

---

Monika Lanzenberger

# Diplomarbeit

## RDF Queries

### Variable Abfragen für Datums- und Zeitwerte

ausgeführt am Institut für  
Softwaretechnik und Interaktive Systeme  
Technischen Universität Wien

unter der Anleitung von  
Univ.Ass. Dipl.-Ing. Dr.techn. Monika Lanzenberger

durch

Sabine Eder  
Kölblgasse 5 /23  
1030 Wien  
[sabine\\_eder@gmx.at](mailto:sabine_eder@gmx.at)

Wien, 25. Dezember 2007

---

Sabine Eder

Bei der Erstellung dieses Dokumentes  
wurde  $\LaTeX 2_{\epsilon}$  mit TeXnicCenter verwendet.

## **Zusammenfassung**

Das „Semantic Web“ ist ein junges und dynamisches Forschungsgebiet, welches eine sehr große Interessensgemeinschaft besitzt. Die Zielsetzung des Semantic Web ist, dass Informationen auch von Maschinen gelesen und verarbeitet werden können. Für die Realisierung wird unter anderem RDF verwendet. RDF ist XML sehr ähnlich, aber RDF arbeitet auf einem höheren Abstraktionsniveau. Informationen können durch RDF explizit dargestellt werden. Da die Informationsmenge stetig wächst, werden Query Sprachen benötigt um Informationen effizient abzufragen, ein Beispiel hierfür ist SPARQL.

In RDF werden Datums- und Zeitwerte durch den XSD Datentyp „xsd:DateTime“ abgespeichert. Dieser Datentyp ist so definiert, dass er keinen Spielraum bei einer Abfrage zulässt. Diese Eigenschaft kann zu Einschränkungen bei einer Suchabfrage führen. Bei einer Abfrage auf den Datentyp „xsd:DateTime“ muss der genaue Datums- sowie Zeitwert bekannt sein. Diese Arbeit beschreibt eine Lösung für diese Problemstellung. Es wird eine flexible Suchabfrage für Datums- und Zeitwerte entwickelt. Dadurch ist es möglich, dass z.B. alle Dokumente abgefragt werden können, welche zwischen 13:00 Uhr und 18:00 Uhr an einem Freitag bearbeitet wurden. Dabei werden z.B. die Datumswerte unberücksichtigt gelassen.

Für die Entwicklung dieser variablen Suchabfrage wird das Semantic Web Framework „Jena“ verwendet, welches auch SPARQL Queries unterstützt. Die konkrete Realisierung erfolgt durch die Verwendung von Filter Funktionen. Diese Filter Funktionen können innerhalb einer SPARQL Query aufgerufen werden. Durch die Kombination von unterschiedlichen Filter Funktionen wird ein weites Spektrum, für ungenaue bzw. unpräzise Suchabfragen, abgedeckt. Die Funktionalität dieser Filter Funktionen wird als Java Bibliothek zur Verfügung gestellt. Bei der Entwicklung lag das Augenmerk auf der Portabilität, da diese Bibliothek ohne großen Aufwand in anderen Applikationen verwendbar sein soll.

# Inhaltsverzeichnis

<b>Abstract</b>	<b>iii</b>
<b>1 Einleitung</b>	<b>2</b>
1.1 Motivation . . . . .	2
1.2 Ziele . . . . .	3
1.3 Aufbau der Arbeit . . . . .	4
<b>2 Begriffsdefinition</b>	<b>6</b>
2.1 Semantik . . . . .	6
2.2 Syntax . . . . .	7
2.3 Pragmatik . . . . .	7
2.4 Daten & Information . . . . .	7
2.5 Metadaten . . . . .	8
2.6 Semantic Web . . . . .	8
2.7 Ontologie . . . . .	9
2.8 Taxonomie . . . . .	9
2.9 Web Ontology Language (OWL) . . . . .	10
2.10 Uniform Resource Identifier (URI) . . . . .	10
2.11 Engine . . . . .	11
2.12 Digitale / Elektronische Signatur . . . . .	11
2.13 Aktives Datenbanksystem . . . . .	11
<b>3 Semantic Web</b>	<b>13</b>
3.1 Zielsetzung . . . . .	13
3.2 Architektur . . . . .	14
3.2.1 Schicht 1: Unicode & URI . . . . .	15
3.2.2 Schicht 2: XML + NS + xmlschema . . . . .	16
3.2.3 Schicht 3: RDF + rdfschema . . . . .	17
3.2.4 Schicht 4: Ontology Vocabulary (Ontologien Vokabular) . . . . .	17
3.2.4.1 OWL Full . . . . .	18
3.2.4.2 OWL DL . . . . .	19

