwandte Ressource
11. **rights:** Informationen über Rechte an der Ressource
12. **source:** eine verwandte Ressource, von der die beschriebene Ressource abgeleitet ist
13. **subject:** Thema der Ressource
14. **title:** Titel der Ressource
15. **type:** Art oder Gattung der Ressource

Das *DCMI Type Vocabulary*[3] liefert eine allgemeine und domainübergreifende Auflistung von geprüften Bezeichnungen, die als Werte für das *type*-Element der Metadatenelementmenge verwendet werden können.

Für Metadatenangaben zu einer Ressource liefern die Elemente eine erste Hilfe. Letztendlich sind sämtliche Dublin Core Elemente jedoch sehr allgemein gehalten, sodass für spezielle Zwecke, wie im Falle von Lernobjekten (*Learning Objects*), auf zusätzliche Metadatenschemata (z. B. LOM) zurückgegriffen werden muss, sollen die Metadaten so umfassend und zielgerichtet wie möglich angelegt werden.

2.3.2 LOM

Bei dem LOM-Standard handelt es sich um ein Metadatenschema (vgl. [Sch04]), das vom Learning Technology Standards Community des Institute of Electrical and Electronics Engineers (IEEE) entwickelt wurde. Der Standard wurde definiert, um die Suche, die Beschaffung, die Bewertung und die Nutzung von Learning Objects zu erleichtern.

Ein Learning Object ist eine in sich geschlossene digitale oder nicht-digitale Lerneinheit, die je nach Bedarf mit anderen Learning Objects kombiniert und in verschiedenen Kontexten mehrfach verwendet werden kann. Lernobjekte umfassen hauptsächlich Daten. Es sind kleine Informationseinheiten wie etwa HTML-Dokumente, Texte, Bilder, Präsentationen, Online- oder Präsenzkurse. Ziel einer modularen Aufbereitung von Inhalten durch Lernobjekte ist die Reduktion von Entwicklungskosten und -zeiten sowie die Personalisierung von Lernangeboten.

Um ein Learning Object zu beschreiben, stellt LOM neun Kategorien zur Verfügung:

1. **general:** Beschreibt allgemeine Metadaten wie beispielsweise Titel, Sprache, Beschreibung, Struktur und Granularität des Learning Objects.
2. **life cycle:** Beschreibt die Entwicklung und den aktuellen Zustand des Learning Objects mit Version, Status und Datum.
3. **meta-metadata:** Metadaten, welche die Metadaten des Learning Objects beschreiben – genauer: Die hier angegeben Metadaten beschreiben nicht das Learning Object selbst, sondern treffen nur Aussagen über die Metadaten des Learning Objects, z. B. wann und von wem die Metadaten in welcher Sprache erstellt wurden.
4. **technical:** Beinhaltet technische Voraussetzungen und Merkmale des Learning Objects, wie beispielsweise Format, Größe, Ort und Anforderungen an das Betriebssystem.
5. **educational:** Enthält Bildungsmerkmale und die pädagogische Beschreibung des Learning Objects, z. B. Zielgruppe, Interaktionstyp, semantische Dichte und Schwierigkeitsgrad.
6. **rights:** Enthält Angaben zu Nutzungsbedingungen, Copyright und Kosten.
7. **relation:** Beschreibt die Beziehungen zu anderen Learning Objects.

3. `http://dublincore.org/documents/dcmi-type-vocabulary/` (Abruf: 26. September 2008)

8. **annotation:** Erstellte Informationen von Nutzern, die das Learning Object verwendet haben.

9. **classification:** Ordnet das Learning Object in ein Klassifizierungssystem ein.

Die Datenelemente innerhalb einer Kategorie sind hierarchisch aufgebaut. Ein Datenelement kann weitere Datenelemente oder konkrete Werte enthalten. Konkrete Werte sind hierbei nur in den Blättern dieser Hierarchie erlaubt. Der LOM-Standard schlägt für einige dieser Datenelemente Werte vor. Dabei handelt es sich um eine empfohlene Liste angebrachter Werte oder es wird auf einen entsprechenden Standard verwiesen, wie beispielsweise auf den ISO-Standard 639:1988 für die Codierung von Sprachbezeichnern. (vgl. [Ha02])

Mit insgesamt 76 Feldern ist LOM, im Gegensatz zu Dublin Core, nicht sonderlich einfach zu verwenden. Dieses Problem versucht der Vorschlag von CanCore zu überwinden, indem nur eine bestimmte Teilmenge von 46 „aktiven“ LOM-Feldern verwendet wird (vgl. [FFR04]).

3 Semantic Web

Tim Berners-Lee formuliert die Idee des Semantic Web kurz und knapp in einem Satz:

> *„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.“* [BLHL01]

Der Begriff „Semantic Web“ wird vom *W3C* wie folgt erläutert:

> *„The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF).“*[1]

Daraus lassen sich drei Merkmale für das Semantic Web ableiten (vgl. [DJMZ04]):

- Das Semantic Web ist kein neues Web, sondern die Erweiterung des heutigen World Wide Web. Alle Anwender und Anwendungen, welche die Möglichkeiten des Semantic Web nicht nutzen möchten, können weiter arbeiten wie vorher.
- Die Erweiterung besteht darin, dass Informationen explizit um Aussagen bzgl. ihrer Bedeutung (Semantik) angereichert werden. Diese Erfassung von Semantik erfolgt durch den Menschen.
- Das Semantic Web hat das Ziel, den Menschen bei der Nutzung des Webs zu unterstützen.

Ein wichtiger Punkt des heutigen Web, die bei der Entwicklung des Semantic Web berücksichtigt werden muss, ist seine Universalität. Darunter ist zu verstehen, dass alles mit allem verlinkt werden kann und dies gilt im

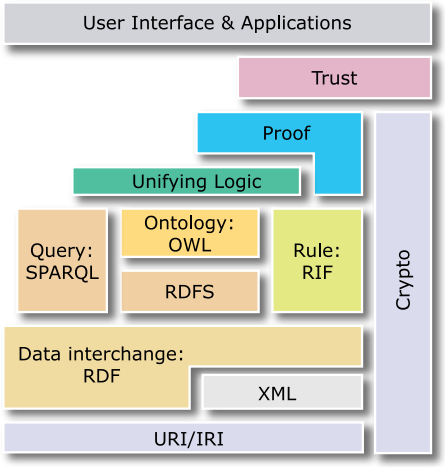
1. Semantic Web Activity Webseite – `http://www.w3.org/2001/sw/` (Abruf: 26. September 2008)

Semantic Web für jede Art von Ressource. Weiterhin sollen die Technologien des Semantic Web keine Unterschiede zwischen kommerziellen und akademischen Informationen, oder zwischen kulturellen, oder sprachlichen Grenzen oder zwischen Medien machen, alles soll gleichermaßen verarbeitet werden können (vgl. [BLHL01]).

Damit das Semantic Web funktioniert, braucht es strukturierte semantische Informationen. Um Semantik explizit in einer Form so beschreiben zu können, dass sie von beliebigen anderen Anwendungen ausgewertet werden kann, bedarf es allgemein akzeptierter herstellerunabhängiger Standards.

Abbildung 3.1 zeigt das Architekturmodell des Semantic Web, wie es auf der Webseite des W3C zu finden ist (vgl. [W3C08f]). Die einzelnen Schichten stellen jeweils eine Abstraktionsebene dar, die zusammen das Semantic Web bilden.

Abbildung 3.1: Schichtenmodell der Semantic Web Architektur



3.1 URI & IRI

> *„URIs are the glue that binds the Web together. IRIs extend and strengthen the glue, by allowing people to identify Web resources in their own language.“* (vgl. [W3C05c])

Mit Hilfe einer *Uniform Resource Identifier (URI)*[2] wird eine Ressource im Web eindeutig bezeichnet. Eine Ressource im Sinne des Semantic Web ist alles, was in eindeutiger Weise benannt werden kann. Dies können sowohl reale Objekte (Personen, Webseiten, Dateien etc.) als auch abstrakte Begriffe und Konzepte (z. B. Frieden oder Freiheit) sein.

Internationalized Resource Identifiers (IRI)[3] erweitern den URI-Standard im erlaubten Zeichensatz von ASCII auf das Unicode-Format. Benutzer müssen gemäß des IRI-Standards nichts ändern, da jeder URI auch bereits ein IRI ist. Es ist ihnen mit der Benutzung von IRIs aber jetzt möglich, Ressourcen in ihrer eigenen Sprache zu beschreiben. Die IRI-Spezifikation formuliert ebenso, wie

2. `ftp://ftp.rfc-editor.org/in-notes/rfc3986.txt` (Abruf: 26. September 2008)
3. `ftp://ftp.rfc-editor.org/in-notes/rfc3987.txt` (Abruf: 26. September 2008)

auf bestehenden Systemen ein IRI in einen URI zu konvertieren ist, die Äquivalenz zwischen IRIs, den Gebrauch von IRIs in verschiedenen Situationen, Überlegungen zur Sicherheit und einen informativen Leitfaden.

3.2 Extensible Markup Language

Auf eine umfangreiche Einführung zu *Extensible Markup Language (XML)* wird hier verzichtet, da XML nicht zentraler Gegenstand dieser Arbeit ist.

Bei XML handelt es sich um eine Sprache zur Definition von Auszeichnungssprachen bzw. Datenaustauschformaten, die gleichzeitig die grundsätzliche Syntax der Auszeichnungssprachen vorgibt. Als Beispiel für eine Auszeichnungssprache sei hier die *Extensible Hypertext Markup Language (XHTML)* für Webseiten im Web genannt.

XML wurde auf der Basis der *Standard Generalized Markup Language (SGML)* entwickelt, und bildet konzeptionell eine Untermenge der Sprachmöglichkeiten von SGML. Der XML-Standard wurde vom W3C entwickelt und liegt derzeit in der Version 1.1 vor (siehe auch [W3C06]).

XML-Dokumente enthalten Texte oder Daten, denen mit Hilfe von Auszeichnungen eine logische baumartige Struktur gegeben wird. Mit der Definition des Dokumenttyps (*Document Type Definition (DTD)*) wird die Syntax einer Auszeichnungssprache festgelegt. Eine Alternative gegenüber der Verwendung von DTDs zur Beschreibung von XML-Auszeichnungssprachen ist XML-Schema, mit der eine differenziertere Beschreibung der gewünschten Strukturen erfolgen kann.

Durch eine geschickte Wahl von Elementnamen und Attributen ermöglicht XML, die Daten und Dokumente mit Semantik bzw. Metadaten anzureichern. Dies allein reicht jedoch nicht aus, da z. B. eine spätere Anwendung die Bedeutung der Tags nicht kennen kann.

Es herrscht daher ein Bedarf an semantischer Interoperabilität. Um XML-Konstrukte in Beziehung zueinander setzen zu können, muss etwas über deren Bedeutung festgelegt werden können. Einen Mechanismus der dieses Problem lösen soll, wird durch das *Resource Description Framework (RDF)* bereitgestellt.

3.3 Resource Description Framework

Das Resource Description Framework[4] ist eine Spezifikation des W3C, welches als Standard für Web-Metadaten veröffentlicht wurde.

RDF bietet ein Grundgerüst für in XML definierte Auszeichnungssprachen zur Beschreibung von beliebigen Ressourcen. Es bildet die Grundlage für die Verarbeitung von Metadaten und unterstützt die Suche und automatische Verarbeitung von Inhalten. Mit Hilfe von RDF können Aussagen der Art „Berlin ist die Hauptstadt von Deutschland“ ausgedrückt werden.

Das RDF-Modell bietet eine syntaxunabhängige Darstellungsform für RDF-Ausdrücke und besteht aus drei Objekten:

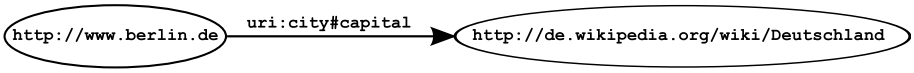
4. Unter `http://www.w3.org/RDF/` (Abruf: 26. September 2008) ist eine Zusammenfassung der Spezifikationen und vieler Ressourcen zu RDF zu finden, die von der RDF Core Working Group des W3C zusammengestellt wurden.

- **Ressourcen:** Alle Dinge, die durch RDF-Ausdrücke beschrieben werden, sind Ressourcen. Sie müssen nur durch einen eindeutigen URI gekennzeichnet sein.
- **Eigenschaften:** Eine Ressource wird durch ihre Eigenschaften definiert und beschrieben, wie z. B. Titel, Ersteller, Datum etc. Eigenschaften können durch Randbedingungen genauer spezifiziert werden.
- **Aussagen:** Eine Aussage besteht aus einer Ressource, einer Eigenschaft und ihrem Wert.

Die RDF-Spezifikation schlägt vor, dass Aussagen als Sätze der Form Subjekt, Prädikat und Objekt interpretiert werden, wobei die Ressource als Subjekt, die Eigenschaften als Prädikat und die Werte der Eigenschaft als Objekt zu verstehen sind (vgl. [W3C04a]).

Die oben formulierte Aussage über Berlin ließe sich somit als Tripel der Form *{Berlin \ ist die Hauptstadt von \ Deutschland}* darstellen. Hierbei ist jedoch laut Definition des RDF-Modells zu beachten, dass statt der Bezeichner URIs zu verwenden sind *{http://www.berlin.de \ uri:city#capital \ http://de.wikipedia.org/wiki/Deutschland}*.

Abbildung 3.2: Grafische Darstellung einer Aussage in RDF-Grammatik



In der Spezifikation zu RDF werden drei Darstellungsformen definiert, die unterschiedlichen Nutzungszwecken dienen:

- **Grafische Darstellung:** In der grafischen Darstellung werden Ressourcen als Ellipsen dargestellt und Beziehungen zwischen diesen Ressourcen als gerichtete Pfeile. Eine grafische Darstellung hat den Vorteil besonders bei komplexen Aussagen übersichtlicher zu sein. Abbildung 3.2 zeigt obiges Beispiel in grafischer Darstellung.
- **N-Tripel:** Ist eine Möglichkeit der Serialisierung der grafischen Darstellung.
- **RDF/XML:** Ist die Serialisierung in Form eines XML-Dokuments, beispielsweise zeigt Listing 3.1 obige Aussage in XML-Form.

Listing 3.1: Beispiel einer Aussage in XML-Syntax

```
1 <rdf:RDF
2     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3     xmlns:ci="uri:city#">
4     <rdf:Description rdf:about="http://www.berlin.de">
5         <ci:capital>http://de.wikipedia.org/wiki/Deutschland</
          →ci:capital>
6     </rdf:Description>
7 </rdf:RDF>
```

Es lassen sich mit RDF sehr komplexe und mächtige Dokumentenstrukturen bilden. Auf die Einzelheiten von RDF kann hier nicht weiter eingegangen werden. Weiterführende Informationen finden sich auf der RDF-Webseite[5] des W3C.

5. `http://www.w3.org/2001/sw/` (Abruf: 26. September 2008)

Mit dem RDF-Modell und der RDF/XML-Syntax können nur XML-Dokumente konstruiert werden, die sich an zusätzliche Regeln bezüglich der Struktur und der Benennung von Elementen und Attributen halten müssen. Diese können mit entsprechenden Werkzeugen überprüft und syntaktisch verarbeitet werden. Durch eine spezielle Wahl der Namen können zusätzliche Informationen abgelegt werden, aber zu einer automatischen Weiterverarbeitung reicht dies noch nicht. Hierfür wurde *RDF-Schema (RDFS)* entwickelt.

3.3.1 RDF-Schema

Mit RDF-Schema (vgl. [W3C04b]) kann ein eigenes strukturiertes RDF-Vokabular definiert werden, mit dem es möglich ist, Begriffe semantisch zueinander in Beziehung zu setzen. Es nutzt die RDF/XML-Syntax und das dazugehörige Modell, wobei es einige Namen für Elemente bzw. Attribute als Eigenschaften bzw. Typen vorgibt.

RDF-Schema bietet definierte Konzepte zur Beschreibung von Klassen, Ressourcen und Eigenschaften sowie deren Zusammenhänge. Es ermöglicht eine Typisierung von RDF-Ressourcen. Weiterhin können mit RDF-Schema hierarchische Strukturen (Taxonomien[6]) und ihre Beziehungen untereinander modelliert, Datentypen definiert und Restriktionen spezifiziert werden.

Mit der Hilfe von RDF-Schema können Vokabulare aufgebaut werden. Es bietet allerdings nur Basiskonzepte, die für viele Fälle nicht ausreichend sind. So treten auch nach der Einführung von RDF/RDF-Schema noch Schwierigkeiten auf, wenn verschiedene, voneinander unabhängige Wissensbereiche (Domänen) auf semantischer Ebene miteinander kommunizieren sollen (vgl. [DJMZ04]):

- Dieselbe Ressource kann über verschiedene URIs identifiziert sein (Synonyme).
- Ressourcen haben mitunter hierarchische Beziehungen oder Teilbeziehungen.
- Semantik muss aus vorhandener Information erschlossen werden (Inferenz).

Diese Schwierigkeiten können mit Hilfe von Ontologien beseitigt werden. Eine Ontologie wird im Kontext des Semantic Web als eine Sammlung von strukturierten zusammengehörigen Begriffen angesehen, die eine formale Beschreibung einer bestimmten Domäne bereitstellen. In Kapitel 4 wird das Thema Ontologien ausführlicher erläutert.

3.4 Anfragesprachen (Query)

Im Allgemeinen ist der Erfolg des Semantic Web von der Möglichkeit abhängig, in einer standardisierten Weise auf Daten zugreifen und Anfragen an Datenmengen stellen zu können. Wie Abbildung 3.1 zeigt, erweitert die Rule-Schicht das Schichtenmodell des Semantic Web, sodass Anfragen an eine Wissensbasis ermöglicht werden.

6. In Bezug auf Dokumente bzw. deren Inhalte wird der Begriff „Taxonomie“ für ein Klassifikationssystem, eine Systematik oder den Vorgang des Klassifizierens verwendet. Klassifizierungen können beispielsweise durch die Erfassung von Metadaten vorgenommen werden.

Mit einem breiteren Einsatz von Semantic Web Technologien – im speziellen RDF – wurden verschiedene Anfragesprachen entwickelt. In [MKA+02] und [BEHV04] wurden diese Anfragesprachen hinsichtlich ihrer Syntax und Semantik verglichen, wobei sich letztendlich keine davon durchsetzen konnte.

Im Februar 2004 gründete das W3C die RDF Data Access Working Group[7], um die Entwicklung eines einheitlichen Anfragestandards voranzutreiben. Diese Arbeitsgruppe arbeitete eine Reihe von Anwendungsfällen und Anforderungen an eine RDF-Anfragesprache heraus (vgl. [W3C05a]). Auf dieser Basis wurde die Anfragesprache SPARQL spezifiziert, auf welche in Abschnitt 5.5 ausführlicher eingegangen wird.

3.5 Wissensverarbeitung (Logic)

Mit Ontologien können Abhängigkeiten zwischen Ressourcen ausgedrückt werden, nicht aber logische Schlüsse oder Handlungsanweisungen (z. B. Hat eine Person aus dem Bekanntenkreis Geburtstag, dann sende ihr eine Grußkarte).

Um logische Schlussfolgerungen ziehen zu können, wird Logik benötigt: *„Logic is the study of the principles of reasoning“* (vgl. [AS06]). Dazu werden wichtige Gründe genannt, die für den Einsatz von Logik sprechen (vgl. [AS06] S. 144 ff. und [AH04] S. 151 ff.):

- Logik stellt eine High-Level Language zur Verfügung, die Wissen transparent darstellt und eine hohe Ausdruckskraft besitzt.
- Logik hat eine gut verstandene formale Semantik, welche die logischen Ausdrücke in eindeutiger Bedeutung verwendet.
- Es existieren Schlussfolgerungssysteme, welche durch Anwendung von Inferenzmechanismen unbekanntes Wissen aus vorhandenen Fakten vollständig, nachweisbar und automatisch ableiten. Für die Prädikatenlogik existiert ein derartiges Schlussfolgerungssystem.

Nach [AH04] S. 152 können RDF, OWL (Lite und DL) als Spezialisierungen der Prädikatenlogik betrachtet werden. Somit können Fakten, die durch Ontologien ausgedrückt werden, als Grundlage für logische Schlussfolgerungen verwendet werden. Ein Reasoner kann anschließend z. B. die Ontologie auf ihre Konsistenz hin überprüfen oder neue logische Schlüsse aus den gegebenen Fakten gewinnen.

3.5.1 Regeln (Rule)

Regeln sind ein wichtiger Bestandteil der Semantic Web Architektur und befinden sich in einer Schicht mit Ontologien (siehe Abbildung 3.1). In der Terminologie von Ontologiesprachen sind im Allgemeinen keine Regeln vorgesehen, da diese Variablen benötigen. Eine Ontologie kann mit Regeln ergänzt werden, um Aussagen zu treffen, die nicht in den Sprachmöglichkeiten der Ontologie definierbar sind. Generell spezifizieren Regeln *„. . . constraints, (implicit) construction of new data, data transformations, updates on data or, more general, event-driven actions“* (vgl. [BKPP07]).

Regeln können in drei Kategorien eingeteilt werden (vgl. [BKPP07]):

7. `http://www.w3.org/2001/sw/DataAccess/` (Abruf: 26. September 2008)

- **Deduction Rules** (Ableitungsregeln) definieren, wie neues Wissen mit logischem Schließen aus anderem Wissen hergeleitet wird. Sie beschreiben statische Abhängigkeiten zwischen Ressourcen und können benutzt werden, um zusätzliches implizites Wissen aus dem explizit vorhandenem Wissen einer Wissensbasis zu schließen. Die Regeln werden oft als Implikation in der Form *head* ← *body* spezifiziert, wobei *head* die zu schließenden Daten und *body* die zugrunde gelegten Daten definiert. Beide Regel-Teile benutzen im Allgemeinen gemeinsame Variablen, welche im *body* an Daten gebunden und im *head* für das Schließen von neuen Daten verwendet werden.

- **Normative Rules** (Normative Regeln) sind Regeln, welche Einschränkungen für die Daten einer Wissensbasis festlegen. Durch diese Regeln sollen Inkonsistenzen in der Wissensbasis vermieden werden.

- Über **Reactive Rules** (Reaktionsregeln) lässt sich dynamisches Verhalten beschreiben, d. h. es werden spezielle Aktionen ausgeführt, sobald bestimmte Ereignisse eintreten bzw. Bedingungen wahr werden. Der Unterschied zu Ableitungsregeln besteht darin, dass Reaktionsregeln Zustandsänderungen beschreiben.

Rule Interchange Format

Im Jahr 2005 hat das W3C die Rule Interchange Format Working Group (RIF WG) gegründet. Diese Arbeitsgruppe *„... is chartered to produce a core rule language plus extensions which together allow rules to be translated between rule languages and thus transferred between rule systems“* (vgl. [W3C05b]).

Die Arbeit der Arbeitsgruppe wurde in zwei Phasen, zu je zwei Jahren, aufgeteilt. Ziel der ersten Phase ist eine W3C Empfehlung eines sehr einfachen, doch nützlichen und erweiterbaren Formats einer *Core Rule Language*. Der Fokus bei der Entwicklung der Sprach-Spezifikation liegt dabei in der Wiederverwendung existierender Technologien und Standards wie XML, RDF, SPARQL und OWL. In der zweiten Phase sind dann W3C Empfehlungen für Erweiterungen zu spezifizieren, welche sich verstärkt an Anwendungsfällen orientieren sollen. (vgl. [W3C05b])

In die Entwicklung der Core Rule Language sollen auch die bisher beim W3C eingereichten Vorschläge für Rule Languages mit einbezogen werden, dazu gehören die *Semantic Web Rule Language (SWRL)*[8] , die *Semantic Web Rule Language First-Order Logic (SWRL FOL)*[9], die *Semantic Web Services Language (SWSL)*[10] und die *Web Rule Language (WRL)*[11].

3.6 Automatische Beweisführung (Proof)

Die Proof-Schicht zielt darauf ab, semantische Aktionen und Inferenzschritte der Maschine, für den menschlichen Benutzer transparent und nachvollziehbar zu machen. Dadurch soll es dem Benutzer ermöglicht werden, die logischen Schlussfolgerungen der Maschine zu verstehen.

8. `http://www.w3.org/Submission/2004/03/` (Abruf: 26. September 2008)
9. `http://www.w3.org/Submission/2005/01/` (Abruf: 26. September 2008)
10. `http://www.w3.org/Submission/SWSF-SWSL/` (Abruf: 26. September 2008)
11. `http://www.w3.org/Submission/2005/08/` (Abruf: 26. September 2008)

3.7 Vertrauen (Trust)

In einem Semantic Web, wie es bisher aufgezeigt wurde, steht es jedem frei alles zu behaupten und zu definieren, ob richtig oder falsch. Um Informationen hinsichtlich ihrer Gültigkeit zu überprüfen, sind geeignete Authentifizierungsmechanismen und Vertrauensprinzipien notwendig. Mit der Unterstützung von *Digitalen Signaturen* und der *Kryptographie* soll dem Semantic Web die Möglichkeit gegeben werden, die Echtheit von Informationen, Software-Agenten und Quellen im Semantic Web feststellen zu können und die Datensicherheit zu gewährleisten.

Mit dem Konzept der *Public Key Infrastructure* soll daraus ein weitreichendes *Web of Trust* entstehen, welches die Transitivität der Vertrauensbeziehungen zwischen den Informationen, Software-Agenten und Quellen im Semantic Web ausnutzt. Wenn z. B. *A* direkt *B* vertraut und *B* direkt *C*, dann sollte *A C* zumindestens in einem gewissen Grad vertrauen können.

4 Ontologien

Die Informatik steht in vielen Bereichen vor der Aufgabe, Erkanntes oder Erdachtes zu repräsentieren und Wissen zu kommunizieren. Menschen können sich gespeichertes Wissen zunutze machen, indem sie auf ihr persönlich gespeichertes Grund- und Kontextwissen des jeweiligen Wissensbereiches zurückgreifen. Kann auf diese Weise keine ausreichende Lösung gefunden werden, helfen umfangreiche Lehrbücher, Regelwerke, Lexika, Schlagwortregister usw., um einheitliche Konventionen über bestimmte Begriffe eines speziellen Wissensbereiches zu verwenden. Ein Computerprogramm kann im Allgemeinen nicht auf ein derartiges Hintergrundwissen zurückgreifen. Das Konzept der Ontologie soll nun diese Lücke schließen und eine differenziertere Beschreibung von Sachverhalten zulassen, um so den Mangel an semantischer Ausdruckfähigkeit des RDF/RDF-Schema Ansatzes (siehe Abschnitt 3.3) auszugleichen.

Der Begriff „Ontologie“ ist ein überlieferter Begriff aus der Philosophie und steht dort für die Lehre vom Sein – genauer: von den Möglichkeiten und Bedingungen des Seienden –, ist also eng verwandt mit der Erkenntnistheorie, die sich mit den Möglichkeiten und Grenzen menschlichen Wahrnehmens und Erkennens auseinandersetzt.

In weitgehender Anlehnung an diese Bedeutung bedient sich die Informatik des Begriffs Ontologie. Er beschreibt hier einen Wissensbereich (knowledge domain) mit Hilfe einer standardisierten Terminologie sowie Beziehungen und ggf. Ableitungsregeln zwischen den dort definierten Begriffen. Das gemeinsam genutzte Vokabular ist in der Regel in Form einer Taxonomie gegeben, die als Ausgangselemente Klassen, Relationen, Funktionen und Axiome enthält. Dabei ist eine Ontologie kein Abbild von Objekten aus der realen Welt, sondern vielmehr eine Spezifikation von Begriffen für diese Objekte. Wie Thomas R. Gruber in [Gru93] schreibt, kann eine Ontologie als *„. . . an explicit specification of a conceptualization“* definiert werden.

Ontologien sind also formale semantische Modelle einer Anwendungsdomäne, die dazu dienen, den Austausch und das Teilen von Wissen, insbesondere zwischen menschlichen und maschinellen Akteuren (z. B. Software-Agenten oder computerbasierte Systeme), zu erleichtern.

Wozu können nun Ontologien in der Informatik verwendet werden? In [GL02] unterscheiden Grunninger und Lee drei Anwendungsfelder: *Kommunikation*,

automatisches Schließen und *Repräsentation sowie Wiederverwendung von Wissen*. Zum Beispiel können Programme beim automatischen Schließen logische Schlüsse aufgrund der per Ontologie bekannten Ableitungsregeln ziehen.

4.1 Ontologiebeschreibungssprachen

Um eine Ontologie in einer formalen Struktur auszudrücken, wird eine Sprache benötigt, mit deren Hilfe die einzelnen Bestandteile einer Ontologie definiert werden können. Diese Sprachen werden als Ontologiebeschreibungssprachen oder einfach als Ontologiesprachen bezeichnet. Fensel formuliert Anforderungen, die eine solche Ontologiesprache erfüllen sollte:

- *„It must be highly intuitive to the human user. Given the current success of the frame-based and object-oriented modeling paradigm, it should have a framelike look and feel.*
- *It must have a well-defined formal semantics with established reasoning properties in terms of completeness, correctness, and efficiency.*
- *It must have a proper link with existing Web languages like XML and RDF, ensuring interoperability.“* (vgl. [FHLW03])

Basierend auf dem Bereich der KI wurde seit 1990 eine Vielzahl von Ontologiebeschreibungssprachen entwickelt. Die Grundidee besteht jeweils darin, aus einer Menge formalen Wissens automatisch neues Wissen abzuleiten. Ontologiebeschreibungssprachen lassen sich insbesondere durch ihre Modellierungskonstrukte und durch die Mächtigkeit ihrer zugehörigen Inferenzmaschinen unterscheiden.

Eine vollständige Übersicht und Beschreibung aller Ontologiesprachen würde den Rahmen dieser Arbeit überschreiten. Daher wird im folgenden Abschnitt nur auf die Sprache OWL eingegangen, die für die Ontologieentwicklung verwendet wurde. Eine ausführliche Zusammenstellung und Erläuterung verschiedener Ontologiesprachen wird von Gómez-Pérez in [GPFLC04] gegeben.

4.1.1 OWL

Die *Web Ontology Language (OWL*[1]*)* ist eine semantische Auszeichnungssprache zum Veröffentlichen und Austauschen von Ontologien. Die Sprache ist sehr stark an die im Dezember 2001 beim W3C eingereichte Ontologiesprache DAML+OIL[2] angelehnt. Die Syntax für die OWL basiert auf XML. Die OWL ist eine Spezifikation des W3C, die 2002 als Sprachentwurf veröffentlicht wurde und derzeit in der Publikation vom Februar 2004 vorliegt (siehe auch [W3C04c]).

Wie bereits erwähnt, werden, um die Ausdrucksmächtigkeit der OWL gegenüber RDF und RDF-Schema zu erweitern, weitere Beschreibungsmöglichkei-

1. Das Akronym für Web Ontology Language hätte eigentlich WOL, nicht OWL sein müssen. In Anspielung auf die literarische Figur der Eule aus Milnes *Pu der Bär*, die als einziges Tier im Wald ihren Namen schreiben kann – allerdings mit einem Buchstabendreher im englischen Original WOL statt OWL – wurde dieser Buchstabendreher umgekehrt für die OWL übernommen.
2. `http://www.w3.org/TR/daml+oil-reference` und `http://www.daml.org/` (Abruf: 26. September 2008)

ten eingeführt. Im folgenden Abschnitt werden diese Erweiterungen an einem kleinen Beispiel näher betrachtet.

Mit der OWL können *Klassen (Classes)*, *Relationen (Properties)* und *Instanzen (Individuals)* beschrieben werden. Klassen fassen gewissermaßen Begriffe einer bestimmten Domäne zu Gruppen zusammen und ordnen diese durch Klassifizierung. Mit der Klassifizierung entsteht eine Hierarchie in Form einer Baumstruktur, die als Taxonomie bezeichnet wird. Mit anderen Worten, Taxonomien sind Ontologien, die lediglich über *isSubclassOf*-Beziehungen verfügen. Relationen definieren spezielle Eigenschaften über einer Klasse. Instanzen sind individuelle Ausprägungen einer oder mehrerer Klassen.

Die OWL-Spezifikation (siehe [W3C04c]) des W3C besteht aus:

- **OWL Overview**[3] stellt eine allgemeine Einführung dar.
- **OWL Guide**[4] demonstriert die Benutzung der OWL anhand einiger Beispiele.
- **OWL Reference**[5] liefert eine systematische, kompakte und informative Beschreibung aller OWL-Elemente. Stellt aber keine Referenz dar.
- **OWL Semantics and Abstract Syntax**[6] ist die eigentliche Referenz zur OWL und ist somit das Hauptdokument der OWL-Spezifikation.
- **OWL Test Cases**[7] beinhaltet eine große Menge von Testfällen für die Sprache.
- **OWL Use Cases and Requirements**[8] spezifiziert Anwendungsfälle, Ziele und allgemeine Anforderungen an eine „web ontology language“.

Aufbau einer OWL-Datei

Eine detaillierte Erläuterung des vollen Sprachumfanges der OWL ist im Rahmen dieser Arbeit nicht möglich. Vielmehr werden im Folgenden der gundlegende Aufbau einer OWL-Datei und die wichtigsten Sprachelemente erläutert.

Im Kopfteil des XML-Dokumentes ist die für XML-Dateien notwendige XML-Erklärung <*?xml version="1.0"?*> untergebracht. Anschließend folgt in einem <*rdf:RDF*>-Tag die Deklaration der XML-Namensräume des verwendeten Vokabulars (siehe Listing 4.1 oben).

Der OWL-Header wird durch das *owl:Ontology*-Element definiert, hier besteht die Möglichkeit Angaben, über das OWL-Dokument selbst zu formulieren. Zum Beispiel können das Angaben über das Importieren anderer Ontologien *owl:imports*, die Benennung der Ontologie *rdfs:label*, die Version *owl:versionInfo* oder ein Kommentar *rdfs:comment* zur Ontologie sein. Listing 4.1 zeigt einen beispielhaften OWL-Header.

Die eigentliche Definition des Wissensbereiches erfolgt zwischen dem OWL-Header und dem schließenden <*/rdf:RDF*>-Tag. Klassen – genauer: Basisklassen – werden mit dem *owl:Class*-Tag definiert. Ausgehend von den Basisklassen können speziellere Klassen abgeleitet *rdfs:subClassOf* werden, wodurch

3. `http://www.w3.org/TR/owl-features/` (Abruf: 26. September 2008)
4. `http://www.w3.org/TR/owl-guide/` (Abruf: 26. September 2008)
5. `http://www.w3.org/TR/owl-ref/` (Abruf: 26. September 2008)
6. `http://www.w3.org/TR/owl-semantics/` (Abruf: 26. September 2008)
7. `http://www.w3.org/TR/owl-test/` (Abruf: 26. September 2008)
8. `http://www.w3.org/TR/webont-req/` (Abruf: 26. September 2008)

Listing 4.1: XML-Namespace Deklaration und OWL-Header Definition

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:ont="http://www.unimd.de/OntImport.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.unimd.de/Example.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.unimd.de/Example.owl">
    <owl:Ontology rdf:about="">
        <rdfs:label xml:lang="en">example-ontologie</rdfs:label>
        <owl:versionInfo rdf:datatype="http://www.w3.org/2001/
        →XMLSchema#string">01.04.2007</owl:versionInfo>
        <rdfs:comment xml:lang="en">comment</rdfs:comment>
        <owl:imports rdf:resource="http://www.unimd.de/OntImport
        →.owl"/>
    </owl:Ontology>
    ...
```

Listing 4.2: OWL-Class Definition

```
    ...
    <owl:Class rdf:ID="woman">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:hasValue>
                    <gender rdf:ID="female"/>
                </owl:hasValue>
                <owl:onProperty>
                    <owl:ObjectProperty rdf:ID="hasGender"/>
                </owl:onProperty>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf>
            <owl:Class rdf:ID="person"/>
        </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:ID="gender"/>
    ...
```

Listing 4.3: OWL-Property Definition

```
    ...
    <owl:ObjectProperty rdf:about="#hasGender">
        <rdfs:range rdf:resource="#gender"/>
        <rdfs:domain rdf:resource="#person"/>
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:ID="hasName">
        <rdfs:domain rdf:resource="#person"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/
        →XMLSchema#string"/>
    </owl:DatatypeProperty>
    ...
```

eine Taxonomie (Klassenstruktur) abgebildet wird. Listing 4.2 zeigt ein einfaches Beispiel.

Des Weiteren zeigt Listing 4.2 in den Zeilen 4–11 ein Beispiel dafür, wie Klassen exakter spezifiziert werden, indem die erlaubten Eigenschaften mit Hilfe der Klasse *owl:Restriction* eingeschränkt werden. Dafür muss innerhalb des Restriction-Elementes ein *owl:onProperty*-Element definiert werden, welches die Eigenschaft angibt, die eingeschränkt werden soll. Die Einschränkungen erfolgen mit mindestens einer der folgenden Eigenschaften: *owl:hasValue*, *owl:allValuesFrom*, *owl:someValuesFrom*, *owl:minCardinality*,*owl:maxCardinality* und *owl:cardinality*.

Die OWL bietet weitere Sprachelemente für Klassen an, so können z. B. Äquivalenzen mit *owl:equivalentClass* bzw. *owl:sameAs*, disjunkte Klassen mit *owl:disjointWith*, Aufzählungstypen mit *owl:oneOf*, Vereinigung mit *owl:unionOf*, Durchschnitt mit *owl:intersectionOf* und Negation mit *owl:complementOf* ausgedrückt werden.

Mit der Definition von Eigenschaften (Properties) lassen sich Aussagen über Klassen und den davon abgeleiteten Instanzen (Individuals) treffen (Listing 4.3). Es können zum Beipsiel datenwertige *owl:DatatypeProperty*, objektwertige *owl:ObjectProperty*, funktionale *owl:FunctionalProperty*, transitive *owl:TransitiveProperty*, symmetrische *owl:SymetricProperty* und inverse *owl:InverseFunctionalProperty* Properties definiert werden. Zu den beiden wichtigsten Typen von OWL-Properties gehören:

- **ObjectProperty:** Mit Hilfe der *owl:ObjectProperty* werden Relationen zwischen Instanzen von Klassen definiert (Listing 4.3 Zeilen 2–5).

- **DatatypeProperty:** Die *owl:DatatypeProperty* erlaubt die Zuordnung von XML-Datentypen zu einer Instanz (Listing 4.3 Zeilen 6–9).

Listing 4.4: OWL-Individual Definition