3.2.4.3	OWL Lite . . . . .	19
3.2.5	Schicht 5: Logic (Logik) . . . . .	19
3.2.6	Schicht 6: Proof (Beweis) . . . . .	19
3.2.7	Schicht 3 - 6: Digital Signature (Digitale Signatur) . . . . .	20
3.2.8	Schicht 7: Trust (Vertrauen) . . . . .	20
3.2.9	Weiterentwicklung . . . . .	20
3.3	Aktuelle Initiativen und Einsatzfelder . . . . .	20
3.3.1	Semantic Systems / Technologien in der EU . . . . .	21
3.3.2	Semantic Systems / Technologien in Österreich . . . . .	22
3.3.3	Semantic Systems / Technologien bei betrieblichen Anwendungen . . . . .	23
3.3.4	Semantic Web Services . . . . .	24
<b>4</b>	<b>RDF (Resource Description Framework)</b>	<b>26</b>
4.1	RDF Datenmodell . . . . .	26
4.2	RDF Syntax . . . . .	28
4.2.1	RDF Basis Syntax . . . . .	28
4.2.2	Container Elemente . . . . .	31
4.2.3	Aussagen modellieren . . . . .	32
4.3	RDF Schema . . . . .	33
4.3.1	Klassen . . . . .	33
4.3.2	Eigenschaften . . . . .	34
4.4	Query Sprachen für RDF . . . . .	35
4.4.1	RQL (RDF Query Language) . . . . .	37
4.4.2	RDQL (RDF Data Query Language) . . . . .	38
4.4.3	SPARQL (SPARQL Protocol and RDF Query Language) . . . . .	39
4.4.4	OWL-QL (Ontology Web Language Query Language) . . . . .	41
4.5	Verwendung von RDF . . . . .	41
4.5.1	World Wide Web (WWW) & Suchmaschinen . . . . .	41
4.5.2	Semantisch unterstützte Suchmaschinen . . . . .	42
4.5.3	Zentralisierte Beschreibungen spezieller Ressourcen . . . . .	42
4.5.4	Web-Portale . . . . .	42
4.6	Entwicklungen / Anwendungen . . . . .	43
4.6.1	Dublin Core Metadata Initiative (DCMI) . . . . .	43
4.6.1.1	Dublin Core Vokabular . . . . .	43
4.6.1.2	Dublin Core in RDF . . . . .	44
4.6.2	Composite Capability/Preference Profiles (CC/PP) . . . . .	44
<b>5</b>	<b>Jena Framework</b>	<b>46</b>
5.1	Allgemeines . . . . .	46
5.2	RDF . . . . .	48
5.2.1	RDF Graph . . . . .	48

5.2.2	RDF Aussagen . . . . .	49
5.2.3	Navigieren durch ein Modell . . . . .	50
5.2.4	Suchen in einem Modell . . . . .	51
5.2.5	Operationen auf ein Modell . . . . .	52
5.3	SPARQL . . . . .	53
5.4	Ontologie API . . . . .	53
<b>6</b>	<b>SemanticLIFE Project</b>	<b>55</b>
6.1	Überblick . . . . .	55
6.1.1	Beispiele für Personal Information Management (PIM) Projekte	57
6.2	System Architektur . . . . .	58
6.2.1	Erwerb von Daten . . . . .	58
6.2.2	Analyse Modul . . . . .	60
6.2.3	Indexing und Speicherungs-Modul . . . . .	60
6.2.4	Ontologie Repository Standard . . . . .	61
6.2.5	Interaktive Informationsabfrage . . . . .	62
6.2.6	Erweiterbarkeit und Persistenz von Information und Semantik .	63
6.2.7	Client Interaktion . . . . .	64
6.3	Virtuelles Query System (VQS) . . . . .	64
6.3.1	VQS Design . . . . .	64
6.3.1.1	„Meta-data Source“ Modul . . . . .	65
6.3.1.2	„Ontology Repository“ Modul . . . . .	66
6.3.1.3	„Sub Query Formulation“ Modul . . . . .	66
6.3.1.4	„Services“ Modul . . . . .	66
6.3.1.5	„Ontology-based Inference“ Modul . . . . .	67
6.3.1.6	„Virtual Query User Interface“ Modul . . . . .	67
6.3.2	VQS Workflow . . . . .	67
6.4	Virtuelle Query Sprache (VQL) . . . . .	69
6.4.1	VQL Query Aufbau . . . . .	69
6.4.2	VQL Query Typen . . . . .	70
6.4.2.1	VQL Query Typ „data“ . . . . .	70
6.4.2.2	VQL Query Typ „schema“ . . . . .	72
6.4.2.3	VQL Query Typ „iTQL“ . . . . .	73
6.4.3	VQL Query Operatoren & Ausdrücke . . . . .	74
6.4.4	VQL Query Ergebnis Formate . . . . .	75
<b>7</b>	<b>Praxis: Analyse &amp; Design</b>	<b>76</b>
7.1	Vorbereitungen . . . . .	76
7.1.1	Vorüberlegungen und Rahmenbedingungen . . . . .	76
7.2	Anforderungsanalyse . . . . .	78
7.2.1	Anwendungsfälle . . . . .	79