```
1     ...
2     <person rdf:ID="Jane_Doe">
3        <hasGender rdf:resource="#female"/>
4        <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#↘
         →string">Jane Doe</hasName>
5     </person>
6  </rdf:RDF>
```

Anhand der Klassendefinition ist es möglich, davon abgeleitete Individuen zu erstellen. Listing 4.4 zeigt ein Beispiel.

Obige Definitionen stellen nur die grundlegenden Konzepte eines OWL-Dokumentes dar. Die OWL stellt darüber hinaus eine Vielzahl von weiteren und komplexeren Sprachmitteln bereit, die in Abhängigkeit der verwendeten OWL-Sprachebene eingesetzt werden. Eine komplette Auflistung aller OWL-Sprachelemente und Hinweise zu ihrer korrekten Verwendung befindet sich in [W3C04c].

OWL-Sprachebenen

Die OWL besteht aus drei aufeinander aufbauenden Sprachversionen[9], deren Ausdrucksstärke von OWL Lite nach OWL Full zunimmt:

Abbildung 4.1: OWL Sprachebenen

- **OWL Full** ist für Anwender, die die maximale Ausdrucksfähigkeit benötigen und die syntaktische Freiheit von RDF, aber ohne Garantie der Berechenbarkeit, einsetzen möchten. Im Gegensatz zu den beiden OWL-Untersprachen sind bei OWL Full *owl:Class*, *owl:ObjectProperty* und *owl:Thing* äquivalent zu *rdfs:Class*, *rdfs:Property* und *rdfs:Resource*. Dadurch ist *owl:ObjectProperty* effektiv eine Oberklasse von *owl:DatatypeProperty*. Darüber hinaus können auch Klassen als Individuen betrachtet werden.

- **OWL DL**[10] beinhaltet den vollständigen OWL-Wortschatz und unterstützt jene Benutzer, die die volle Ausdrucksmöglichkeit unter Beibehaltung der vollständigen Berechenbarkeit einsetzen möchten. Diese Sprachversion wurde außerdem mit dem Ziel entworfen, den Bereich Description Logic zu unterstützten und die Entscheidbarkeit in einem Reasoner[11] zu gewährleisten. OWL DL ist eine Unterklasse von OWL Full, da eine Menge von Einschränkungen gilt. Diese Einschränkungen beziehen sich zum Teil auch auf die RDF-Schema-Konzepte, weshalb OWL DL keine Oberklasse von RDF-Schema darstellt. So sind z. B. in OWL DL alle OWL-Klassen disjunkt. Das gilt insbesondere für *owl:ObjectProperty* und *owl:DatatypeProperty*. Dadurch können die Unterklassen von *owl:ObjectProperty* nicht für *owl:DatatypeProperty* spezifiziert werden.

- **OWL Lite** ist eine Unterklasse der OWL DL und unterstützt in erster Linie Anwender, welche einfache Klassenhierarchien und einfache Beschränkungseigenschaften erstellen wollen. Folgende Sprachkonzepte sind nicht erlaubt: *owl:oneOf*, *owl:unionOf*, *owl:complementOf*, *owl:hasValue*, *owl:disjointWith*, *owl:DataRange* und *owl:Nothing*. Kardinalitäten werden unterstützt, der Wert darf aber lediglich 0 oder 1 betragen. Mit der OWL Lite ist eine Sprache gegeben, die effizient zu realisieren ist.

---

9. Eine detaillierte Erläuterung zur Syntax der einzelnen Sprachversionen findet sich in der OWL Referenz des W3C unter `http://www.w3.org/TR/owl-ref/`. (Abruf: 26. September 2008)
10. DL steht für Description Logics.
11. Reasoner (wie z. B. Pellet[12], Racer oder FaCT++) setzen auf Ontologien auf und leiten neues Wissen ab.
12. `http://pellet.owldl.com/` (Abruf: 26. September 2008)

OWL 2

Anfang 2008 wurde beim W3C der Entwurf für die *OWL 2 Web Ontology Language* veröffentlicht (siehe auch [W3C08a]). OWL 2 soll den derzeitigen OWL-Standard um zusätzliche Konstrukte und Eigenschaften erweitern, dazu gehören beispielsweise *„... additional property and qualified cardinality constructors, extended datatype support, simple metamodelling, and extended annotations“* (vgl. [W3C08a]).

4.2 Entwicklung von Ontologien

Staab bezeichnet den Prozess der *Ontologieentwicklung* als Wissensmetaprozess und bedient sich an dem Methoden- und Werzeuginventar des Software Engineering. Dabei untergliedert er diesen Prozess in folgende Schritte (vgl. [Sta02]):

1. **Machbarkeitsstudie:** Die Machbarkeitsstudie ist dem eigentlichen Ontologieentwicklungsprozess vorgelagert. In ihr werden im Wesentlichen die organisatorischen Rahmenbedingungen erfasst, wie z. B. die Frage nach den zu benutzenden Werkzeugen sowie nach den involvierten Personen. Am Ende der Machbarkeitsstudie muss „Go“ oder „No Go“ entschieden werden, d. h., ob eine ontologiebasierte Lösung der betrachteten Probleme angestrebt wird oder ob andere Maßnahmen zur Erreichung der Ziele geeigneter sind.

2. **Kickoff:** Die Kickoff-Phase dient dazu, eine erste Vorstellung von der Ontologie zu gewinnen: Welche Anforderungen müssen erfüllt werden, auf welchen Wissensquellen sollte aufbaut werden und wie könnte eine initiale semiformale Ontologiebeschreibung beschaffen sein? Am Ende der Kickoff-Phase muss entschieden werden, ob die betrachteten Anforderungen vollständig sind und die relevanten Wissensquellen ausreichend abgedeckt wurden.

3. **Verfeinerung:** In der Verfeinerungsphase wird die semiformale Ontologiebeschreibung auf ihre Konsistenz und Vollständigkeit geprüft und, wo nötig, erweitert. Schließlich erfolgt in der Verfeinerungsphase die Formalisierung der Ontologie. Abschließend muss überprüft werden, ob die Ontologie die Anforderungen, die in der Kickoff-Phase erhoben wurden, erfüllt. Ist dies nicht der Fall, muss die Ontologie berichtigt oder erweitert werden.

4. **Evaluierung:** Für die Evaluierung der Ontologie in der konkreten Anwendung wird ein lauffähiger Prototyp benötigt. Anhand des Prototyps können typische Anfragefälle analysiert werden. Falls sich aus der Evaluierung heraus ein Revisionsbedarf ergibt, ist es nötig, in die Verfeinerungsphase zurückzukehren, um dort die notwendigen Änderungen an der Ontologiestruktur vorzunehmen.

5. **Erweiterung und Anpassung:** Im Bereich der Erweiterung und Anpassung von Ontologien wird zwischen „kleineren“ und „größeren“ Änderungen unterschieden. Größere Änderungen, vor allem Umstrukturierungen, bedürfen einer neuerlichen Verfeinerungs- und Evaluierungsphase, während kleinere Änderungen das System nicht wesentlich beeinflussen.

Die eben beschriebene Vorgehensweise von Staab beschreibt lediglich eine Methode der Ontologieentwicklung. Für das Vorgehen bei der Entwicklung von Ontologien gibt es aber eine Reihe weiterer Möglichkeiten. Gómez-Pérez et al. geben in [GPFLC04] einen detaillierten Überblick über die verschiedenen Methoden der Ontologieentwicklung.

Weitere Ontologiebegriffe die mit der Entwicklung von Ontologien in Zusammenhang stehen sind z. B. *Ontologieentwurf* („ontology design (vgl. [HJ02])“) und *Ontologietechnik* („ontological engineering (vgl. [GL02])“). Auf diese Begriffe soll aber hier nicht weiter eingegangen werden.

4.3 Werkzeuge zur Ontologieentwicklung

Die Erstellung von Ontologien wird durch eine große Auswahl verschiedener Werkzeuge unterstützt. Um eine bessere Übersicht über diese Ontologiewerkzeuge zu erhalten, kann eine Klassifizierung anhand ihrer Funktionalität durchgeführt werden. Eine solche Übersicht von unterschiedlichen Werkzeugklassen wird von Gómez-Pérez gegeben (vgl. [GP02]):

- **Werkzeuge zur Ontologieentwicklung** sind darauf ausgerichtet, Ontologien von Grund auf neu zu entwickeln oder bereits existierende Ontologien wieder zu verwenden. Abgesehen von der Möglichkeit des Editierens und Browsing bieten diese Werkzeuge normalerweise auch die folgenden Funktionen: Ontologiedokumentation, Exportieren und Importieren von verschiedenen Sprachformaten, Visualisierung von Ontologien, Ontologiebibliotheken, Anbindung von Inferenzsystemen.

- **Merge- und Integrationswerkzeuge für Ontologien** wurden entwickelt, um verschiedene Ontologien, die für die gleiche Anwendungsdomäne erstellt wurden, in einer gemeinsamen Ontologie zu vereinen oder zu integrieren.

- **Evaluationswerkzeuge für Ontologien** kommen zum Einsatz, um sicherzustellen, dass Ontologien, sowie deren damit verbundenen Technologien, einem gewissen Qualitätsniveau genügen. Dies ist von großer Bedeutung, um Probleme bei der Integration von Ontologien und ontologiebasierten Technologien zu vermeiden.

- **Annotationswerkzeuge auf der Basis von Ontologien** sind konzipiert worden, um Nutzern semantische Annotationen von Webseiten auf Basis von Ontologien zu ermöglichen.

- **Ontologiespeicher- und Anfragewerkzeuge** wurden entwickelt, um den Gebrauch von und das Anfragen an Ontologien zu erleichtern.

- **Ontologielernwerkzeuge** ermöglichen eine teilautomatisierte Ableitung von Ontologien aus natürlichsprachlichen Texten.

Konkrete Beispiele einzelner Ontologiewerkzeuge zu den hier beschriebenen Klassen sind ebenfalls in [GP02] angegeben.

4.3.1 Ontologieframeworks für die Ontologieentwicklung

Im Rahmen dieser Arbeit wird als Entwicklungsumgebung für die Ontologieentwicklung *Protégé*[13] eingesetzt. Der Editor SWOOP[14] von der Universität Maryland ist ein weiteres Werkzeug für die Ontologieentwicklung, welches der Vollständigkeit halber erwähnt wird.

Bei Protégé handelt es sich um einen Editor für die Ontologieentwicklung, der als Open-Source-Project an der Universität Stanford entwickelt wurde. Die Anwendung wurde in Java geschrieben und ist damit plattformunabhängig. Neben Protégé selbst sind auch diverse Plugins für Protégé als Open-Source-Projekte im Internet verfügbar. Auf den Projektseiten von Protégé ist ein Überblick über alle zur Verfügung stehenden Plugins[15] gegeben.

Protégé bietet mit dem OWL-Plugin eine OWL-Unterstützung an, womit es möglich ist, OWL basierte Ontologien zu erstellen und zu editieren. Das OWL-Plugin benutzt als Grundlage die Jena-API[16], welche eine grundlegende Schnittstelle zur Arbeit mit RDF und Ontologien bereitstellt. Weitere Plugins, wie beispielsweise OntoViz[17] und Jambalaya[18] für eine grafische Visualisierung bei besonders komplexen Ontologien, können über eine Plugin-Schnittstelle in Protégé eingebunden werden.

13. `http://protege.stanford.edu/` (Abruf: 26. September 2008)
14. `http://www.mindswap.org/2004/SWOOP/` (Abruf: 26. September 2008)
15. `http://protege.stanford.edu/download/plugins.html` (Abruf: 26. September 2008)
16. Jena ist eine Open-Source Java API und wurde speziell für Semantic Web Applikationen entwickelt. `http://jena.sourceforge.net` (Abruf: 26. September 2008)
17. `http://protegewiki.stanford.edu/index.php/OntoViz` (Abruf: 26. September 2008)
18. `http://www.thechiselgroup.org/jambalaya` (Abruf: 26. September 2008)

5 Technologien der semantischen Suche

Der Begriff „semantischen Suche“ wird von Mangold wie folgt beschrieben:

> *„Unter dem Begriff der semantischen oder wissensbasierten Suche versteht man im Allgemeinen eine Suche nach Dokumenten, welche von Domänenwissen Gebrauch macht. Dabei ist weder die Art oder die Struktur des Domänenwissens festgelegt noch die Art der Einflussnahme auf den Suchprozess.“* [Man07b]

Analog zur vorliegenden Arbeit besteht das Ziel der semantischen Suche darin, die Suche nach Dokumenten durch die Verwendung von Kontextinformation, d. h. in diesem Fall Domänenwissen, zu verbessern.

Bevor die Vorteile einer semantischen Suche betrachtet werden können, müssen die Eigenschaften und Grenzen aktueller Suchdienste untersucht werden.

5.1 Suchdienste

Im Folgenden soll ein allgemeiner Überblick über die grundlegenden Konzepte der verschiedenen angebotenen Suchdienste gegeben werden. Eine detaillierte Erläuterung der gesamten Funktionsweise der verschiedenen Suchkonzepte[1] ist im Rahmen dieser Arbeit nicht möglich.

In [AS06] S. 193 ff. werden zwei Arten von webbasierter Suchtechnologie unterschieden, die den heutigen Suchmaschinen zugrunde liegen: Die *Human Direct Search* und die *Automated Search* Technologie. Lewandowski bezeichnet diese zwei Formen der Suchtechnologien als *manuell erstellte Verzeichnisse* und als *algorithmische Suchmaschinen* (vgl. [Lew05] S. 24).

5.1.1 Human Directed Search

Die Human Directed Search Technologie verwendet eine Datenbank von Schlüsselwörtern, Konzepten und Referenzen. In kontentbasierten Systemen wird oft eine einfache Suchmöglichkeit angeboten, welche Schlüsselwörter

1. Weitere Informationen kann hier z. B. das Buch von Lewandowski [Lew05] bieten.

und deren Relevanz benutzt. Die Bestimmung der Relevanz eines Dokumentes erfolgt über die Anfrage-Schlüsselwörter und in welcher Häufigkeit diese Schlüsselwörter für jede Seite in der Datenbank enthalten sind. Nach [AS06] kann diese einfache Suchmethode zu einer Ergebnismenge führen, die viele irrelevante und verfälschte Suchergebnisse enthält. Eine Verbesserung der eben genannten Suchmethode könnte darin bestehen, dass der Ort der Schlüsselwörter mit in die Relevanz einfließt. So hätte z. B. ein Schlüsselwort, welches im Titel eines Dokumentes vorkommt, eine höhere Relevanz, als ein Schlüsselwort, welches erst später im Text auftritt.

Eine andere Art der Human Directed Search Technologie basiert auf dem Konzept von Hierarchien und Webkatalogen. Mit diesem Konzept lässt sich Suchergebnissen einer ausgewählten Kategorie eine höhere Relevanz zuordnen. Hierarchien und Webkataloge werden redaktionell erstellt und gepflegt, d. h. Menschen entscheiden darüber, ob und wie bestimmte Dokumente den Kategorien zugeordnet werden. Dies hat ein hohes Qualitätsniveau der Suchergebnisse zur Folge, da irrelevante, unseriöse und kriminelle Inhalte bei der Erstellung herausgefiltert werden können. Jedoch ist dies auch ein Grund dafür, dass Hierarchien und Webkataloge nur einen Teil aller im WWW verfügbaren Dokumente berücksichtigen können. Weiterhin kann durch die redaktionelle Erstellung und Pflege des Kataloges die Aktualität erheblich verzögert werden. Ein Vertreter dieses Typs ist z. B. der Webkatalog dmoz[2].

5.1.2 Automated Search

Suchdienste, die auf der Automated Search Technologie basieren, durchsuchen das WWW automatisch und erfassen die gefundenen Dokumente in einer eigenen Datenbank. Wird eine Suchanfrage an die Suchmaschine gestellt, werden die Ergebnisse aus dieser Datenbank gewonnen und mittels eines Ranking-Algorithmus in einer bestimmten Reihenfolge ausgegeben.

Im Allgemeinen sind diese Suchdienste aus nachstehenden Komponenten aufgebaut (vgl. [Lew05] S. 27 ff.):

- Die Aufgabe eines **Webrobots** ist es, neue Dokumente aufzufinden, indem Hyperlinks innerhalb der bekannten Dokumente verfolgt werden. Dieser Prozess findet kontinuierlich statt, d. h. es werden ständig neue Dokumente erfasst und bereits Erfasste werden geupdated.

- Das **Parsing Module** zerlegt die gefundenen Dokumente in indexierbare Einheiten und erfasst deren Vorkommen innerhalb des Dokumentes.

- Das **Indexing Module** analysiert die im Parsing Module gewonnenen Informationen durch sog. Information Retrieval Verfahren. Das Ergebnis sind zwei Indizes, mit denen erstens sämtliche Dokumente, die ein bestimmtes Wort oder mehrere bestimmte Wörter enthalten und zweitens alle in einem Dokument vorkommenden Wörter ermittelt werden können.

- Das **Query Module** setzt Suchanfragen der Nutzer in eine weiterverarbeitbare Form um, dabei können beispielsweise Befehle und Operatoren aufgelöst werden. Dann wird diese Anfrage verarbeitet und die Informationen zu den gefundenen Dokumenten werden eventuell sortiert an

2. `www.dmoz.org` (Abruf: 26. September 2008)

den Nutzer zurückgegeben. Um eine hohe Relevanz der Suchtreffer zu erzielen, können Ranking-Algorithmen[3] verwendet werden.

Ein bekannter Vertreter dieser Gattung ist z. B. der Google[4]-Suchdienst mit seinem PageRank-Algorithmus[5].

5.1.3 Meta-Suchmaschinen

Meta-Suchmaschinen verbinden das Konzept der parallelen Suchanfrage an verschiedene Suchdienste, d. h. die Anfragen werden an verschiedene Suchdienste, ob Webkatalog oder Suchmaschine, gestellt und zu einem neuen Ergebnis aufbereitet. Mit z. B. MetaGer[6] wird die Suche über deutschsprachige Suchmaschinen ermöglicht.

5.2 Grenzen heutiger Suchtechnologien

Suchdienste helfen zwar dem Nutzer bei der Suche, sind jedoch mit ihren heutigen Technologien nicht in der Lage alle Dokumente des WWW zu erfassen bzw. dem Nutzer ausreichend relevante Suchergebnisse zu liefern. Trotz spezieller Suchstrategien ist es den heutigen Suchdienstebetreibern nicht möglich, die Gesamtheit aller im WWW verfügbaren Dokumente in ihrer Datenbank aufzunehmen. Tatsächlich ergeben Schätzungen, dass weniger als 50 % aller Dokumente von Suchdiensten erfasst werden (vgl. [Lew05] S. 41 ff.).

In [DSW06] S. 140 ff. werden einige Hauptursachen für die Grenzen heutiger Suchtechnologien identifiziert:

- **Query Construction:** Im Allgemeinen geben Nutzer, wenn sie eine Suchanfrage stellen, nur eine geringe Anzahl an Ausdrücken ein. Diese Ausdrücke beschreiben die gesuchten Informationen und basieren auf den Wörtern, welche die Nutzer in den zu findenden Dokumenten erwarten. Dies verursacht aber ein grundlegendes Problem, da nicht alle Dokumente die gleichen Wörter und Begrifflichkeiten verwenden. Folglich können nicht alle Dokumente durch eine einfache Schlüsselwortsuche gefunden werden.

  Weiterhin ist es möglich, dass Anfrageterme mehrere Bedeutungen besitzen können. Da Suchdienste den Sinn der Anfrage eines Nutzer nicht interpretieren können, führen diese Mehrdeutigkeiten in einer Anfrage zu irrelevanten Suchergebnissen. Obwohl dieses Problem der *Polysemie* zu einem gewissen Grad durch die Eingabe von zusätzlichen Schlüsselwörtern überwunden werden könnte, sind Nutzer nicht bereit dies zu tun. Analysen zufolge enthalten Suchanfragen durchschnittlich zwei Schlüsselwörter (vgl. [DSW06] S. 140).

- **Lack of Semantics:** Wird eine Anfrage, in der Wörter vorkommen, die nicht im Suchindex enthalten sind, an einen Suchdienst gestellt, obwohl gleichbedeutende Wörter im Suchindex vorhanden sind, kann diese Anfrage üblicherweise nicht beantwortet werden. Eine thesaurusbasierte

3. Für weitere Informationen über Ranking und Ranking-Algorithmen siehe [Lew05] S. 89 ff.
4. `www.google.com` (Abruf: 26. September 2008)
5. Ein Erklärung zum Google PageRank-Algorithmus ist z. B. in [AS06] S. 195 zu finden.
6. `http://meta.rrzn.uni-hannover.de/` (Abruf: 26. September 2008)

Expansion der Anfrage könnte dieses Problem der *Synonymie* auflösen, würde aber zu einem Vielfachen an Suchergebnissen führen.

Da Polysemie und Synonymie von Suchdiensten nicht gehandhabt werden können, kann daraus geschlossen werden, dass konventionelle Suchdienste auch keine semantischen Beziehungen zwischen Konzepten herstellen können (vgl. [DSW06] S. 140).

- **Lack of Context:** Suchdienste könnten Nutzeraspekte in den Anfragen mit berücksichtigen, um Mehrdeutigkeiten in einer Suchanfrage zu verringern. Solche Aspekte könnten Informationen wie z. B. Abteilung, Erfahrung oder Interessen einer Person umfassen.

- **Presentation of Results:** Die Suchergebnisse einer Anfrage werden dem Nutzer normalerweise als eine geordnete Liste dargestellt. Der Nutzer muss nun anhand der angezeigten Ergebnisfragmente entscheiden, ob er ein gefundenes Dokument ansehen möchte oder nicht. Nach einer Studie betrachten die meisten Nutzer nicht mehr als die Suchergebnisse auf der ersten Ergebnisseite bei einer Anzeige von 10 Ergebnissen pro Seite. Nur 17 % der Nutzer sahen sich die Ergebnisse an, die über die erste Ergebnisseite hinausgingen. (vgl. [DSW06] S. 141)

- **Managing Heterogeneity:** Unternehmensinterne Suchdienste werden benötigt, um eine große Anzahl an Informationen aus verschiedenen Quellen wie z. B. Webseiten, Content Management Systemen, Dokumenten Management Systemen und Datenbanken durchsuchbar zu machen. Aber nicht nur die Anbindung der verschiedenen Quellen stellt eine besondere Herausforderung dar, sondern auch die einheitliche Ansicht der Suchergebnisse aus den verschiedenen Quellen.

Als ein weiterer Grund für diese Beschränkungen kann der Aufbau heutiger webbasierter Informationsangebote identifiziert werden. Diese Informationsangebote basieren meist auf dem Client-Server-Prinzip, indem Inhalte in Form von Hypertext auf dem Server bereitgestellt werden und der Nutzer (Client) auf diese Inhalte zugreift. Dies führte dazu, dass *„… the Web has developed as a medium for humans without a focus on data that could be processed automatically“* (vgl. [AS06] S. 67). Durch diesen visuell basierten Ansatz ist es Suchdiensten nicht möglich, komplexe semantische Beziehungen zwischen webbasierten Inhalten herzustellen.

5.3 Die Rolle der Semantik

Nach [DSW06] S. 142 kann unter der Ausnutzung semantischer Technologien und eines erhöhten Informationszuganges, auf der Basis von maschinell verarbeitbaren Metadaten, für viele der in Abschnitt 5.2 genannten Beschränkungen eine Lösung gefunden werden. Mit Hilfe des Semantic Web – im speziellen Ontologien – kann der Austausch und die Wiederverwendung von Informationen erleichtert werden, da Ontologien eine Möglichkeit liefern, eine Anwendungsdomäne konsistent und formal zu beschreiben. Somit könnten Metadaten in direktem Bezug zu der in der Ontologie beschriebenen Domäne erstellt werden. Dies würde zu einer einheitlichen Informationsquelle führen, der eine maschinell verarbeitbare Ontologie zugrunde liegt. Mit dem Einsatz von Metadaten auf der Basis von Ontologien können Anfragen der Nutzer präziser,

d. h. durch ontologische Konzepte, ausgedrückt werden. Somit wird es möglich, Anfragen über die Metadaten zu spezifizieren und diese Anfragen ggf. mit einer textbasierten Suche zu kombinieren.

Weiterhin kann die Verwendung semantischer Technologien eine grundlegende Änderung des Informationszugriffs ermöglichen. In [DSW06] S. 143 wird vorgeschlagen, dass *„... the future of search engines lies in supporting more of the information management process, as opposed to seeking incremental and modest improvements to relevance ranking of documents."* Dieser informationszentrierte Ansatz würde es ermöglichen, dass Suchdienste die relevanten Informationen in Dokumenten analysieren, anstatt nur eine Ergebnisliste zu präsentieren, um die eigentliche Informationssuche dem Nutzer zu überlassen. Auf diese Weise können dem Nutzer die Informationen geliefert werden, die er benötigt. Dies könnten beispielsweise einfache numerische Antworten, Textabschnitte, ganze Texte oder auch mehrere Texte sein.

5.4 Anfragetechniken

Anfragetechniken beschreiben den Anfrageprozess des Nutzers an das System. Diese auch als Information Retrieval bezeichneten Prozesse lassen sich in Anfragen an oder die Navigation über eine vorgegebene Struktur differenzieren (vgl. [Joh06]). In [DSW06] werden diese Techniken als *Semantic Search* (semantische Suche) und *Semantic Browsing* (semantische Navigation) bezeichnet.

Im Bereich der semantischen Anfragetechniken wurde in den vergangenen Jahren eine Vielzahl von Ansätzen und Ideen entwickelt, die sich sowohl im Anwendungsgebiet als auch in ihrer Realisierung unterscheiden. Eine Übersicht und Klassifizierung der verschiedenen Ansätze wurde von Mangold und Hoang vorgenommen (vgl. [Man07a], [HT06]). Die dort beschriebenen Merkmale und Kriterien werden in Abschnitt 6.3.1 verwendet, um die grundlegenden Eigenschaften eines ontologiebasierten Anfragesystems zu spezifizieren.

5.4.1 Semantic Search

Unter Semantic Search kann ein Anfrageprozess verstanden werden, der eine assoziative Suche von ähnlichen Begriffen unterstützt und andere Beziehungen oder Regeln aus der Ontologie bei der Suche mit einbeziehen kann. Semantic Search hebt sich gegenüber dem konventionellen Anfrageprozess durch vielfältigere Möglichkeiten bei der Auswahl von Suchparametern ab.

Im Fokus dieser Arbeit stehen Anfragen, bei denen die Eingabe eines Suchausdrucks in einer formalen Sprache erfolgt – im speziellen SPARQL (siehe Abschnitt 5.5). Nicht erläutert werden die Ansätze zur semantischen Suche auf XML-Dokumenten oder auf Peer-To-Peer-Systemen. Weiterhin werden auch die Ansätze nicht weiter betrachtet, bei denen die Anfragen auf natürlichsprachliche Weise ausgedrückt werden.

5.4.2 Semantic Browsing

Semantic Browsing ist eine explorative Form der Anfrage, die eine Navigation über die Baum oder Netzstruktur einer semantischen Anwendung ermöglicht. Dabei unterstützt Semantic Browsing die Navigation in einem semantischen Netz, indem zu jeder wählbaren Kategorie die entsprechend zugeordneten Do-

kumente angezeigt werden. Eine Ausprägung des Semantic Browsing ist das Facetet Browsing, welches in Abschnitt 5.6 näher betrachtet wird.

Das Navigieren auf nativen RDF-Daten kann als ein weiterer Ansatz des Semantic Browsing betrachtet werden. Dabei erfolgt die Navigation direkt über URIs. Dieser Ansatz wird überwiegend auf Peer-To-Peer-Systemen verwendet. Ein Vertreter dieses Ansatzes ist z. B. das Browser-Plugin Tabulator[7], welches ursprünglich von Tim Berners-Lee entwickelt wurde.

5.5 SPARQL

„Trying to use the Semantic Web without SPARQL is like trying to use a relational database without SQL.“ erklärte Tim Berners-Lee in [W3C08e].

SPARQL ist eine RDF-Anfragesprache und wurde im Januar 2008 vom W3C als Standard veröffentlicht. SPARQL[8] steht dabei für *SPARQL Protocol and RDF Query Language*. Die Spezifikation besteht aus

- der Anfragesprache SPARQL (**SPARQL Query Language for RDF**) [W3C08c],
- einem Remote-Protokoll (**SPARQL Protocol for RDF**) [W3C08b], um SPARQL-Anfragen und Ergebnisse über Web-Dienste ausführen zu können und
- einem XML-Format (**SPARQL Query Results XML Format**) [W3C08d] zur Beschreibung von Anfrage-Ergebnissen.

Auf der Webseite der RDF Data Access Working Group des W3C befindet sich eine Zusammenstellung aller dem W3C bekannten SPARQL-Implementationen.[9]

SPARQL ist eine deklarative Anfragesprache, welche *„. . . contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs“* (vgl. [W3C08c]).

Die Ausführung der Anfragen basieren auf dem Konzept des *Pattern Matching*. Hierfür werden in einer Anfrage *Graph-Pattern* beschrieben, welche Variablen enthalten können. Bei der Anfrageausführung werden zu diesen Graph-Pattern übereinstimmende Teilgraphen aus den abgefragten RDF-Graphen bestimmt. Die Variablen sind an die entsprechenden URIs dieser Teilgraphen gebunden und bilden die Lösung.

Listing 5.1 zeigt eine einfache SPARQL-Anfrage die nach allen Dokumenten sucht, die beim W3C erschienen sind und für die ein Titel bekannt ist. Die Daten-Grundlage der Anfrage bildet der in Listing 5.2 abgebildete Datensatz in

7. `http://dig.csail.mit.edu/2007/tab/` (Abruf: 26. September 2008)
8. ausgesprochen „sparkle“
9. `http://www.w3.org/2001/sw/DataAccess/impl-report-ql` (Abruf: 26. September 2008)

Turtle-Syntax[10]. Das Ergebnis der Anfrage besteht aus einer zweielementigen Menge und ist in Listing 5.3 dargestellt.

Listing 5.1: einfache SPARQL-Anfrage

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE  { ?document dc:creator <http://www.w3.org/> ;
                   dc:title   ?title . }
```

Listing 5.2: RDF Daten

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .

_:SPARQL-QL dc:creator <http://www.w3.org/> ;
            dc:title   "SPARQL Query Language for RDF" .
_:Turtle-TL dc:creator <http://www.w3.org/> ;
            dc:title   "Turtle - Terse RDF Triple Language" .
```

Listing 5.3: Ergebnis der einfachen SPARQL-Anfrage

```
 ?title
=======================================
 "Turtle - Terse RDF Triple Language"
 "SPARQL Query Language for RDF"
```

Eine SPARQL-Anfrage setzt sich aus drei wesentlichen Bestandteilen zusammen, die durch die Schlüsselwörter *PREFIX*, *SELECT* und *WHERE* gekennzeichnet sind (vgl. [HKRS08]).

Mit dem Schlüsselwort PREFIX können eine Basis-URI und eine Menge von Namespaces festgelegt werden.

Die Anfrageform der Anfrage wird mit dem Schlüsselwort SELECT bestimmt – der SPARQL-Standard definiert insgesamt vier Anfrageformen (vgl. [W3C08c]):

- **SELECT** gibt die angegebenen Variablen mit den im Pattern Matching gefundenen Werten zurück. Mit der Angabe von *SELECT* * können alle Variablen ausgewählt werden. Die Rückgabe des Ergebnisses kann serialisiert als RDF-Graph oder in einem XML-Format erfolgen. Eine detaillierte Beschreibung des XML-Formates befindet sich in [W3C08d].
- **CONSTRUCT** gibt einen RDF-Graphen zurück, welcher durch ein Graph-Template beschrieben wird. Dieser RDF-Graph wird durch Pattern Matching und durch die im Graph-Template beschriebenen Ersetzungsregeln konstruiert.
- **ASK** prüft, ob das angegebene Pattern in dem RDF-Graphen enthalten ist, und gibt einen booleschen Wert zurück. Es werden keine weiteren Informationen über die gefundenen Ergebnisse zurückgegeben.
- **DESCRIBE** erzeugt einen RDF-Graphen, welcher die Ergebnismenge beschreibt. Die Ergebnismenge kann dabei durch Variablen und durch Graph Pattern beschrieben werden.

10. Turtle (Terse RDF Triple Language)[11]ist eine Form der Serialisierung für RDF, hat aber keinen offiziellen Status bei einer Standardisierungs-Organisation. Die Turtle-Syntax ist wegen ihrer einfacheren Darstellungsweise bei Semantic Web Entwicklern populär.
11. `http://www.w3.org/TeamSubmission/turtle/` (Abruf: 26. September 2008)

Die eigentliche Anfrage wird durch das Schlüsselwort WHERE eingeleitet. Ihm folgt, eingegrenzt durch die Schlüsselwörter *{* und *}*, ein Graph-Pattern, wobei das Graph-Pattern auf verschiedene Weise gebildet werden kann (vgl. [W3C08c]).

5.5.1 Einfache Graph-Pattern

Das *einfache Graph-Pattern* (Basic Graph Pattern) besteht aus einer Menge von *Tripel-Pattern* (Tripel Pattern) (vgl. [W3C08c]). Tripel-Pattern sind Tripel, in denen Variablen an Stelle der RDF-Ausdrücke für Subjekt, Prädikat und Objekt stehen können. Das Pattern Matching eines einfachen Graph-Pattern in Bezug auf einen RDF-Graphen erfolgt dadurch, dass alle Variablen der Tripel durch RDF-Terminale ersetzt werden und dann das Pattern auf enthalten sein im RDF-Graphen geprüft wird.

5.5.2 Gruppierende Graph-Pattern

Ein *gruppierendes Graph-Pattern* (Group Graph Pattern) besteht aus einer Menge von einfachen Graph-Pattern und wird durch die Schlüsselwörter *{* und *}* begrenzt. Das Pattern Matching eines gruppierenden Graph-Pattern ist erfolgreich, wenn alle enthaltenen einfachen Graph-Pattern passend sind.

5.5.3 Optionale Graph-Pattern

Optionale Graph-Pattern (Optional Graph Pattern) ermöglichen es, zusätzliches und unvollständiges Wissen in die Anfrage mit einzubeziehen. Der optionale Teil eines Graph-Pattern wird mit dem Schlüsselwort *OPTIONAL* eingeleitet und kann beliebige Graph-Pattern enthalten. Das Pattern Matching des optionalen Teils erfolgt zusätzlich, d. h. *„. . . if the optional part does not match, it creates no bindings but does not eliminate the solution“* (vgl. [W3C08c]).

5.5.4 Alternative Graph-Pattern

Alternative Graph-Pattern (Alternative Graph Pattern) dienen der Beschreibung von Lösungsalternativen, hierfür können Graph-Pattern durch das Schlüsselwort *UNION* miteinander verkettet werden. Passen mehrere der alternativ angegebenen Pattern überein, werden mehrere Ergebnisse in der Ergebismenge erzeugt.

5.5.5 Benannte Graph-Pattern

Eine SPARQL-Anfrage wird üblicherweise auf einer RDF-Datenmenge ausgeführt. Der SPARQL-Standard sieht vor, dass diese RDF-Datenmenge in mehrere Graphen unterteilt werden kann. Diese einzelnen Graphen werden durch URIs bezeichnet und sind deshalb als *benannte Graphen* (Named Graphs) bekannt. Mit dem Schlüsselwort *GRAPH* kann dann explizit angegeben werden, welche Graphen für das Pattern Matching verwendet werden sollen.

5.5.6 Filter

SPARQL erlaubt es, die möglichen Ergebnisse einer Anfrage durch boolesche Ausdrücke in einer *FILTER*-Funktion einzuschränken. Diese booleschen Ausdrücke formulieren Bedingungen für die, in den Graph Pattern gebundenen Variablen. Filter können Vergleichsoperationen, boolesche Operationen, arithmetische Operationen und spezielle Operatoren (z. B. für den Zugriff auf

RDF-spezifische Informationen oder für reguläre Ausdrücke) enthalten. Eine komplette Auflistung der Filteroperationen befindet sich in [W3C08c].

In Listing 5.4 ist eine SPARQL-Anfrage abgebildet, welche die obige einfache Anfrage um einen Filter erweitert. Die Anfrage ist ein Beispiel für eine Suche nach allen Dokumenten, die beim W3C erschienen sind, für die ein Titel bekannt ist und dieser Titel das Wort „SPARQL“ enthält. Wenn die Anfrage auf den RDF-Datensatz aus Listing 5.2 angewendet wird, dann ergibt sich das in Listing 5.5 dargestellte Ergebnis.

Listing 5.4: SPARQL-Anfrage mit Filter

```
1 PREFIX dc: <http://purl.org/dc/elements/1.1/>
2 SELECT ?title
3 WHERE { ?document dc:creator <http://www.w3.org/> ;
4                   dc:title   ?title .
5                   FILTER     ( REGEX(?title, "SPARQL") ) }
```

Listing 5.5: Ergebnis der SPARQL-Anfrage mit Filter

```
1 ?title
2 ==================================
3 "SPARQL Query Language for RDF"
```

5.5.7 Modifikatoren

Weiterhin bietet der SPARQL-Standard *Modifikatoren* (Solution Sequences and Modifiers) an, um die genaue Form und Größe der Ergebnismenge zu kontrollieren. Hierbei handelt es sich um folgende Optionen, die einzeln oder auch kombiniert angegeben werden können:

- Mit dem Schlüsselwort **ORDER BY** und einer Liste von Ordnungskriterien, kann die Ergebnismenge einer SPARQL-Anfrage sortiert werden. Diese Ordnungskriterien können aus Variablen und Funktionsaufrufen bestehen, welche für eine Sortierung der Ergebnisse sorgen. Die Sortierung der Ergebnisse erfolgt per default aufsteigend, aber mit den Schlüsselwörtern **ASC** und **DESC** kann eine Sortierordnung explizit angegeben werden.

- Mit dem Schlüsselwort **OFFSET** und einer darauf folgenden Zahl wird der Index des ersten relevanten Ergebnisses innerhalb der Ergebnismenge festgelegt. Alle Ergebnisse vor dem so spezifizierten Index werden ignoriert.

- Dem Schlüsselwort **LIMIT** folgt ebenfalls eine Zahl. Diese Zahl legt die maximale Anzahl der zu liefernden Ergebnisse fest.

- Mit dem Schlüsselwort **DISTINCT**, welches nur direkt nach dem Schlüsselwort SELECT auftreten darf, kann die Disjunktheit aller Ergebnisse der Ergebnismenge gefordert werden. Genauer gesagt wird jedes Ergebnis, welches dieselben Variablen an dieselben RDF-Terme bindet, aus der Ergebnismenge eliminiert.

5.6 Faceted Browsing

Die Ursprünge des Gebrauchs von Facetten in der Suche nach Informationen reichen bis in das 18. Jahrhundert zurück. Das Wort „Facette“ wurde erstmals im Jahr 1933 von Shiyali Ramamrita Ranganathan (1892–1972) in der Veröffentlichung zur sog. Colon-Klassifikation verwendet.

Mit Hilfe des *Faceted Browsing* soll aus einem großen Angebot von Metadatenbeständen, mit der Unterstützung von *Facetten*, schrittweise das zu suchende Ziel eingeschränkt werden. Unter Facetten sind hierbei einzelne, wesentliche Eigenschaften der gesuchten Ergebnisse zu verstehen.

Ein Beispiel im Bereich des Weins (vgl. [Den03]): Jeder Wein hat eine bestimmte Farbe. Er wird aus einer bestimmten Art Trauben hergestellt. Der Wein kommt aus einer bestimmten Gegend. Das Jahr seiner Weinlese ist bekannt. Er kommt in einem Behältnis mit einem bestimmten Volumen. Er hat einen bestimmten Preis. Eine Liste von Weinen mit all ihren Eigenschaften könnte erstellt werden, aber diese wäre lang und unhandlich. Mit Facetten können nun eine handvoll Kategorien aufgestellt werden, um die Weine vollständig zu beschreiben: Farbe, Traubenart, Herkunftsort, Jahr, Volumen und Preis. Jede dieser Kategorien ist mit den entsprechenden Werten bestückt. Dann wird jede Flasche Wein durch eine entsprechende Auswahl der richtigen Werte aus jeder Kategorie klassifiziert. Diese Art der Klassifizierung wird *Faceted Classification* genannt und sie ergibt eine Gruppe eindeutiger und sich gemeinsam ergänzender Kategorien, um alle fraglichen Objekte vollständig zu beschreiben. Mit Hilfe dieser Facetten können Benutzer durch Suchen bzw. Browsen das finden, was sie benötigen.

Welche Eigenschaften macht ein Faceted Browsing aus?

- Die Auswahl der Facetten erfolgt dynamisch, d. h. es können weitere Facetten ausgewählt und bereits ausgewählte Facetten wieder entfernt werden.
- Zu jeder angebotenen Facette werden die enthaltenen Treffer angezeigt.
- Facetten mit keinem Treffer (zero results) werden nicht angezeigt.
- Bereits ausgewählte Facetten werden für ein mögliches Entfernen angezeigt.
- Nach jeder Auswahl einer Facette, ob hinzugefügt oder entfernt, erfolgt eine Neuberechnung der Treffer und der angezeigten Facetten.

Nach [HEE+02] haben Nichtexperten erhebliche Schwierigkeiten bei der Formulierung von *boolean queries* an einen Metadatenbestand. Das Konzept des Faceted Browsing erlaubt Benutzern *„... to easily compose queries consisting of ANDs of ORs: selecting a category term is effectively an OR of all of its subcategories, and selecting more than one facet produces an AND across facets“* (vgl. [HEE+02]).

An der University of California in Berkley wurde im Jahr 2002 eine Nutzerstudie in Bezug auf das Nutzerverhalten mit Faceted Browsing in umfangreichen Metadatenbeständen publiziert. Ergebnis der Studie war u. a. das Suchinterface *Flamenco*[12] mit dem primären Designziel *„... to allow users to move*

12. `http://flamenco.berkeley.edu` (Abruf: 26. September 2008)

through large information spaces in a flexible manner without feeling lost“ (vgl. [HEE+02]).

Ein Hauptaugenmerk bei dem Entwurf und der Entwicklung des Interfaces Flamenco lag dabei in der expliziten Verwendung von *Hierarchical Faceted Metadata*. Hierarchical Faceted Metadata erweitern Faceted Metadata um ein hierarchisches Konzept. So könnte z. B. in dem oben ganannten Wein-Beispiel die Facette des Herkunftsorts nach Kontinent → Land → Ort angeboten werden, d. h. die Facette würde sich nach jeder Auswahl weiter verfeinern. Weiterhin wurde eine Freitext-Suche in das Suchinterface integriert. Diese erlaubt es, in Kombination mit der Facett-Auswahl, Suchterme in die Auswahl mit einzubeziehen.

Teil II

PlOSS^X

6 Konzept

6.1 Vorgehensweise

Im Folgenden werden die für die Lösung der Aufgabenstellung eingesetzten Methoden Schritt für Schritt erarbeitet und teils an Beispielen und Abbildungen beschrieben. Die aufgetretenen Probleme werden nicht verschwiegen und bilden an einigen Stellen die Grundlage für die Begründung eines eingeschlagenen Ansatzes.

Nachdem im nächsten Abschnitt das Gesamtkonzept beschrieben wird, beginnt die Beschreibung des Anforderungskontextes für ein ontologiebasiertes Anfragesystem. Dies beinhaltet sowohl die grundlegenden Anforderungen einer semantischen Suchtechnologie als auch die speziellen Anforderungen in Bezug auf die Aufgabenstellung bzw. auf das Gesamtkonzept. Im Anschluss daran werden verwandte Ansätze, welche Metainformationen auf verschiedene Art und Weise mit Plone verbinden, betrachtet.

In Kapitel 7 folgt eine Beschreibung der zu durchlaufenden Arbeitsschritte bei der Erstellung einer Ontologie. Dabei wird auch die Wiederverwendung bereits vorhandener Ontologien näher betrachtet. Weiterhin wird gezeigt, wie Objekte eines Content Management Systems auf Instanzen einer Ontologie abgebildet werden können.

In Kapitel 8 wird dann gezeigt, wie eine semantische Suche auf Basis der in Kapitel 7 erstellten Ontologie erfolgen kann.

6.2 Gesamtkonzept

Seit dem Wintersemester 2003/2004 werden vom Institut für Wissens- und Sprachverarbeitung an der Otto-von-Guericke-Universität Magdeburg multimediale Lehrangebote in einer E-Learning-Umgebung in der universitären Lehre eingesetzt. Die Bereitstellung der Lehr- und Lernmaterialien erfolgt auf einer zentralen, offenen und erweiterbaren Plattform, dem Content Management System Plone. Plone kann durch Produkte genannte Module erweitert werden, die z. B. Inhaltsobjekte oder andere Funktionalitäten bereitstellen können. Die E-Learning-Umgebung basiert auf den *eduComponents*, einer

Sammlung von Plonemodulen, die Plone um E-Learning-Funktionen erweitern (vgl. [APR06]). *„Die eduComponents umfassen Module für die Verwaltung von Lehrveranstaltungen (ECLecture), für Multiple-Choice-Tests (ECQuiz) und für die Einreichung von manuell bewerteten Übungsaufgaben (ECAssignmentBox)“* (vgl. [APR07]). Während des Einsatzes *„... entstand eine umfangreiche Sammlung von mehr als hundert Übungsaufgaben zu den Lehrveranstaltungen*

- *Programmierkonzepte und Modellierung und*
- *KI-Programmierung und Wissensrepräsentation“* (vgl. [AR05]).

Aber nicht nur die Erstellung und Verwaltung dieser Lehr- und Lernmaterialien ist von besonderer Bedeutung, sondern auch die Suche nach diesen Materialien.

Im Rahmen dieser Arbeit soll geprüft werden, wie sich die multimedialen Lehrangebote in Plone um ontologiebasierte Metadaten erweitern lassen und wie eine semantische Suche auf Basis dieser Metadaten erfolgen kann. Hierfür sollen die multimedialen Lehrangebote mit fachspezifischen Metadaten aus dem Bereich der Informatik, d. h. Metadaten die zu den oben genannten Lehrveranstaltungen in Verbindung stehen, versehen werden können. Des Weiteren sollen auch die vom Institut im CMS Plone veröffentlichten Dokumente mit Metadaten erweitert werden können, um so eine semantische Suche zu ermöglichen.

Das hier zu entwickelnde Konzept eines semantischen Anfragesystems wurde auf den Name $PlOSS^X$[1] getauft.

6.3 Beschreibung des Anforderungskontextes

Im Folgenden werden die Anforderungen für das oben beschriebene Gesamtkonzept spezifiziert. Zunächst werden die grundlegenden Eigenschaften einer semantischen Suchtechnologie beschrieben. Aus diesen Eigenschaften und dem Gesamtkonzept lassen sich dann die Anforderungen für das hier zu entwickelnde semantische Anfragesystem $PlOSS^X$ ableiten.

6.3.1 Eigenschaften eines ontologiebasierten Anfragesystems

In [Man07a] und [HT06] wurden Ansätze von semantischen Suchtechnologien anhand bestimmter Merkmale und Kriterien klassifiziert. Diese Merkmale und Kriterien werden verwendet, um die grundlegenden Eigenschaften eines ontologiebasierten Anfragesystems zu spezifizieren.

1. **Kopplung:** Ein wichtiges Unterscheidungskriterium ist die Kopplung zwischen Dokument und Wissensstruktur. Die Kopplung ist z. B. entscheidend für die einzusetzende Ontologiesprache und damit auch die Möglichkeit des logischen Schließens. Eine detailliertere Betrachtung

1. Der Name $PlOSS^X$ ist mehr ein Kunstname als ein Akronym und setzt sich aus ***Pl****one* ***O****ntology* ***S****emantic* ***S****earch e****X****tension* zusammen. Damit soll das Konzept ausgedrückt werden, dass mit Hilfe einer Ontologieerweiterung die Inhalte des CMS Plone mit Metainformationen versehen werden können, um so eine semantische Suche auf Basis dieser Metainformationen durchführen zu können.

hinsichtlich der Kopplung wurde in [MCG04] vorgenommen. In dieser Arbeit wird die Kopplung nach zwei grundlegenden Ansätzen unterschieden:

- *Enge Kopplung:* In eng gekoppelten Systemen werden die Metainformationen im Dokument selbst gespeichert, beispielsweise im Header des Dokumentes. Eine Suchmaschine kann dann diese Informationen benutzen, um beispielsweise Nutzer-Anfragen zu beantworten.
- *Lose Kopplung:* Ein System wird als lose gekoppelt bezeichnet, falls die Metainformationen nicht im Dokument selbst, sondern in wissensbasierten Systemen bzw. Ontologien gespeichert werden. Diese Wissensbasis kann dann genutzt werden, um z. B. neues Wissen zu schließen oder um die Konsistenz der Ontologie zu prüfen.

2. **Suchstrategie:** Die Suchstrategie legt die Möglichkeiten der Anfragen an das System fest.

   - *Anfrage in einer Anfragesprache:* D. h. die Anfrage erfolgt in einer formalen Form und in einer speziellen Syntax. Eine detaillierte Erläuterung zu der Anfragesprache SPARQL befindet sich in Abschnitt 5.5.
   - *Native ontologiebasierte Suche:* In den Metainformationen kann native gesucht oder navigiert werden.
   - *Schlüsselwortbasierte Suche:* Es erfolgt eine Indizierung der Metainformationen, damit eine Freitext-Suche angeboten werden kann.
   - *Faceted Browsing:* Mit dem Faceted Browsing wird aus einer großen Menge von Metainformationen schrittweise das zu suchende Ziel eingeschränkt (siehe auch Abschnitt 5.6).

3. **Transparenz:** Die Transparenz eines Systems wird dadurch festgelegt, inwieweit die Benutzerschnittstelle die semantischen Eigenschaften eines Systems reflektiert.

   - *Transparent:* Die Benutzerschnittstelle unterscheidet sich nicht von Benutzerschnittstellen nicht-semantischer Systeme.
   - *Interaktiv:* Der Benutzer wird direkt mit der semantischen Suche des Systems konfrontiert. Das System kann etwa Rückfragen an den Benutzer stellen oder Verbesserungen der Anfrage vorschlagen.
   - *Hybrid:* Standardanfragen werden transparent abgewickelt. Erfahrenen Benutzern wird jedoch die Möglichkeit angeboten, erweiterte Anfrageformulierungen vorzunehmen.

4. **Benutzerkontext:** Die Relevanz eines Dokumentes hängt vom Benutzerkontext ab.

   - *Lernend:* Der Benutzerkontext wird aus der Interaktion mit dem System abgeleitet.
   - *Statisch:* Anfragen werden festgelegten Kategorien zugeordnet und dementsprechend ausgewertet.

5. **Anfragemodifikation:** Auf dem Gebiet der Modifikation von Anfragen existiert eine große Anzahl von Ansätzen. Diese Ansätze lassen sich allgemein in folgende Klassen aufteilen:

   - *Manuell:* Der Benutzer wird mit ausgewählten Teilen der Wissensstruktur konfrontiert und aufgefordert, die Suchanfrage zu verfeinern. Als Beispiel sei hier das Faceted Browsing genannt. Aber auch die manuelle Umformulierung einer formalen Anfrage kann dieser Klasse zugeordnet werden.
   - *Grafisch:* Wissensstruktur und Dokumente werden als Graphenstruktur aufgefasst. Durch die Anfrage werden einzelne Knoten im Graphen selektiert und dann die relevanten Dokumente identifiziert.
   - *Umformulierung:* Das System nimmt selbstständig eine Umformulierung der Anfrage vor. Dabei kann die Anfrage sowohl erweitert als auch verkürzt werden, oder es können einzelne Suchbegriffe ersetzt werden.

6. **Ontologiesprachen:** Beispielsweise sind RDF und OWL Ontologiesprachen (siehe Kapitel 3).

7. **Inferenzsysteme:** Schlussfolgerungssysteme werden zur Erschließung neuen Wissens eingesetzt.

6.3.2 Anforderungen

Folgende Anforderungen an das semantische Anfragesystem $PlOSS^X$ ergeben sich damit:

- Die Anfragen sollen in einer Anfragesprache erfolgen.
- Die Verwendung von Inferenzsystemen soll gewährleistet sein, sodass neu erschlossenes Wissen in die Anfragen mit einfließen kann.
- Die verwendete Ontologie soll erweiterbar sein, sodass zusätzliche Ontologien bzw. Metadaten eingebunden werden können.
- Die Integration in Plone soll gewährleistet sein.
- Das Anfragesystem soll möglichst adaptierbar, d. h. auch in andere Anwendungen (z. B. Content Management Systeme) integrierbar sein.

6.4 Verwandte Ansätze

In diesem Abschnitt werden verwandte Ansätze betrachtet, die sich mit dem CMS Plone und Metainformationen bzw. Ontologien auseinandersetzen. Dazu werden die folgenden Plone Module betrachtet, welche Metainformationen auf verschiedene Art und Weise mit Plone verbinden.

6.4.1 PloneCollections

Um Objekte in Plone ordnerbasiert zusammenfassen zu können, wird das Element *Collections*[2] von Plone bereitgestellt. Mit diesem Element können bereits vorhandene Inhalte nach definierbaren Kriterien zusammengefasst und als Liste dargestellt werden. D. h. es werden keine neuen Inhalte in Collections-Elemente eingefügt, sondern Kriterien anhand der zu den Inhalten abgelegten Metainformationen definiert, nach denen vorhandene Inhalte ausgewählt werden. Wird ein Collections-Element aufgerufen, werden anhand der eingestellten Kriterien die indizierten Inhalte durchsucht und die Treffer angezeigt. Es werden auch die Inhalte angezeigt, welche nach dem Anlegen eines Collections-Elementes hinzugefügt wurden und den eingestellten Kriterien entsprechen.

6.4.2 PloneOntology

PloneOntology[3] verfolgt einen kollaborativen Ansatz zum Aufbau einer Ontologie als Netzwerk von Schlagwörtern:

- Die Begriffe entstehen als Schlagwörter zusammen mit den Veröffentlichungen.
- Schlagwörter können zueinander in Beziehung gesetzt werden.
- Es können Gewichtungen für diese Beziehungen angegeben werden.
- Jeder Nutzer kann Schlagwörter und ihre Beziehungen vorschlagen.
- Die Vorschläge durchlaufen einen Review-Prozess.
- Vorhandene Ontologien lassen sich importieren und erweitern.

Die verwendete OWL-API wurde von den Entwicklern von PloneOntology selbst implementiert und enthält daher nur die von ihnen benötigten Funktionen, damit ist die Anbindung einer externen Anwendung wie beispielsweise eines Schlussfolgerungssystems nicht möglich. Die Suche nach Schlagwörtern kann ausschließlich über die in Plone integrierte Suche erfolgen. Eine Anfragesprache wie z. B. SPARQL wird nicht unterstützt.

6.4.3 ContentLicensing

Die Hauptidee von *ContentLicensing*[4] ist, einen einfachen Weg aufzuzeigen, um Metadaten in maschinenlesbarer Form in Ploneseiten integrieren zu können. Der Ansatz besteht darin, die Metadaten des Plone zugrundeliegenden Webapplication-Servers Zope zu erweitern. ContentLicensing ermöglicht es damit, Metadaten in Form von inline RDF auf Ploneseiten anzubieten. Eine Suche nach diesen erweiterten Metadaten wird von der in Plone integrierten Suche unterstützt.

2. Bis zur Ploneversion 2.5.x hieß dieses Element noch *Smart Folder*, mit der Ploneversion 3.x wurde es in *Collections* umbenannt.
3. `http://plone.org/products/ploneontology/` (Abruf: 26. September 2008)
4. `http://plone.org/products/contentlicensing/` (Abruf: 26. September 2008)

7 Erstellung der Ontologie

Bevor die Möglichkeiten der semantischen Suche untersucht werden können, muss eine geeignete Ontologie für dieses spezielle Problem erstellt werden. Wie bereits in Abschnitt 4.2 erläutert wurde, kann bei der Erstellung einer Ontologie nach verschiedenen Methoden vorgegangen werden. Die in Abschnitt 4.2 vorgestellte Methode von Staab kann in dieser Arbeit nicht verwendet werden, da diese vor allem eine allgemeine Vorgehensweise für die Ontologieentwicklung, unter Benutzung von Methoden des Software Engineering, bereitstellt. Daher wird die Methode von Noy und McGuinness verwendet, welche einen konkreteren und dabei intuitiven Ansatz bietet (vgl. [NM01]). Die Autoren weisen auf zwei wesentliche Aspekte hin, die bei der Ontologieentwicklung berücksichtigt werden sollen (vgl. [NM01]):

- *„There is no one correct way to model a domain – there are always viable alternatives. ...“* D. h. es existieren immer mehrere Alternativen, um eine Domäne zu modellieren. Diese unterschiedlichen Varianten ergeben sich aus den verschiedenen Entscheidungen bei der Ontologieerstellung, wie beispielsweise den Einsatzzielen der Ontologie oder aus der Betrachtungsweise der Domäne.

- *„Ontology development is necessarily an iterative process.“* Aus diesem Grund wird die entwickelte Ontologie als prototypisch betrachtet. Sie soll durch Evaluation an den vorgesehenen Anwendungsfällen sowie durch Diskussion überprüft und schrittweise an die Anforderungen angepasst werden. Nach [NM01] zieht sich dieser iterative Prozess durch den ganzen Lebenszyklus einer Ontologie.

Noy und McGuinness definieren folgende Schritte, um eine initiale Ontologie zu erzeugen (vgl. [NM01]):

- Bestimmung des Anwendungsbereichs und des Umfangs der Ontologie (dies wurde bereits in Abschnitt 6.2 erläutert).

- Prüfen, ob existierende Ontologien wiederverwendet werden können.

- Identifikation der benötigten Konzepte, Eigenschaften und Relationen sowie die Festlegung der dazugehörigen Hierarchien und Restriktionen.

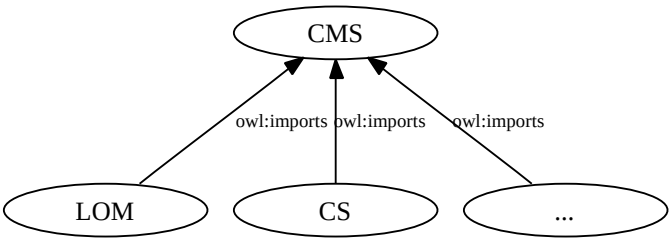
- Erstellung von Instanzen.

7.1 Wiederverwendung von Ontologien

Als Quelle für existierende Ontologien wurde neben Standards und aktueller Literatur auch das Semantic Web herangezogen. Dieses bietet eine sehr umfangreiche Sammlung an frei verfügbaren Ontologien, wobei aber für bestimmte Domänen oft verschiedene Ontologien existieren. Ontologien können sich sowohl in der verwendeten Ontologiebeschreibungssprache als auch in der Umsetzungsart unterscheiden. Die Grundlage für die Auswahl einer Ontologie bildeten die semantische Suchmaschine Swoogle[1], die auf der Webseite Ping the Semantic Web[2] veröffentlichte Übersicht zu den gebräuchlichsten Namespaces und Google[3].

Auf Basis des in Abschnitt 6.2 beschriebenen Konzeptes konnten folgende Teilontologien festgestellt werden (Abbildung 7.1 zeigt den schematischen Aufbau).

Abbildung 7.1: Import-Relationen (owl:imports) der Teilontologien



Die Teilontologien decken dabei die folgenden Bereiche ab:

- **CMS** stellt die CMS-spezifischen Konzepte bereit.
- **LOM** enthält die semantische Beschreibung von Lernobjekten.
- **CS** umfasst die fachspezifischen Konzepte der Informatik.
- **...** deutet an, dass die CMS-Ontologie um andere Ontologien erweitert werden kann.

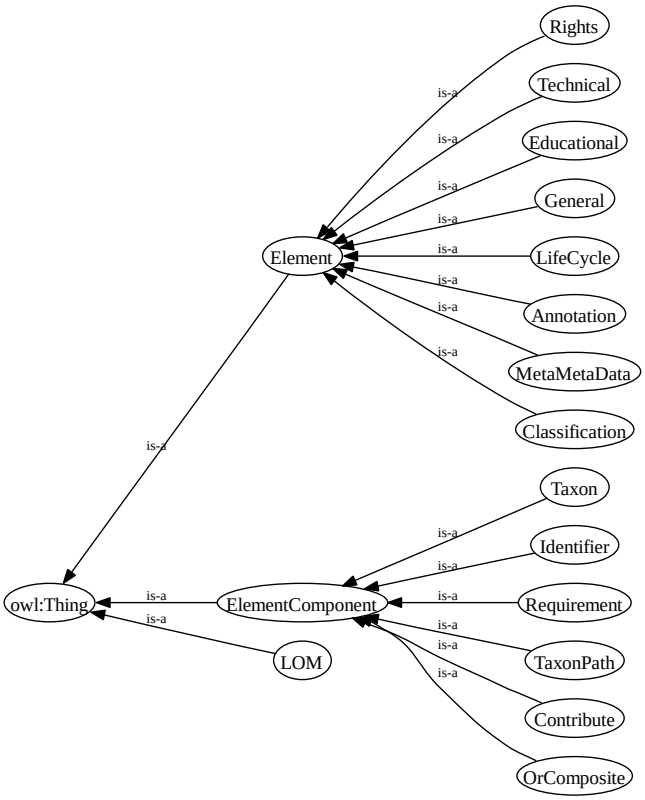
Bei der Verwendung von bereits existierenden Ontologien wurden diese nicht vollständig übernommen, sondern die Terme wurden lediglich ausgeliehen und in Bereichen, wo es angebracht war, durch Äquivalenz-Statements ergänzt, um so die Austauschbarkeit zu verbessern.

7.2 LOM-Ontologie

Der LOM-Standard wurde definiert, um die Nutzung von Lernobjekten zu erleichtern (vgl. Abschnitt 2.3.2). In dieser Arbeit wird die LOM-Ontologie dazu verwendet, die durch die eduComponents bereitgestellten Lehr- und Lernmaterialien ausführlich zu beschreiben. Abbildung 7.2 zeigt das Klassen-Konzept der LOM-Ontologie.

1. `http://swoogle.umbc.edu/` (Abruf: 26. September 2008)
2. `http://pingthesemanticweb.com/` (Abruf: 26. September 2008)
3. `www.google.com` (Abruf: 26. September 2008)

Abbildung 7.2: Taxonomie der LOM-Ontologie



Die LOM-Ontologie wurde nicht vollständig neu entwickelt, sondern es wurde eine bereits existierende Ontologie überarbeitet und wiederverwendet.[4] Die Überarbeitung bestand in der Anpassung der in den Relationen verwendeten Datentypen und Werte, da diese nicht vollständig dem LOM-Standard entsprachen. Des Weiteren wurde die Klasse *Relation* mit ihren definierten Eigenschaften entfernt. Diese wurde zwar laut LOM-Standard exakt umgesetzt, hatte jedoch in der Ontologie nicht die erforderliche semantische Bedeutung, um direkte, inverse oder transitive Beziehungen zwischen Lernobjekten herstellen zu können. Um nun Inhaltstypen zueinander in Beziehung setzen zu können, wurden die Relation-Eigenschaften von LOM in der Klasse ContentTypes der CS-Ontologie definiert. Damit können die Inhaltstypen zueinander in Relation gesetzt werden, wobei inverse und transitive Abhängigkeiten beachtet werden.

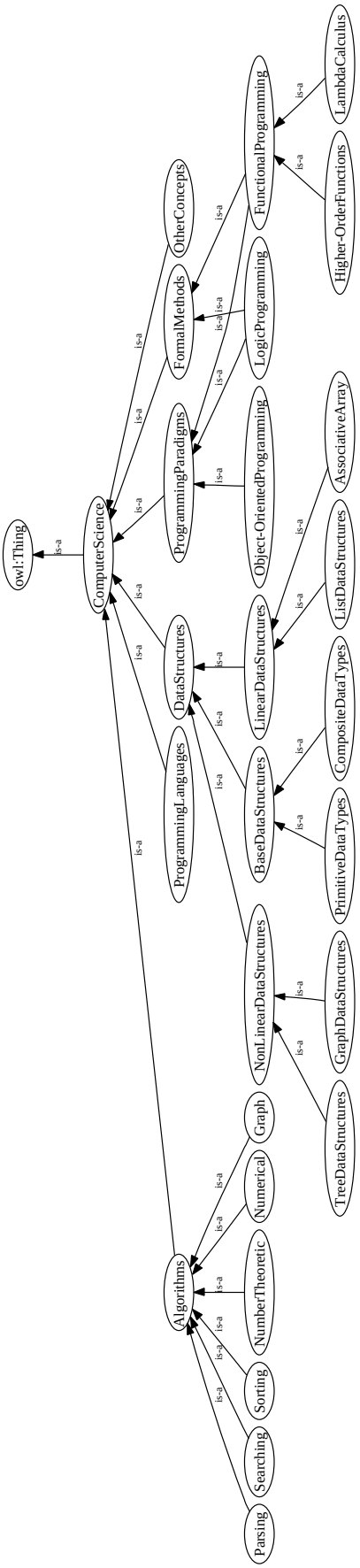
Als Eigenschaft der Klasse LOM wurde *hasElement* definiert. Mit hasElement können Instanzen der Klassen *General*, *LifeCycle*, *MetaMetaData*, usw. (Subklassen der Klasse *Element*) mit Instanzen der Klasse LOM in Beziehung gesetzt werden. Die Datentypen und Werte, die von den Instanzen der Element-Subklassen benötigt werden, wurden analog des LOM-Standards in den Subklassen der Klasse *ElementComponent* definiert und werden gegebenenfalls bei einer Verwendung detaillierter erläutert.

7.3 CS-Ontologie

Nach Abschnitt 7.1 sind in der CS-Ontologie die fachspezifischen Konzepte der Informatik definiert, welche die multimedialen Lehrangebote des Institutes für Wissens- und Sprachverarbeitung an der Otto-von-Guericke-Universität

4. `http://users.teilar.gr/~hartonas/lom.owl` (Abruf: 26. September 2008)

Abbildung 7.3: Taxonomie der CS-Ontologie



Magdeburg beschreiben. Dies sollen im speziellen Konzepte über funktionale Programmierung sein, um die dort vorhandenen Übungsaufgaben ausführlicher mit Metainformationen auszuzeichnen.

Auch nach ausführlicher Recherche konnte keine Ontologie eruiert werden, in welcher die Konzepte der Informatik über funktionale Programmierung definiert sind. Aus diesem Grund wurde die CS-Ontologie neu erarbeitet.

Zunächst erfolgte die Identifizierung der benötigten Klassen, hierbei wurde auf das Kategorie-Konzept von Wikipedia[5] zurückgegriffen. Nach diesem Kategorie-Konzept kann jeder Artikel verschiedenen Kategorien und jede Kategorie mehr als einer Superkategorie zugeordnet sein. Weiterhin existieren verschiedene Kategorisierungs-Schemata gleichzeitig. D. h. *„... categories do not form a strict hierarchy or tree structure, but a more general directed acyclic graph (or close to it)“* (vgl. [Wik08]). Des Weiteren ist zu beachten, dass in Wikipedia zu einem Begriff sowohl eine Kategorie als auch ein Artikel existieren kann. Der Detaillierungsgrad der zu entwickelnden Ontologie hat einen entscheidenden Einfluss darauf, wie dieser Begriff in der Ontologie umgesetzt wird.

Die Taxonomie der CS-Ontologie wurde auf Basis dieses gerichteten azyklischen Graphen entwickelt. Abbildung 7.3 zeigt das Klassen-Konzept der CS-Ontologie. Als Beispiel sei hier die Wikipedia-Kategorie *Functional programming* genannt, welche u. a. die Subkategorien *Higher-order functions* und *Lambda calculus* enthält. Dies wurde in der CS-Ontologie durch die Klasse *FunctionalProgramming* mit den Subklassen *LambdaCalculus* und *Higher-OrderFunctions* umgesetzt. Als Instanzen der Klasse Higher-OrderFunctions wurden analog Wikipedia u. a. die Entitäten *Apply*, *Filter*, *Fold* und *Map* angelegt. In Anhang B Tabelle B.1 sind sämtliche in der CS-Ontologie angelegten Instanzen aufgelistet.

Die Autoren Suchanek et al. beschreiben in [SKW07] einen anderen Ansatz, um eine Taxonomie aus den Kategorien von Wikipedia zu erzeugen. Sie untersuchten dabei explizit konkrete Wikipedia-Artikel über Personen und stellten fest, dass der gerichtete azyklische Graph eine Kategorie-Hierarchie erzeugt, welche lediglich die thematische Struktur der Wikipedia-Artikel widerspiegelt. Aus diesem Grund verwendeten Suchanek et al. ausschließlich die Blätter der Kategorie-Hierarchie, um in Verbindung mit WordNet[6] eine Taxonomie zu entwickeln. Der Ansatz von Suchanek et al. konnte für die Entwicklung der CS-Ontologie nicht verwendet werden, da WordNet nicht die hier benötigten Konzepte der Informatik über funktionale Programmierung enthält.

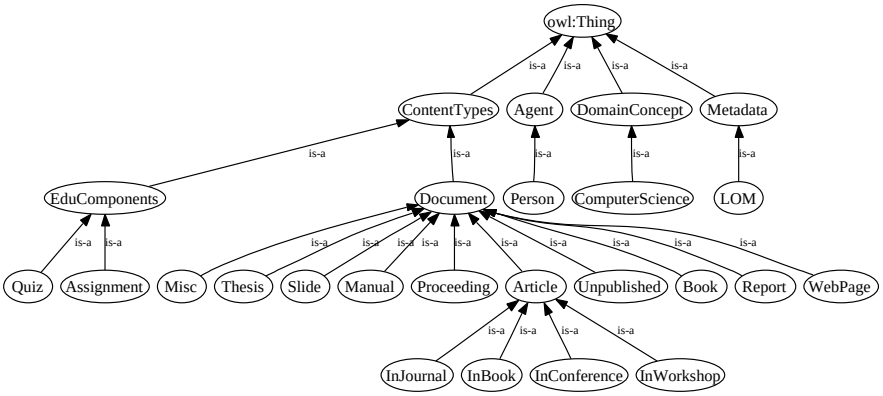
7.4 CMS-Ontologie

Die CMS-Ontologie importiert alle anderen Ontologien und führt diese damit zusammen. Sie kann somit als *Upper Ontology*[7] betrachtet werden. Die in der CMS-Ontologie definierten Klassen spiegeln erstens die Inhaltstypen des Content Management Systems Plone wider und werden zweitens für die Abbildung zwischen Ontologie und Plone benötigt. Abbildung 7.4 zeigt alle

5. `http://en.wikipedia.org/` (Abruf: 26. September 2008)
6. `http://wordnet.princeton.edu/` (Abruf: 26. September 2008)
7. Das Vokabular einer Upper Ontology ist auf einem sehr allgemeinen Niveau, kann aber über einen Domänenbezug verfügen. Eine Top-Level Ontology enthält im Gegensatz dazu sehr allgemeine Konzepte (z. B. Raum, Zeit, Materie etc.), die unabhängig von einem bestimmten Anwendungsbereich sind. (vgl. [BR06])

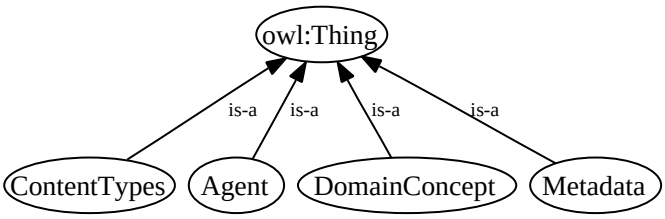
Klassen der CMS-Ontologie, die für diese Arbeit letztendlich als relevant eingeschätzt wurden. Aus Gründen der Übersichtlichkeit wurde in Abbildung 7.4 auf die importierten Ontologien verzichtet. In Anhang A Abbildung A.1 ist die komplette Taxonomie der CMS-Ontologie abgebildet.

Abbildung 7.4: Taxonomie der CMS-Ontologie



Die Klassen der im Rahmen des Konzeptes *PlOSS*^{X} erarbeiteten CMS-Ontologie, sind in vier Kategorien aufgeteilt. Diese Kategorien entsprechen den in Abbildung 7.5 dargestellten Superklassen und werden in den folgenden Abschnitten einzeln erläutert.

Abbildung 7.5: Taxonomie der Superklassen der CMS-Ontologie



7.4.1 ContentTypes

Die Inhaltstypen des CMS werden durch die Klasse *ContentTypes* repräsentiert, sie enthält damit die gesamte Beschreibung der Inhaltstypen. Subklassen dieser Klasse sind die Klassen *EduComponents* und *Document*, die eine Kategorisierung der Inhaltypen ermöglichen. Die Klassen *Quiz* und *Assignment* wurden als Subklassen der Klasse EduComponents definiert, wobei Instanzen dieser Subklassen den entsprechenden Instanzen der Inhaltstypen des CMS entsprechen. Die Klasse Document wurde ebenfalls untergliedert, hierbei wurde der bibliographische Standard BIBTEX als Grundlage für die Bildung der Taxonomie verwendet (siehe Abbildung 7.4).

Da die Inhaltstypen eines CMS verschiedene Inhalte beschreiben können, wurde für die Inhaltstypen eine grundlegende Menge an Eigenschaften auf der Basis von DC definiert (vgl. Abschnitt 2.3.1). Die Entscheidung über die endgültige Menge an Eigenschaften wurde nach einigen Tests im Zusammenhang mit dem Anfragesystem getroffen. Eine Eigenschaft wurde dabei als Notwendigkeit für das in Abschnitt 6.2 beschriebene Gesamtkonzept *PlOSS*^{X} identifiziert: *uid*. Über die Eigenschaft uid wird eine eindeutige Zuordnung einer im CMS enthaltenen Instanz eines Inhaltstypen zu einer Instanz aus der Ontologie sichergestellt. Dieser Ansatz gehört zur Hauptidee, welche hinter dem hier beschriebenem Konzept *PlOSS*^{X} steckt.

Tabelle 7.1: Eigenschaften der Klasse ContentTypes

Name	Wertebereich	Wertrestriktion
uid	xsd:anyURI	Functional
dc:creator	Agents	Inverse hasCreated
dc:date	xsd:date	keine
dc:description	xsd:string	keine
dc:identifier	xsd:string	keine
dc:title	xsd:string	keine
dcterm:hasFormat	ContentTypes	Inverse dcterm:isFormatOf
dcterm:hasPart	ContentTypes	Inverse dcterm:isPartOf, Transitive
dcterm:hasVersion	ContentTypes	Inverse dcterm:isVersionOf
dcterm:isBasedOn	ContentTypes	Inverse dcterm:isBasisFor
dcterm:isBasisFor	ContentTypes	Inverse dcterm:isBasedOn
dcterm:isFormatOf	ContentTypes	Inverse dcterm:hasFormat
dcterm:isPartOf	ContentTypes	Inverse dcterm:hasPart, Transitive
dcterm:isReferencedBy	ContentTypes	Inverse dcterm:references
dcterm:isRequiredBy	ContentTypes	Inverse dcterm:requires, Transitive
dcterm:isVersionOf	ContentTypes	Inverse dcterm:hasVersion
dcterm:references	ContentTypes	Inverse dcterm:isReferencedBy
dcterm:requires	ContentTypes	Inverse dcterm:isRequiredBy, Transitive

In Tabelle 7.1 sind alle definierten Eigenschaften der Klasse ContentTypes aufgelistet. Die Klasse EduComponents hat zusätzlich, zu den von der Klasse ContentTypes geerbten Eigenschaften, noch die Eigenschaften *hasMetadata* mit dem Wertebereich LOM und *hasConcept* mit dem Wertebereich ComputerScience. Als entsprechende inverse Relationen wurden die Eigenschaften *isMetadataIn* und *isConceptIn* definiert. Mit diesen Eigenschaften können die Instanzen der Inhaltstypen in der Ontologie mit LOM-Metadaten und Konzepten aus dem Bereich der Informatik in Relation gesetzt werden.

7.4.2 Agent

Die Klasse *Agent* dient der Modellierung von Personen. Hier wurde lediglich die Klasse *Person* als Subklasse der Klasse Agent definiert.

Tabelle 7.2 zeigt die Eigenschaften der Klasse Person. Die Eigenschaft *hasCreated* wird dabei von der Klasse Agent geerbt. Durch das Konzept der Vererbung können einmal in einer Klasse definierte Eigenschaften an die jeweiligen Subklassen vererbt werden. Die auf diese Weise vererbten Eigenschaften werden in eckigen Klammern eingefasst dargestellt.

Tabelle 7.2: Eigenschaften der Klasse Person

Name	Wertebereich	Wertrestriktion
firstName	xsd:string	keine
lastName	xsd:string	keine
[hasCreated]	ContentTypes	Inverse dc:creator

Als Instanzen der Klasse Person wurden Mitarbeiter des Institutes angelegt, welche die Übungsaufgaben und Dokumente im CMS Plone erstellten.

7.4.3 DomainConcept

Die Superklasse *DomainConcept* fasst die fachspezifischen Konzepte zusammen. Als Subklasse der Klasse DomainConcept wurde die Klasse *ComputerScience* definiert, welche die fachspezifischen Konzepte der Informatik enthält. In Tabelle 7.3 ist die Eigenschaft der Klasse ComputerScience dargestellt.

Tabelle 7.3: Eigenschaft der Klasse ComputerScience

Name	Wertebereich	Wertrestriktion
isConceptIn	EduComponents	Inverse hasMetadata

Durch das OWL-Äquivalenz-Statement *owl:equivalentClass* wurde die Klasse ComputerScience der CS-Ontologie (siehe Abschnitt 7.3) mit der Klasse ComputerScience der CMS-Ontologie verbunden.

7.4.4 Metadata

Unter der Superklasse *Metadata* werden die Klassen zusammengefasst, die es ermöglichen, Inhaltstypen mit Metadaten auszuzeichnen. Nach Abschnitt 7.1 konnte vorerst nur der Metadatenstandard LOM identifiziert und damit die Klasse *LOM* als Subklasse der Klasse Metadata definiert werden. Tabelle 7.4 zeigt die Eigenschaften der Klasse LOM.

Tabelle 7.4: Eigenschaften der Klasse LOM

Name	Wertebereich	Wertrestriktion
isMetadataIn	EduComponents	Inverse hasMetadata
[lom:hasElement]	lom:Element	keine

Die Klasse LOM der LOM-Ontologie (siehe Abschnitt 7.2) wurde durch das OWL-Äquivalenz-Statement *owl:equivalentClass* mit der Klasse LOM der CMS-Ontologie verbunden.

7.5 Erstellen einer Wissensbasis

Auf Grundlage der zuvor definierten CMS-Ontologie wurde eine Wissensbasis aufgebaut. Voraussetzung war hierfür eine umfangreiche Recherche im Content Management System Plone der AG *Wissensbasierte Systeme und Dokumentverarbeitung*[8] an der Otto-von-Guericke-Universität Magdeburg, welches die dort veröffentlichten Übungsaufgaben und Dokumente enthält.

Da die vorhandenen Metadaten, welche die Übungsaufgaben und Dokumente beschreiben, eng mit den Ploneinstanzen verbunden sind, war eine automatische Extraktion nicht möglich. Daher erfolgte die Identifizierung der Metadaten und das Erstellen der Instanzen in der CMS-Ontologie manuell.

Zum Beispiel sei hier die Übungsaufgabe *Newton-Verfahren* aus dem PKM-Repository[9] genannt. Als Metadaten dieser Übungsaufgabe wurde der Instanznamen *newton-Verfahren*, das Label *Newton-Verfahren*, die uid *newton-verfahren-haskell*, der Ersteller *RoesnerDietmar*, das Erstellungsdatum *2007-07-03* und die Konzepte *cs:Haskell, cs:NewtonsMethod, cs:SquareRoot* ermit-

8. `http://wdok.cs.uni-magdeburg.de/` (Abruf: 26. September 2008)
9. `http://wdok.cs.uni-magdeburg.de/studium-und-lehre/pkm-repository-1` (Abruf: 26. September 2008) Für den Zugang zum Übungsaufgaben-Repository der Lehrveranstaltung *Programmierkonzepte und Modellierung (PKM)* wird eine Authentifikation benötigt.

telt. Eine Auflistung der angelegter Instanzen unterhalb der Klasse Content-Types der CMS-Ontologie wird in Anhang B Tabelle B.2 gegeben.

7.6 Anforderungen an eine $PlOSS^X$-konforme Ontologie

Obwohl für $PlOSS^X$ eine Datenunabhängigkeit bezüglich der verwendeten Ontologie angestrebt wurde, konnte dieser Anspruch nicht vollständig aufrechterhalten werden. Folgende Beschränkungen haben sich in der Entwicklungsphase des Anfragesystems ergeben und sind bei einer Ontologieerstellung zu beachten, damit diese Ontologie später als $PlOSS^X$-konform verwendet werden kann:

- Mit Verwendung der **OWL-Syntax in Form von OWL DL** wird eine erfolgreiche Verarbeitung der Ontologie durch Schlussfolgerungssysteme gewährleistet.
- Durch die **Eigenschaft uid** wird eine eindeutige Abbildung zwischen den Instanzen eines CMS und den Instanzen in der Ontologie sichergestellt.
- *optional:* Anlegen einer **Wurzelklasse**, unter welcher die Inhaltstypen des CMS zusammengefasst werden können.

Werden alle oben aufgeführten Punkte beachtet, kann im Prinzip jede beliebige, auf diese Weise erstellte, Ontologie in ein CMS eingebunden werden.

Plone verwendet zur Identifikation von Objekten eindeutige Schlüssel, sogenannte UIDs. Wird in Plone ein Objekt z. B. verschoben oder umbenannt, führt dies zu einer Änderung der URI des Objektes, über die UID kann es aber jederzeit referenziert werden. Aus diesem Grund sollte bei einer Verwendung von Plone in Verbindung mit $PlOSS^X$, die UID eines Ploneobjektes zur eindeutigen Adressierung herangezogen und als Wert der Eigenschaft uid gespeichert werden.

8 Semantische Suchanfragen

In diesem Kapitel wird die semantische Suche anhand von Anwendungsfällen und Beispielen detaillierter betrachtet. Die in Kapitel 7 erstellte Ontologie und die in Anhang B beschriebenen Instanzen sind dabei Wissensbasis für die hier beschriebenen Suchanfragen.

Um das Konzept der semantischen Suche bzw. semantischer Suchanfragen im Kontext dieser Arbeit besser einordnen zu können, wird der Begriff „semantische Suche“ in folgender Definition verwendet:

„Semantic Search is a process of information access, where one or several activities can be supported by a set of functionalities enabled by semantic technologies.“[1]

Die Terminologie hat die folgende Bedeutung:

- **Informationen** sind Dokumente bzw. Fakten, welche in der Wissensbasis enthalten sind.
- **Semantische Technologien** sind die Technologien, welche durch die Semantic Web Architektur des W3C bereitgestellt werden, z. B. Anfragesprachen, Reasoner und Regeln (vgl. Kapitel 3).

8.1 Anfrageform

Durch die Semantic Web Architektur des W3C werden verschiedene Möglichkeiten zur Verfügung gestellt, um Anfragen an Ontologien durchführen zu können (vgl. [HKRS08]):

- Mit **SPARQL** als Anfragesprache können Anfragen an RDF-Graphen formuliert werden.
- **Konjunktive Anfragen** bzw. **Regeln** können als grundlegender Formalismus betrachtet werden, um in komplexen Ontologiesprachen wie OWL DL Anfragen formulieren zu können.

1. Die Definition wurde den Vorlesungsunterlagen *Semantic Web Technologies II* der Universität Karlsruhe entnommen (`http://semantic-web-grundlagen.de/index.php/SWebT2_SS08` (Abruf: 26. September 2008)).

8.2 Negation

Die *Negation* stellt bei Anfragen an eine Ontologie ein konzeptionelles Problem dar, da für die Interpretation dieser Daten die sogenannte *Open World Assumption (OWA)* gilt. OWA bedeutet, dass solange etwas nicht als „wahr“ ausgesagt wurde, nicht angenommen werden kann, dass es „falsch“ ist – es wird lediglich angenommen, dass das Wissen noch nicht zur Wissensbasis hinzugefügt wurde.

8.3 SPARQL

8.3.1 SPARQL-Prolog

Um die SPARQL-Anfragen platzsparend und übersichtlich darstellen zu können, wurde jeweils der Prolog mit den PREFIX-Angaben für die Präfixspezifikation zur Festlegung der Namespaces weggelassen. Dieser Anfragekopf ist bei allen Anfragen derselbe und ist in Listing 8.1 abgebildet.

Listing 8.1: Präfixe der verwendeten Namespaces

```
1 PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX owl:      <http://www.w3.org/2002/07/owl#>
4 PREFIX xsd:      <http://www.w3.org/2001/XMLSchema#>
5 PREFIX dc:       <http://purl.org/dc/elements/1.1/>
6 PREFIX dcterms:  <http://purl.org/dc/terms/>
7 PREFIX :         <http://wdok.cs.uni-magdeburg.de/ontologies/↘
  →CMS.owl#>
8 PREFIX cs:       <http://wdok.cs.uni-magdeburg.de/ontologies/↘
  →ComputerScience.owl#>
9 PREFIX lom:      <http://www.owl-ontologies.com/lom.owl#>
```

8.3.2 Einfache Anfrage

Bereits bei einfachen Anfragen kann der Einsatz eines Reasoners eine wesentliche Auswirkung auf die Ergebnismenge ausüben. Dies sei beispielsweise durch die in Listing 8.2 abgebildete SPARQL-Anfrage gezeigt. Mit dieser Anfrage wird nach dem Label sämtlicher Subklassen der Klasse ContentTypes der CMS-Ontologie (siehe Abbildung 7.3) gefragt, wobei eine Sortierung anhand des Labels vorgenommen werden soll. Die Suchanfrage für diese SPARQL-Anfrage kann wie folgt formuliert werden:

„Welche Kategorien stehen für eine Klassifizierung der Inhaltstypen zur Verfügung?“

Listing 8.2: Subklassen von ConentTypes

```
1 SELECT ?subClOfContentTypes ?label
2 WHERE  { ?subClOfContentTypes rdfs:subClassOf :ContentTypes .
3          ?subClOfContentTypes rdfs:label      ?label
4        } ORDER BY ?label
```

Listing 8.3 zeigt die Ergebnismenge dieser Anfrage, ohne die Verwendung eines Reasoners. Der Ergebnismenge konnten einzig die direkten Subklassen der Klasse ContentTypes zugeordnet werden.

Listing 8.3: Ergebnismenge: Subklassen von ConentTypes ohne Reasoner

```
 ?subClOfContentTypes | ?label
======================================
 Document             | "Document"
 EduComponents        | "eduComponents"
```

Wird ein Reasoner bei dieser Anfrage verwendet, können sämtliche Subklassen der Klasse ContentTypes durch den Reasoner geschlussfolgert und der Ergebnismenge hinzugefügt werden (siehe Listing 8.4).

Listing 8.4: Ergebnismenge: Subklassen von ConentTypes mit Reasoner

```
 ?subClOfContentTypes | ?label
================================================
 Article              | "Article"
 InBook               | "Article␣in␣book"
 InConference         | "Article␣in␣conference"
 InJournal            | "Article␣in␣journal"
 InWorkshop           | "Article␣in␣workshop"
 Assignment           | "Assignment"
 Book                 | "Book"
 ContentTypes         | "Content␣types"
 Document             | "Document"
 Manual               | "Manual"
 Misc                 | "Misc"
 Proceeding           | "Proceeding"
 Quiz                 | "Quiz"
 Report               | "Report"
 Slide                | "Slide"
 Thesis               | "Thesis"
 Unpublished          | "Unpublished"
 WebPage              | "Web␣page"
 EduComponents        | "eduComponents"
```

8.3.3 Oder-Verknüpfung

Um verschiedene alternative Pattern darzustellen, können zwei gruppierende Graph-Pattern mit Hilfe des Schlüsselwortes UNION miteinander verknüpft werden (vgl. Abschnitt 5.5). Dieser Ausdruck entspricht einem *logischen Oder*, d. h. jedes Ergebnis muss mit einem der angegebenen Pattern übereinstimmen, darf aber durchaus auch auf beide passen. Die folgende Suchanfrage beschreibt die in Listing 8.5 abgebildete SPARQL-Anfrage einer Oder-Verknüpfung:

*„Welchen Übungsaufgaben sind die Konzepte Apply **oder** List zugeordnet?“*

Listing 8.5: Anfrage mit Oder-Verknüpfung

```
SELECT ?assignment ?title
WHERE  { { ?assignment rdf:type :Assignment .
           ?assignment dc:title ?title }
         { { ?assignment :hasConcept cs:Apply } UNION
           { ?assignment :hasConcept cs:List } }
       } ORDER BY ?title
```

Die Ergebnismenge dieser Anfrage ist in Listing 8.6 abgebildet. Hierbei wurden sämtliche Übungsaufgaben, denen entweder das Konzept Apply oder das Konzept List zugeordnet wurde, in die Ergebnismenge aufgenommen.

Listing 8.6: Ergebnismenge der Anfrage mit Oder-Verknüpfung

```
 ?assignment            | ?title
=================================================
 addToEach              | "Add to each"
 applyAll-composeList   | "Apply all - compose list"
 applyList              | "Apply list"
 length-mit-map         | "Length with map"
 mapThree               | "Map three"
 partition              | "Partition"
 quicksort              | "Quicksort"
 searchTree-1           | "SearchTree (1)"
```

8.3.4 Und-Verknüpfung

Ein *logisches Und* kann durch die Konkatenation zweier gruppierender Graph-Pattern ausgedrückt werden, d. h. jedes Ergebnis muss mit jedem der angegebenen Pattern übereinstimmen (vgl. Abschnitt 5.5). In Listing 8.7 ist die SPARQL-Anfrage abgebildet, welche durch die folgende Suchanfrage beschrieben werden kann:

*„Welchen Übungsaufgaben sind die Konzepte Apply **und** List zugeordnet?“*

Listing 8.7: Anfrage mit Und-Verknüpfung

```
SELECT ?assignment ?title
WHERE  { { ?assignment rdf:type :Assignment .
           ?assignment dc:title ?title }
         { { ?assignment :hasConcept cs:Apply } .
           { ?assignment :hasConcept cs:List } }
       } ORDER BY ?title
```

Listing 8.8 zeigt die Ergebnismenge dieser Anfrage. Hierbei wurden sämtliche Übungsaufgaben in die Ergebnismenge aufgenommen, denen sowohl das Konzept Apply als auch das Konzept List zugeordnet wurde.

Listing 8.8: Ergebnismenge der Anfrage mit Und-Verknüpfung

```
 ?assignment            | ?title
=================================================
 applyAll-composeList   | "Apply all - compose list"
 applyList              | "Apply list"
```

8.3.5 Komplexe Anfragen

Komplexe Anfragen können durch eine Kombination von Und- oder Oder-Verknüpfungen, durch die Verwendung von optionalen Graph-Pattern und durch den Einsatz von Filter-Funktionen und Modifikatoren gebildet werden (vgl. Abschnitt 5.5). Weiterhin kann durch die Verwendung eines Reasoners zusätzlich erschlossenes Wissen in die Ergebnismenge mit einfließen.

Ein Beispiel für eine SPARQL-Anfrage, welche die beiden obigen Anfragekonzepte aus Listing 8.5 und 8.7 miteinander kombiniert zeigt Listing 8.9. Folgende Suchanfrage beschreibt diese SPARQL-Anfrage:

*„Welchen Übungsaufgaben sind die Konzepte Apply **oder** List **und** das Konzept Lambda-Calculus zugeordnet?“*

Die Ergebnismenge dieser Anfrage ist in Listing 8.10 dargestellt. Hierbei wurden sämtliche Übungsaufgaben der Ergebnismenge hinzugefügt, denen sowohl

entweder das Konzept Apply oder List als auch das Konzept Lambda-Calculus zugeordnet wurde.

Listing 8.9: Anfrage mit Oder-Und-Verknüpfung

```
1 SELECT ?assignment ?title
2 WHERE  { { ?assignment rdf:type :Assignment .
3            ?assignment dc:title ?title }
4          { { ?assignment :hasConcept cs:Apply } UNION
5            { ?assignment :hasConcept cs:List } .
6            { ?assignment :hasConcept cs:Lambda-Calculus } }
7        } ORDER BY ?title
```

Listing 8.10: Ergebnismenge der Anfrage mit Oder-Und-Verknüpfung

```
1 ?assignment | ?title
2 ============================
3 applyList   | "Apply list"
4 mapThree    | "Map three"
```

Listing 8.11: Anfrage nach Inhaltstypen mit FP-Konzepten

```
1 SELECT DISTINCT ?contentTypes ?title
2 WHERE  { ?labelFP   rdfs:label   "Functional programming" .
3          ?concepts  rdf:type     ?labelFP .
4          ?concepts  :isConceptIn ?contentTypes .
5          ?contentTypes dc:title ?title
6        } ORDER BY ?title
```

Ein weiteres Beispiel für die Verwendung eines Reasoners zeigt die folgende Suchanfrage:

*„Welchen Übungsaufgaben sind Konzepte aus dem Bereich **„Functional programming“** zugeordnet?“*

In Listing 8.11 ist diese Suchanfrage als SPARQL-Anfrage abgebildet, dabei ist Folgendes zu beachten:

1. Zeile 2 bestimmt die Klassen, welche den Wert „Functional programming“ im Label besitzen.
2. Mit dem Ausdruck in Zeile 3 werden sämtliche Konzept-Instanzen zu den in (1.) bestimmten Klassen ermittelt, wobei durch die Verwendung eines Reasoners auch die Instanzen in den Subklassen dieser Klassen berücksichtigt werden.
3. Mit Zeile 4 werden alle Instanzen von Inhaltstypen ermittelt, denen eine oder mehrere Instanzen aus (2.) zugeordnet sind.
4. Durch den Ausdruck in Zeile 5 wird der Titel zu den in (3.) ermittelten Instanzen bestimmt.

Listing 8.12 zeigt die Ergebnismenge dieser SPARQL-Anfrage.

8.3.6 Negation in SPARQL

Die SPARQL-Spezifikation empfiehlt die Negation in einer SPARQL-Anfrage durch *Negation As Failure* auszudrücken (vgl. [W3C08c]). Dies erfolgt erstens dadurch, dass sämtliche Instanzen, an denen die zu negierende Eigenschaft

Listing 8.12: Ergebnismenge der Anfrage nach Inhaltstypen mit FP-Konzepten

```
 ?contentTypes          | ?title
================================================
 accumulate             | "Accumulate"
 applyAll-composeList   | "Apply all - compose list"
 applyList              | "Apply list"
 length-mit-map         | "Length with map"
 mapThree               | "Map three"
```

gebunden ist, durch das Schlüsselwort *OPTIONAL* ausgewählt werden. Und zweitens dann diese ermittelten Instanzen daraufhin getestet werden, ob sie an einen bzw. keinen Wert gebunden sind.

Ein Nachteil dieser Art der Negation besteht darin, dass diese ausschließlich für *funktionale Eigenschaften* verwendet werden kann und einer funktionalen Eigenschaft kann lediglich ein einzelner Wert zugeordnet werden.

In Listing 8.13 ist eine SPARQL-Anfrage abgebildet, welche die Negation durch Negation As Failure zeigt. Die entsprechende Suchanfrage kann folgendermaßen formuliert werden:

*„Welchen Übungsaufgaben ist **keine** UID zugeordnet?“*

Listing 8.13: Negation As Failure

```
SELECT ?assignment ?title
WHERE  { { ?assignment rdf:type :Assignment .
           ?assignment dc:title ?title }
         { OPTIONAL  { ?assignment :uid ?uid } .
           FILTER ( !bound( ?uid ) ) }
       } ORDER BY ?title
```

Die Ergebnismenge dieser Anfrage ist leer, da sämtlichen Instanzen der Klasse Assignment eine eindeutige ID zugeordnet wurde.

8.3.7 Negation by divide

Wurde z. B. eine Suchanfrage folgendermaßen beschrieben:

*„Welchen Übungsaufgaben ist das Konzept List aber **nicht** das Konzept Apply zugeordnet?“*

Dann schlägt die Negation durch Negation As Failure fehl, da die Eigenschaft hasConcept nicht als funktional in der Ontologie definiert wurde. Um diese Anfrage aber trotzdem ausführen zu können, sind folgende Schritte erforderlich:

1. Die Anfrage wird in zwei Anfragen unterteilt und bildet damit zwei Ergebnismengen.
2. Es folgt die Differenzbildung der Ergebnismengen aus (1.).

Die obige Suchanfrage kann somit in zwei SPARQL-Anfragen unterteilt werden (siehe Listing 8.14). Dabei ermittelt die erste (obere) Anfrage alle Übungsaufgaben, welchen das Konzept List zugeordnet wurde. Mit der zweiten (unteren) Anfrage werden alle Übungsaufgaben ermittelt, denen das Konzept Apply zugeordnet wurde. Die entsprechenden Ergebnismengen sind in Listing 8.15 abgebildet, dabei sei L die Menge der ersten Anfrage und A die Menge der zweiten Anfrage.

Listing 8.14: Negation by divide

```
SELECT assignment ?title
WHERE { { ?assignment rdf:type :Assignment .
          ?assignment dc:title ?title }
          ?assignment :hasConcept cs:List
      } ORDER BY ?title

SELECT assignment ?title
WHERE { { ?assignment rdf:type :Assignment .
          ?assignment dc:title ?title }
          ?assignment :hasConcept cs:Apply
      } ORDER BY ?title
```

Listing 8.15: Negation by divide - Ergebnismengen

```
 ?assignment           | ?title
=================================================
 addToEach             | "Add to each"
 applyAll-composeList  | "Apply all - compose list"
 applyList             | "Apply list"
 length-mit-map        | "Length with map"
 mapThree              | "Map three"
 partition             | "Partition"
 quicksort             | "Quicksort"
 searchTree-1          | "SearchTree (1)"

 ?assignment           | ?title
=================================================
 applyAll-composeList  | "Apply all - compose list"
 applyList             | "Apply list"
```

In einem zweiten Schritt kann nun die Differenz der Ergebnismengen L und A auf folgende Weise gebildet werden:

$$L \setminus A = \{x \mid (x \in L) \wedge (x \notin A)\}$$

8.3.8 Beschränkungen von SPARQL

- SPARQL wurde als Anfragesprache für RDF-Graphen entwickelt und unterstützt daher weder die Semantik von RDFS noch die Semantik von OWL.
- Es werden weder verschachtelte noch rekursive Anfragemöglichkeiten unterstützt.
- SPARQL stellt keine Negation bzw. lediglich Negation As Failure zur Verfügung.
- Es werden keine Aggregations-Funktionen, wie beispielsweise Zähler, von SPARQL bereitgestellt.

8.4 Faceted Browsing

Das Konzept des Faceted Browsing wird durch eine Kombination von Und- und Oder-Verknüpfungen realisiert (siehe Abschnitt 5.6). Um die Anfragen mit Faceted Browsing formulieren zu können, wird dem Benutzer im Allgemeinen eine visuelle Schnittstelle angeboten. Mit diesem visuellen Benutzerinterface

ist es auch unerfahrenen Benutzern möglich *„. . . to easily compose queries consisting of ANDs of ORs: selecting a category term is effectively an OR of all of its subcategories, and selecting more than one facet produces an AND across facets“* (vgl. [HEE+02]). Die Wiederholung des Zitates aus Abschnitt 5.6 soll an dieser Stelle die Funktionsweise einer Anfrageformulierung durch Faceted Browsing nochmals verdeutlichen.

Abbildung 8.1: Faceted Browsing

Abbildung 8.1 zeigt ein Benutzerinterface[2], mit welchem Anfragen durch Faceted Browsing erfolgen können. Auf der rechten Seite der Abbildung sind die für dieses Beispiel verfügbaren Facetten zu sehen. Das Startszenario ohne Einschränkung umfasst sämtliche Übungsaufgaben.

Im folgenden Beispiel wird die SPARQL-Anfrage aus Listing 8.5 im Kontext des Faceted Browsing umgesetzt:

„Welchen Übungsaufgaben sind die Konzepte Apply ***oder*** *List zugeordnet?“*

Abbildung 8.2 zeigt auf der rechten Seite die ausgewählten Konzepte der Facette *Concept*. Durch die Auswahl von *Apply* und *List* innerhalb der Facette

2. Dieses Benutzerinterface ist ein TestCase basierend auf der Exhibit-API, siehe `http://simile.mit.edu/exhibit/` (Abruf: 26. September 2008) .

Concept werden diese durch ein **Oder** verknüpft. Die Ergebnismenge in Abbildung 8.2 umfasst 8 Übungsaufgaben und entspricht der Ergebnismenge der SPARQL-Anfrage aus Listing 8.6.

Abbildung 8.2: Faceted Browsing Selected

8.5 Konjunktive Anfragen und Regeln

Wie bereits erwähnt wurde, können *konjunktive Anfragen* bzw. *Reglen* als weiterer grundlegender Formalismus betrachtet werden, um Anfragen in einer Ontologiesprache formulieren zu können. Im Gegensatz zu SPARQL handelt es sich bei konjunktiven Anfragen nicht um eine offiziell spezifizierte Anfragesprache (vgl. [HKRS08]).

Eine konjunktive Anfrage besteht aus einer Konjunktion mehrerer Bedingungen $(B_1, \ldots, B_n)$. Diese Bedingungen sind einfache beschreibungslogische Formeln ohne Operatoren und können eine der folgenden Formen annehmen:

- $C(e)$, wobei C ein Klassenname und e eine Variable oder der Name eines Individuums ist.
- $R(e, f)$, wobei R der Name einer Eigenschaft bzw. eines Properties und e und f jeweils Variablen oder Namen eines Individuums sind.

Die Funktion μ ist Lösung für eine Variablenbelegung einer konjunktiven Anfrage q bezüglich eine Ontologie O, wenn die Anfrage q nach Anwendung der Ersetzung μ aus O prädikatenlogisch folgt: $O \models \mu(q)$.

Ein Beispiel für eine einfache konjunktive Anfrage nach allen Übungsaufgaben, welche den Konzepten Apply und List zugeordnet sind könnte wie folgt aussehen:

$$Assignment(x) \wedge hasConcept(x, Apply) \wedge hasConcept(x, List)$$

Eine konjunktive Anfrage kann als *Regel*[3] (siehe Abschnitt 3.5.1) bzw. Implikation der Form

$$H \leftarrow B_1 \wedge \ldots \wedge B_n$$

ausgedrückt werden (vgl. [HKRS08]). H wird als *head* und die Bedingungen $(B_1, \ldots, B_n)$ werden als *body* der Regel bezeichnet. Das obige Beispiel kann somit wie folgt als Regel dargestellt werden:

$$Solution(x) \leftarrow Assignment(x) \wedge hasConcept(x, Apply) \wedge hasConcept(x, List)$$

Diese Regel soll im Folgenden anhand der *Semantic Web Rules Language (SWRL)* näher betrachtet werden.

Die Semantic Web Rules Language ist ein beim W3C eingereichter Vorschlag für eine Rule Language und *„... combining OWL DL (and thus OWL Lite) with function-free Horn logic, written in Datalog RuleML“* (vgl. [AH08]). Motik et al. zeigen in [MSS04], dass logisches Schließen in OWL DL in Kombination mit SWRL unentscheidbar ist. Die Autoren stellen einen Algorithmus vor, der eine Entscheidbarkeit durch die Einführung von *DL-safe rules* garantiert. Auf weitere Einzelheiten zu SWRL soll in dieser Arbeit nicht weiter eingegangen werden. Detaillierte Informationen zu SWRL sind auf der SWRL-Webseite[4] zu finden.

Listing 8.17 zeigt die obige Regel bzw. die SPARQL-Anfrage aus Listing 8.7 als SWRL-Regel. Die Definition der SWRL-Regel erfolgte in zwei Schritten. Im ersten Schritt wurden die von der Regel benötigten Klassen und die darin verwendeten Variablen definiert. Im zweiten Schritt erfolgte die Definition der Regel selbst.

Zeile 1 des Listings zeigt die Definition der OWL-Klasse *Solution*, welche im *head* der Regel verwendet wird. Die Definition, der in der Regel verwendeten Variablen *x*, zeigt Zeile 3 des Listings. Die Zeilen 5-29 des Listings definieren die eigentlichen Regel. Hierbei entspricht der *head* den Zeilen 6-11 und *body* den Zeilen 13-28 des Listings.

Führt ein Reasoner diese SWRL-Regel aus, dann werden sämtliche Instanzen der Klasse Assignment, welche den Konzepten Apply und List zugeordnet sind, der Klasse Solution zugeordnet. Listing 8.16 zeigt ein Beispiel für eine SPARQL-Anfrage, welche nach sämtlichen Instanzen der Klasse Solution fragt. Die Ergebnismenge dieser SPARQL-Anfrage entspricht der Ergebnismenge aus Listing 8.8.

Listing 8.16: Anfrage nach Anwendung einer SWRL-Regel

```
1 SELECT ?assignment ?title
2 WHERE  { ?assignment rdf:type :Solution .
3          ?assignment dc:title ?title
4        } ORDER BY ?title
```

3. Regeln werden auch *Hornklauseln* oder *definite Logikprogramme* genannt (vgl. [AH08]).
4. `http://www.w3.org/Submission/2004/03/` (Abruf: 26. September 2008)

Listing 8.17: SWRL-Regel

```
<owl:Class rdf:ID="Solution"/>

<swrl:Variable rdf:about="#x"/>

<swrl:Imp rdf:about="#SolutionRule">
   <swrl:head rdf:parseType="Collection">
      <swrl:ClassAtom>
         <swrl:classPredicate rdf:resource="#Solution"/>
         <swrl:argument1 rdf:resource="#x"/>
      </swrl:ClassAtom>
   </swrl:head>

   <swrl:body rdf:parseType="Collection">
      <swrl:ClassAtom>
         <swrl:classPredicate rdf:resource="#Assignment"/>
         <swrl:argument1 rdf:resource="#x"/>
      </swrl:ClassAtom>
      <swrl:IndividualPropertyAtom>
         <swrl:propertyPredicate rdf:resource="#hasConcept"/>
         <swrl:argument1 rdf:resource="#x"/>
         <swrl:argument2 rdf:resource="http://wdok.cs.uni-↘
         →magdeburg.de/ontologies/ComputerScience.owl#Apply"/>
      </swrl:IndividualPropertyAtom>
      <swrl:IndividualPropertyAtom>
         <swrl:propertyPredicate rdf:resource="#hasConcept"/>
         <swrl:argument1 rdf:resource="#x"/>
         <swrl:argument2 rdf:resource="http://wdok.cs.uni-↘
         →magdeburg.de/ontologies/ComputerScience.owl#List"/>
      </swrl:IndividualPropertyAtom>
   </swrl:body>
</swrl:Imp>
```

Teil III

Zusammenfassung und Ausblick

9 Zusammenfassung

Ziel dieser Arbeit war, ein Konzept für ein semantisches Anfragesystem zu entwickeln, welches Suchanfragen auf Basis von Ontologien in einem Bestand von Dokumenten ermöglicht. Im Rahmen dieser Arbeit wurden Ontologien als ein Konzept der Wissensrepräsentation in maschinenverarbeitbarer Form vorgestellt. Auf diesem Wissen aufbauend wurde eine *leicht erweiterbare Ontologiestruktur* entwickelt, welche dem hier entwickelten semantischen Anfragesystem zugrunde liegt. Des Weiteren hatten Überlegungen zu verschiedenen Einsatzmöglichkeiten und notwendigen Anforderungen Einfluss auf die Konzeption der Ontologiestruktur. Auf Grundlage der entwickelten Ontologiestruktur wurde ein Konzept für ein *semantisches Anfragesystem* entworfen, dass einen *modularen* Ansatz wählt und damit die erforderlichen *integrativen* Eigenschaften zur Anbindung an einen Dokumentenbestand bzw. CMS bietet. Dafür wurde ausschließlich auf *Standards der Semantic Web Architektur des W3C* zurückgegriffen, um so eine Erweiterung durch andere semantische Technologien zu ermöglichen.

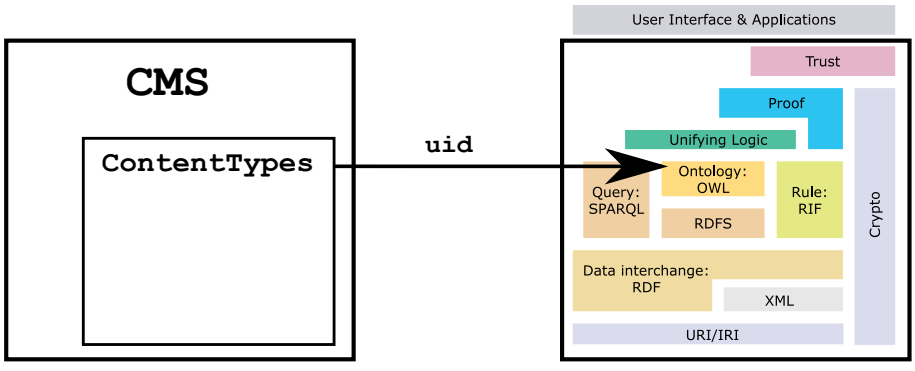


Abbildung 9.1: Architektur von PlOSS^X

Abschließend soll Abbildung 9.1 die Hauptidee (vgl. Abschnitt 7.4.1) des Ansatzes von *PlOSS^X* nochmals veranschaulichen. Hierbei wird über die Eigenschaft *uid* eine eindeutige Abbildung zwischen einer *Instanz eines ContentTypes* des CMS und einer *Instanz der Ontologie* hergestellt. Besondere Beachtung gilt der *offenen Architektur* des Ansatzes, welche aufgrund der Art der

Kopplung prinzipell die Verwendung weiterer semantischer Technologien der Semantic Web Architektur des W3C ermöglicht.

9.1 Einordnung von $PlOSS^X$

In Abschnitt 6.3.1 wurden sieben Merkmale zur Kategorisierung von Ansätzen zur semantischen Suche aufgestellt. Im Folgenden wird das im Rahmen der vorliegenden Arbeit entwickelte semantische Anfragesystem $PlOSS^X$ entlang dieser Kategorisierung diskutiert.

1. **Kopplung:** Das semantische Anfragesystem $PlOSS^X$ erfordert eine *lose Kopplung* zwischen einer Instanz im CMS und einer Instanz in der Ontologie. Eine enge Kopplung beim Ansatz von $PlOSS^X$ ist derzeit nicht denkbar, da hierfür das Content Management System die notwendigen Integrationsmöglichkeiten und semantischen Technologien bereitstellen müsste.

2. **Suchstrategie:** Die Suche kann sowohl in einer *Anfragesprache* als auch durch *Faceted Browsing* erfolgen. Faceted Browsing kann hierbei nur einen Teil der Anfragemöglichkeiten abdecken, welche durch eine Anfragesprache möglich sind.

3. **Transparenz:** Die Anfrageformulierung in einer Anfragesprache kann für den unerfahrenen Benutzer durch Faceted Browsing verborgen werden. Dem versierten Benutzer werden erweiterte Möglichkeiten angeboten, um in einer Anfragesprache Anfragen stellen zu können. Das Anfragesystem kann somit als *hybrid* klassifiziert werden.

4. **Benutzerkontext:** Der Kontext von Benutzern wurde beim Ansatz von $PlOSS^X$ nicht weiter beachtet, dies wird erst bei einer Umsetzung dieses Ansatzes notwendig. Hierbei muss dann über eine Integration in das CMS und somit über den Benutzerkontext entschieden werden.

5. **Anfragemodifikation:** Die Modifikation von Anfragen wird vom Benutzer vorgenommen, dies kann durch eine visuelle oder manuelle Anfrageumformulierung erfolgen. $PlOSS^X$ fällt somit in die Kategorie der *manuellen* Ansätze.

6. **Ontologiesprachen:** Der Fokus von $PlOSS^X$ liegt in der Verwendung von Ontologien im *OWL DL* Format, da diese hinsichtlich logischen Schließens entscheidbar sind.

7. **Inferenzsysteme:** Bei dem Ansatz von $PlOSS^X$ wird der Einsatz eines Reasoners als *notwendig* erachtet, um geschlussfolgertes Wissen in der Ergebnismenge einer Anfrage zu berücksichtigen.

9.2 Abdeckung der Anforderungen

Um die wesentlichen Punkte des im Rahmen dieser Arbeit entwickelten Anfragesystems $PlOSS^X$ abzustecken, wurden in Abschnitt 6.3.2 einige Anforderungen formuliert. Die Abdeckung dieser Anforderungen auf die vorgestellten Anfragekonzepte und das Anfragesystem $PlOSS^X$ werden an dieser Stelle kurz zusammengefasst.

- **Anfragemöglichkeiten:** Das semantische Anfragesystem *PlOSS*^{X} verwendet für Suchanfragen die Anfragesprache SPARQL. SPARQL ist ein zentraler Bestandteil der Semantic Web Architektur des W3C und wird für die Abfrage von in RDF spezifizierten Informationen und für die Darstellung der Resultate verwendet. Der Ansatz von *PlOSS*^{X} zeigt weiterhin, dass eine visuelle Anfrageformulierung, wie z. B. durch das Konzept des Faceted Browsing, durch SPARQL-Anfragen umgesetzt werden kann.

- **Verwendung von Inferenzsystemen:** Die Anforderung nach einem Inferenzsystem wirkt sich bei *PlOSS*^{X} dahingehend aus, dass durch logisches Schließen semantische Suchanfragen an die Wissensbasis bzw. Ontologie ermöglicht werden.

- **Anpassbarkeit und Erweiterbarkeit:** Um eine leichte Anpassbarkeit und Erweiterbarkeit des *PlOSS*^{X}-Ansatzes zu gewährleisten, wurde der Upper Ontology Ansatz gewählt. Mit diesem Ansatz können der CMS-Ontologie weitere Ontologien hinzugefügt werden, wobei diese dann durch das OWL-Äquivalenz-Statement in die CMS-Ontologie integriert und damit verwendet werden können.

- **Integrierbarkeit:** Der Ansatz von *PlOSS*^{X} zeigt, dass die Instanzen des CMS Plone mit den Instanzen einer Ontologie so verbunden werden können, dass auf Basis dieser Ontologie eine semantische Suche erfolgen kann. Der in dieser Arbeit beschriebene Ansatz zeigt weiterhin, wie auch andere Content Management Systeme durch eine semantische Suche erweitert werden können.

Die in Abschnitt 6.3.2 aufgestellten Anforderungen können somit als erfüllt betrachtet werden.

10 Ausblick

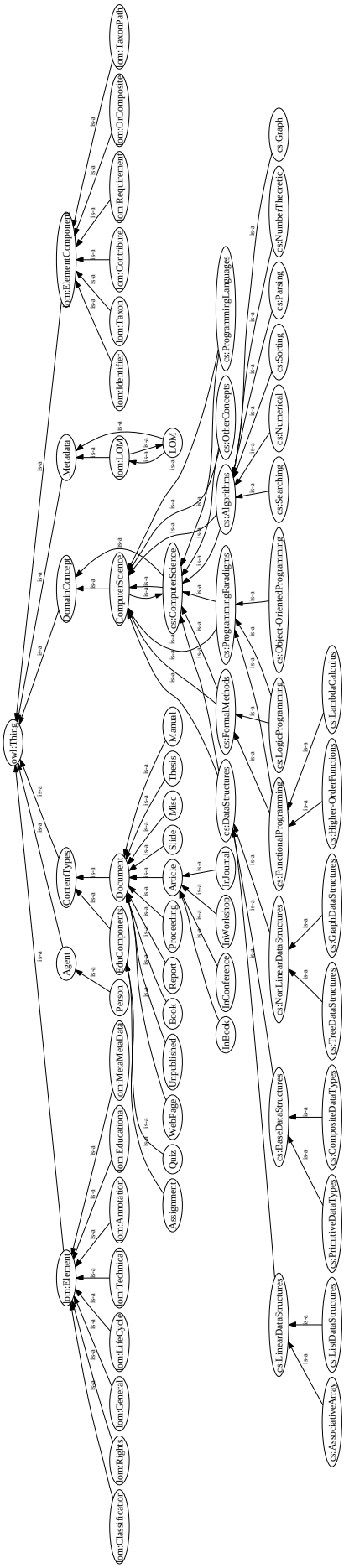
Zum Schluss der vorliegenden Arbeit soll in diesem Kapitel ein Ausblick auf Aspekte gegeben werden, die im Rahmen dieser Arbeit noch nicht zu leisten waren, welche jedoch als weiterführende Aktivitäten notwendig erscheinen.

Der hier vorgestellte Ansatz bildet eine vielversprechende Grundlage für die semantische Suche in einem Bestand von Dokumenten. Hierfür sollte zunächst die Implementierung eines Prototypen erfolgen, um die Evaluierung des $PlOSS^X$-Ansatzes zu ermöglichen. Beispielsweise ist die Realisierung der Aspekte „User Interface & Applications“ (vgl. Abbildung 9.1) noch offen, d. h. ob und wie ein Benutzerinterface in das CMS integriert werden kann.

Weiterhin sind die Technologien der Semantic Web Architektur des W3C Gegenstand intensiver Forschung und Entwicklung. So sollten insbesondere die Aktivitäten der *Rule Interchange Format Working Group* (vgl. Abschnitt 3.5.1) und die Entwicklung der *OWL 2 Web Ontology Language* (vgl. Abschnitt 4.1.1) berücksichtigt sowie gegebenenfalls evaluiert werden.

A Taxonomie der CMS-Ontologie

Abbildung A.1: Komplette Taxonomie der CMS-Ontologie



B Instanzen der Ontologien

Tabelle B.1: Instanzen der CS-Ontologie

Name	Label	Beschreibung	Klassen-Typ
AbstractSyntaxTree	Abstract syntax tree	An abstract syntax tree (AST) is a tree representation of the syntax of some source code.	TreeDataStructures
AdjacencyList	Adjacency list	An adjacency list is the representation of all edges or arcs in a graph as a list.	GraphDataStructures
AdjacencyMatrix	Adjacency matrix	An adjacency matrix is a matrix that shows the relationship between two classes of objects (edges, vertices).	GraphDataStructures
AnonymousFunction	Anonymous function	An anonymous function is a function (or a subroutine) defined, and possibly called, without being bound to a name.	LambdaCalculus
Apply	Apply	Apply is a function that applies functions to arguments.	Higher-OrderFunctions, LambdaCalculus
Array	Array	An array is a data structure consisting of a group of elements that are accessed by indexing.	ListDataStructures
BinarySearch	Binary search	Locates an item in a sorted list.	Searching
BinarySearchTree	Binary search tree	Uses binary tree to maintain elements.	Searching
BinarySearchTrees	Binary search trees	Binary search trees used to construct more abstract data structures such as sets, multisets, and associative arrays.	TreeDataStructures
BinaryTree	Binary tree	A binary tree is a tree data structure in which each node has at most two children.	TreeDataStructures
Breadth-firstSearch	Breadth-first search	Traverses a graph level by level.	Searching
BubbleSort	Bubble sort	For each pair of indices, swap the items if out of order.	Sorting
Character	Character	A character is a unit of information that roughly corresponds to a symbol.	PrimitiveDataTypes
Closures	Closures	A closure is a function that is evaluated in an environment containing one or more bound variables.	FunctionalProgramming
Currying	Currying	Currying is the technique of transforming a function that takes multiple arguments (or more accurately an n-tuple as argument) in such a way as it can be called as a chain of functions each with a single argument.	Higher-OrderFunctions
Depth-firstSearch	Depth-first search	Traverses a graph branch by branch.	Searching
DijkstrasAlgorithm	Dijkstra's algorithm	Computes shortest paths in a graph with non-negative edge weights.	Graph
Double	Double	Double precision floating point is an IEEE 754 standard for encoding floating point numbers that uses 8 bytes.	PrimitiveDataTypes
EuclideanAlgorithm	Euclidean algorithm	Computes the greatest common divisor.	NumberTheoretic
Filter	Filter	Filter is a higher-order function that processes a data structure (typically a list) in some order to produce a new data structure containing exactly those elements of the original data structure for which a given predicate returns the boolean value true.	Higher-OrderFunctions
Float	Float	The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely-used standard for floating-point computation.	PrimitiveDataTypes
Fold	Fold	Fold, also known variously as reduce, accumulate, compress or inject, is a family of higher-order functions that process a data structure in some order and build up a return value.	Higher-OrderFunctions
FunctionComposition	Function composition	Function composition is an act or mechanism to combine simple functions to build more complicated ones.	FunctionalProgramming
Gauss-JordanElimination	Gauss-Jordan elimination	Solves systems of linear equations.	Numerical
HashTable	Hash table	A hash table, or a hash map, is a data structure that associates keys with values.	AssociativeArray
HashTree	Hash tree	A hash tree is a type of data structure which contains a tree of summary information about a larger piece of data used to verify its contents.	TreeDataStructures

Continued on next page

Name	Label	Beschreibung	Klassen-Typ
Haskell	Haskell	Haskell is a standardized purely functional programming language with non-strict semantics, named after the logician Haskell Curry.	ProgrammingLanguages
Heap	Heap	A heap is a specialized tree-based data structure that satisfies the heap property.	TreeDataStructures
InsertionSort	InsertionSort	Determine where the current item belongs in the list of sorted ones, and insert it there.	Sorting
Integer	Integer	An Integer represents some finite subset of the mathematical integers.	PrimitiveDataTypes
IntegerFactorization	Integer factorization	Breaking an integer into its prime factors.	NumberTheoretic
Lambda-Calculus	Lambda calculus	Lambda calculus is a formal system designed to investigate function definition, function application and recursion.	LambdaCalculus
LinearSearch	Linear search	Finds an item in an unsorted list.	Searching
LinkedList	Linked list	A linked list consists of a sequence of nodes, each containing arbitrary data fields and one or two references (links) pointing to the next and/or previous nodes.	ListDataStructures
List	List	A list consists of a sequence of nodes, each containing arbitrary data fields.	ListDataStructures
ListComprehension	List comprehension	A list comprehension is a syntactic construct available in some programming languages for creating a list based on existing lists.	FunctionalProgramming
LLParser	LL parser	A relatively simple linear time parsing algorithm for a limited class of context-free grammars.	Parsing
LRParser	LR parser	A more complex linear time parsing algorithm for a larger class of context-free grammars.	Parsing
Map	Map	The higher-order function map applies a given function to a sequence of elements (such as a list) and returns a sequence of results.	Higher-OrderFunctions
MergeSort	Merge sort	Sort the first and second half of the list separately, then merge the sorted lists.	Sorting
Monad	Monad	A monad is a kind of abstract data type used to represent computations (instead of data in the domain model).	FunctionalProgramming
NewtonsMethod	Newton's method	Finds zeros of functions with calculus.	Numerical
ParseTree	Parse tree	A parse tree is a concrete syntax tree.	TreeDataStructures
Prolog	Prolog	Prolog is a logic programming language.	ProgrammingLanguages
Python	Python	Python is a general-purpose, high-level programming language.	ProgrammingLanguages
Quicksort	Quicksort	Quicksort makes comparisons to sort n items.	Sorting
Recursion	Recursion	Recursion is a method of defining functions in which the function being defined is applied within its own definition.	ProgrammingParadigms
RecursiveDescent	Recursive descent	A top-down parser suitable for LL(k) grammars.	Parsing
Scheme	Scheme	Scheme is a minimalist, multi-paradigm dialect of Lisp.	ProgrammingLanguages
SelectionSort	Selection sort	Pick the smallest of the remaining elements, add it to the end of the sorted list.	Sorting
Self-balancingBSTrees	Self-balancing binary search trees	A self-balancing binary search tree or height-balanced binary search tree is a binary search tree that attempts to keep its height, or the number of levels of nodes beneath the root, as small as possible at all times, automatically.	TreeDataStructures
SieveOfEratosthenes	Sieve of Eratosthenes	Determining whether a given number is prime.	NumberTheoretic
SquareRoot	Square root	Approximates the square root of a number.	Numerical
String	String	A string is an ordered sequence of symbols.	PrimitiveDataTypes
Struct	Struct	Struct is a way and practice to combine simple objects or data types into more complex ones.	CompositeDataTypes
SyntaxTree	Syntax tree	A syntax tree is an (ordered, rooted) tree that represents the syntactic structure of a string according to some formal grammar.	TreeDataStructures
Tuple	Tuple	A tuple is an immutable list. A tuple can not be changed in any way once it is created.	ListDataStructures
Union	Union	Union is a data structure used to hold a value that could take on several different, but fixed types.	CompositeDataTypes

1. An Stelle der UID wurde die URI eines Ploneobjektes, als Wert der Eigenschaft *uid*, verwendet. Die komplette URI setzt sich aus `http://wdok.cs.uni-magdeburg.de/studium-und-lehre/pkm-repository-1/haskell/` + uid-Wert zusammen.

Tabelle B.2: Instanzen der Inhaltstypen der CMS-Ontologie

Name	Label	uid[1]	dc:creator	dc:date	Relation	Konzepte
accumulate	Accumulate	accumulate-haskell	AmelungMario	2006-11-08		cs:Fold, cs:Haskell, cs:Lambda-Calculus
addToEach	Add to each	addtoeach-haskell	AmelungMario	2006-11-01		cs:Haskell, cs:Integer, cs:List
applyAll-composeList	Apply all - compose list	applyAll-composeList-haskell	BluemelIlona	2008-08-07		cs:Apply, cs:Haskell, cs:List, cs:Recursion
applyList	Apply list	applyList	AmelungMario	2006-11-15		cs:Apply, cs:Haskell, cs:Integer, cs:Lambda-Calculus, cs:-List, cs:ListComprehension, cs:Recursion
length-mit-map	Length with map	length-mit-map-haskell	AmelungMario	2007-04-24		cs:Haskell, cs:List, cs:Map
mapThree	Map three	mapThree-haskell	AmelungMario	2006-11-15		cs:Haskell, cs:Lambda-Calculus, cs:List
newton-Verfahren	Newtons method	newton-verfahren-haskell	RoesnerDietmar	2007-07-03		cs:Haskell, cs:NewtonsMethod, cs:SquareRoot
partition	Partition	partition-haskell	AmelungMario	2006-11-22	dcterms:isRequiredBy quicksort	cs:Haskell, cs:List, cs:Tuple
quicksort	Quicksort	quicksort	AmelungMario	2006-11-22	dcterms:requires partition	cs:Haskell, cs:List, cs:Quicksort
searchTree-1	SearchTree (1)	searchtree-1-haskell	AmelungMario	2006-11-29	dcterms:isBasisFor searchTree-2	cs:BinarySearchTrees, cs:Haskell, cs:List
searchTree-2	SearchTree (2)	searchtree-2-haskell	AmelungMario	2006-11-29	dcterms:isBasedOn searchTree-1	cs:BinarySearchTrees, cs:Haskell

C Software zur Arbeit

Unter der URL `http://www-e.uni-magdeburg.de/miotto/diplomarbeit/` sind die folgenden Dateien verfügbar:

1. Die Arbeit in elektronischer Form als PDF-Datei (miotto-diplom.pdf)
2. Die Ontologien als OWL-Dateien
   - CMS-Ontologie (cms.owl)
   - LOM-Ontologie (lom.owl)
   - CS-Ontologie (cs.owl)
3. Die Dateien für das Beispiel des Faceted Browsing als zip-Datei (wdok-exhibit.zip)

Literaturverzeichnis

[AH04] ANTONIOU, Grigoris ; HARMELEN, Frank van ; ANTONIOU, Grigoris (Hrsg.): *A Semantic Web Primer*. The MIT Press, 2004. – ISBN 0–262–01210–3

[AH08] ANTONIOU, Grigoris ; HARMELEN, Frank van ; ANTONIOU, Grigoris (Hrsg.): *A Semantic Web Primer*. second edition. The MIT Press, 2008. – ISBN 978–0–262–01242–3

[APR06] AMELUNG, Mario ; PIOTROWSKI, Michael ; RÖSNER, Dietmar: EduComponents: Experiences in EAssessment in Computer Science Education. In: *ITiCSE '06: Proceedings of the 11th annual conference on Innovation and technology in computer science education* (2006), S. 88–92

[APR07] AMELUNG, Mario ; PIOTROWSKI, Michael ; RÖSNER, Dietmar: Webbasierte Dienste für das E-Assessment. (2007)

[AR05] AMELUNG, Mario ; RÖSNER, Dietmar: XML-Technologie zur Unterstützung der Entwicklung und Wiederverwendung von Lehr- und Lernmaterialien. (2005)

[AS06] ALESSO, H. P. ; SMITH, Craig F. ; ALESSO, H. P. (Hrsg.): *Thinking on the Web: Berners-Lee, Gödel, and Turing*. John Wiley & Sons, Inc., 2006. – ISBN 0–471–76814–6

[BEHV04] BROEKSTRA, Jeen ; EBERHART, Andreas ; HAASE, Peter ; VOLZ, Raphael: *A comparison of RDF query languages*. Proceedings of the Third International Semantic Web Conference, Hiroshima, 2004. `http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/rdfquery.pdf`. Version: 11 2004, Abruf: 26. September 2008

[BKPP07] BOLEY, Harold ; KIFER, Michael ; PATRANJAN, Paula-Lavinia ; POLLERES, Axel: Reasoning Web: Rule Interchange on the Web. In: *Springer-Verlag Berlin Heidelberg* (2007), Nr. 4636, S. 269–309. ISBN 978–3–540–74613–3

[BL97] BERNERS-LEE, Tim: *Metadata Architecture*. http://www.w3.org/DesignIssues/Metadata.html. Version: 1997, Abruf: 26. September 2008

[BLF99] BERNERS-LEE, Tim ; FISCHETTI, Mark: *Der Web-Report*. Econ, 1999. – ISBN 3–430–11468–3

[BLHL01] BERNERS-LEE, Tim ; HENDLER, James ; LASSILA, Ora: The Semantic Web. In: *Scientific American* (2001)

[BR06] BAUMGARTNER, Norbert ; RETSCHITZEGGER, Werner: A SURVEY OF UPPER ONTOLOGIES FOR SITUATION AWARENESS. In: *Proceedings of the International Conference on Knowledge Sharing and Collaborative Engineering (KSCE '06)* (2006). http://www.bioinf.jku.at/publications/ifs/2006/1306.pdf, Abruf: 26. September 2008

[Cap03] CAPLAN, Priscilla: *Metadata Fundamentals for All Librarians*. American Library Association, 2003. – ISBN 0838908470

[DCM08a] DCMI: *DCMI Metadata Terms*. Dublin Core Metadata Initiative. http://dublincore.org/documents/dcmi-terms/. Version: 2008, Abruf: 26. September 2008

[DCM08b] DCMI: *Dublin Core Metadata Element Set, Version 1.1*. Dublin Core Metadata Initiative. http://dublincore.org/documents/dces/. Version: 2008, Abruf: 26. September 2008

[Den03] DENTON, William: How to Make a Faceted Classification and Put It On the Web. (2003). http://www.miskatonic.org/library/facet-web-howto.html, Abruf: 26. September 2008

[DJMZ04] DOSTAL, Wolfgang ; JECKLE, Mario ; MELZER, Ingo ; ZENGLER, Barbara: Semantic Web. In: *Objektspektrum* (2004)

[DSW06] DAVIES, John ; STUDER, Rudi ; WARREN, Paul: *Semantic Web Technologies – Trends and Research in Ontology-based Systems*. John Wiley & Sons Ltd, 2006. – ISBN 0–470–02596–4

[FFR04] FRIESEN, Norm ; FISHER, Sue ; ROBERTS, Anthony: *CanCore Guidelines*. http://www.cancore.ca/en/guidelines.html. Version: 2004, Abruf: 26. September 2008

[FHLW03] FENSEL, Dieter (Hrsg.) ; HENDLER, James A. (Hrsg.) ; LIEBERMAN, Henry (Hrsg.) ; WAHLSTER, Wolfgang (Hrsg.): *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential [outcome of a Dagstuhl seminar]*. MIT Press, 2003 . – ISBN 0–262–06232–1

[GL02] GRUNINGER, M. ; LEE, J.: Ontology - applications and design. In: *Comm.ACM* 45 (2002), Nr. 2, S. 39–41

[GP02] GÓMEZ-PÉREZ, Asunción: *OntoWeb – Deliverable 1.3: A survey on ontology tools*. http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-3.pdf. Version: 2002, Abruf: 26. September 2008

[GPFLC04] GÓMEZ-PÉREZ, Asunción ; FERNÁNDEZ-LÓPEZ, Mariano ; CORCHO, Oscar: *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer-Verlag London, 2004. – ISBN 1–85233–551–3

[Gru93] GRUBER, Thomas R.: A Translation Approach to Portable Ontology Specifications. In: *Academic Press* (1993)

[Ha02] HODGINS, Wayne ; AL., Erik D. ; HODGINS, Wayne (Hrsg.) ; AL., Erik D. (Hrsg.): *Draft Standard for Learning Object Metadata*. Institute of Electrical and Electronics Engineers, Inc., 2002

[HEE+02] HEARST, Marti ; ELLIOTT, Ame ; ENGLISH, Jennifer ; SINHA, Rashmi ; SWEARINGEN, Kirsten ; YEE, Ka-Ping: FINDING THE FLOW IN WEB SITE SEARCH – Designing a search system and interface may best be served (and executed) by scrutinizing usability studies. In: *ACM* 45 (2002), 9, Nr. 9

[HJ02] HOLSAPPLE, Clyde W. ; JOSHI, K.D.: A COLLABORATIVE APPROACH to ONTOLOGY DESIGN. In: *Comm.ACM* 45 (2002), Nr. 2, S. 42–47

[HKRS08] HITZLER, Pascal ; KRÖTZSCH, Markus ; RUDOLPH, Sebastian ; SURE, York: *Semantic Web – Grundlagen*. 1. Springer-Verlag Berlin Heidelberg, 2008. – ISBN 978–3–540–33993–9

[HT06] HOANG, Hanh H. ; TJOA, A M.: The State of the Art of Ontology-based Query Systems: A Comparison of Existing Approaches. In: *IEEE* (2006)

[Joh06] JOHN, Michael: *Semantische Technologien in der betrieblichen Anwendung*. Fraunhofer-Institut für Rechnerarchitektur und Softwaretechnik (FIRST), 2006

[Lew05] LEWANDOWSKI, Dirk: *Web Information Retrieval: Technologien zur Informationssuche im Internet*. Deutsche Gesellschaft für Informationswissenschaft und Informationspraxis e.V., 2005. – ISBN 3–925474–55–2

[Man07a] MANGOLD, Christoph: A survey and classification of semantic search approaches. In: *Int. J. Metadata, Semantics and Ontology* 2 (2007), Nr. 1

[Man07b] MANGOLD, Christoph M.: *Konzepte und Realisierung einer kontextbasierten Intranet-Suchmaschine*, Institut für Parallele und Verteilte Systeme (IPVS), Universität Stuttgart, Diss., 2007

[MCG04] MCGUINNES, Deborah L.: Question Answering on the Semantic Web. In: *IEEE Intelligent Systems* (2004)

[MKA+02] MAGKANARAKI, Aimilia ; KARVOUNARAKIS, Grigoris ; ANH, Ta T. ; CHRISTOPHIDES, Vassilis ; PLEXOUSAKIS, Dimitris: Ontology Storage and Querying / Foundation for Research and Technology Hellas, Institute of Computer Science, Information Systems Laboratory. Version: 4 2002. `http://www.ics.forth.gr/isl/publications/paperlink/tr308.pdf`, Abruf: 26. September 2008. 2002 (Technical Report No 308). – Forschungsbericht

[MSS04] MOTIK, Boris ; SATTLER, Ulrike ; STUDER, Rudi: Query Answering for OWL-DL with Rules. In: *Proceedings of the 3rd International Semantic Web Conference (ISWC 2004)*. Hiroshima, Japan, 11 2004

[NM01] NOY, Natalya F. ; MCGUINNESS, Deborah L.: *Ontology Development 101: A Guide to Creating Your First Ontology*. http://protege.stanford.edu/publications/ontology_development/ontology101.pdf. Version: 2001, Abruf: 26. September 2008

[PRR04] PUGLIA, Steven ; REED, Jeffrey ; RHODES, Erin: *Technical Guidelines for Digitizing Archival Materials for Electronic Access: Creation of Production Master Files – Raster Images*. U.S. National Archives and Records Administration (NARA). http://www.archives.gov/preservation/technical/guidelines.pdf. Version: 2004, Abruf: 26. September 2008

[Sch04] SCHMIDT, Ingrid ; LOBIN, Henning (Hrsg.) ; LEMNITZER, Lothar (Hrsg.): *Texttechnologie - Modellierung von Metadaten*. Stauffenberg Verlag Brigitte Narr GmbH, 2004. – 143–164 S. – ISBN 3–86057–287–3

[SKW07] SUCHANEK, Fabian M. ; KASNECI, Gjergji ; WEIKUM, Gerhard: Yago: A Core of Semantic Knowledge. In: *16th international World Wide Web conference (WWW 2007)*. New York, NY, USA : ACM Press, 2007

[Sta02] STAAB, Steffen: Wissensmanagement mit Ontologien und Metadaten. In: *Informatik Spektrum* 25 (2002), Nr. 3, S. 194–209

[W3C04a] W3C: *RDF Primer*. http://www.w3.org/TR/rdf-primer/. Version: 2004, Abruf: 26. September 2008

[W3C04b] W3C: *RDF Vocabulary Description Language 1.0: RDF Schema*. http://www.w3.org/TR/rdf-schema/. Version: 2004, Abruf: 26. September 2008

[W3C04c] W3C: *Web Ontology Language (OWL)*. http://www.w3.org/2004/OWL/. Version: 2004, Abruf: 26. September 2008

[W3C05a] W3C: *RDF Data Access Use Cases and Requirements*. http://www.w3.org/TR/rdf-dawg-uc/. Version: 3 2005, Abruf: 26. September 2008

[W3C05b] W3C: *Rule Interchange Format Working Group Charter*. http://www.w3.org/2005/rules/wg/charter.html. Version: 2005, Abruf: 26. September 2008

[W3C05c] W3C: World Wide Web Consortium Supports the IETF URI Standard and IRI Proposed Standard. (2005). http://www.w3.org/2004/11/uri-iri-pressrelease.html, Abruf: 26. September 2008

[W3C06] W3C: *Extensible Markup Language (XML) 1.1 (Second Edition)*. http://www.w3.org/TR/xml11. Version: 2006, Abruf: 26. September 2008

[W3C08a] W3C: *OWL 2 Web Ontology Language: Primer*. http://www.w3.org/TR/owl2-primer/. Version: 2008, Abruf: 26. September 2008

[W3C08b] W3C: *SPARQL Protocol for RDF*. http://www.w3.org/TR/rdf-sparql-protocol/. Version: 2008, Abruf: 26. September 2008

[W3C08c] W3C: *SPARQL Query Language for RDF*. http://www.w3.org/TR/rdf-sparql-query/. Version: 2008, Abruf: 26. September 2008

[W3C08d] W3C: *SPARQL Query Results XML Format*. http://www.w3.org/TR/rdf-sparql-XMLres/. Version: 2008, Abruf: 26. September 2008

[W3C08e] W3C: *W3C Opens Data on the Web with SPARQL – Powerful Technology for Querying Distributed and Diverse Data*. http://www.w3.org/2007/12/sparql-pressrelease.html. Version: 2008, Abruf: 26. September 2008

[W3C08f] W3C: *W3C Semantic Web Activity – Latest „layercake“ diagram*. http://www.w3.org/2001/sw/. Version: 5 – 2008, Abruf: 26. September 2008

[Wik08] WIKIPEDIA: *Wikipedia:Categorization*. http://en.wikipedia.org/wiki/Category:Wikipedia_categorization. Version: 2008, Abruf: 26. September 2008

Abschließende Erklärung

Hiermit versichere ich, dass ich die von mir eingereichte Diplomarbeit bzw. die von mir namentlich gekennzeichneten Teile selbständig verfasst habe und ich ausschließlich die angegebenen Hilfsmittel und Quellen benutzt habe.

Magdeburg, 26. September 2008