7.3	Konzeption & Software Design . . . . .	81
7.3.1	Erweiterung von ARQ . . . . .	81
7.3.2	Aufbau der Implementierung . . . . .	82
7.3.3	Anwendung . . . . .	83
<b>8</b>	<b>Praxis: Implementierung &amp; Evaluierung</b>	<b>85</b>
8.1	Implementierung . . . . .	85
8.1.1	Beschreibung der Filter Funktionen . . . . .	85
8.1.1.1	getYearValue(xsd:DateTime obj) . . . . .	87
8.1.1.2	getMonthValue(xsd:DateTime obj) . . . . .	87
8.1.1.3	getDayValue(xsd:DateTime obj) . . . . .	87
8.1.1.4	getDateValue(xsd:DateTime obj) . . . . .	88
8.1.1.5	getHoursValue(xsd:DateTime obj) . . . . .	88
8.1.1.6	getMinutesValue(xsd:DateTime obj) . . . . .	88
8.1.1.7	getSecondsValue(xsd:DateTime obj) . . . . .	89
8.1.1.8	getTimeValue(xsd:DateTime obj) . . . . .	89
8.1.1.9	getTimezoneValue(xsd:DateTime obj) . . . . .	89
8.1.1.10	filterEqualTimezone(xsd:DateTime obj, String time- zone . . . . .	89
8.1.1.11	filterUnequalTimezone(xsd:DateTime obj, String ti- mezone) . . . . .	90
8.1.1.12	filterEqualWeekday(xsd:DateTime obj, String week- day) . . . . .	90
8.1.1.13	filterUnequalWeekday(xsd:DateTime obj, String we- ekday) . . . . .	91
8.1.1.14	rangeYear(xsd:DateTime obj, Integer year, Integer year)	91
8.1.1.15	rangeMonth(xsd:DateTime obj, String month, String month) . . . . .	92
8.1.1.16	rangeDay(xsd:DateTime obj, String weekday, String weekday) . . . . .	92
8.1.1.17	rangeHours(xsd:DateTime obj, String hour, String hour, String format) . . . . .	93
8.1.1.18	rangeMinutes(xsd:DateTime obj, Integer minute, In- teger minute) . . . . .	93
8.1.1.19	rangeSeconds(xsd:DateTime obj, Integer second, In- teger second) . . . . .	94
8.1.1.20	rangeDayInWeeks(xsd:DateTime obj, Integer occur- rence, String weekday) . . . . .	94
8.1.1.21	rangeDayInMonths(xsd:DateTime obj, Integer day, String month, String month) . . . . .	95
8.1.2	Beschreibung der Hilfsklassen . . . . .	95

8.1.2.1	Klasse für Datums- & Zeitumwandlungen . . . . .	96
8.1.2.2	Klasse für Konstanten . . . . .	97
8.1.2.3	Klasse für die Zeitzone . . . . .	97
8.1.2.4	Klasse für Validierungen . . . . .	97
8.2	Evaluierung . . . . .	98
8.2.1	Funktionstest . . . . .	98
8.2.2	Test in einer Applikation . . . . .	99
8.2.2.1	Voraussetzungen . . . . .	99
8.2.2.2	Testtool . . . . .	100
8.3	Einsatz . . . . .	105
8.3.1	PIB - Personal Information Box . . . . .	106
<b>9</b>	<b>Abschluss</b>	<b>111</b>
9.1	Zusammenfassung . . . . .	111
9.2	Schlussfolgerung . . . . .	112
9.3	Ausblick . . . . .	113
<b>A</b>	<b>RDF - Klassen &amp; Eigenschaften</b>	<b>115</b>
<b>B</b>	<b>Dublin Core - Metadata Element Set</b>	<b>118</b>
	<b>Danksagung</b>	<b>120</b>
	<b>Abbildungsverzeichnis</b>	<b>121</b>
	<b>Tabellenverzeichnis</b>	<b>121</b>
	<b>Listings</b>	<b>122</b>
	<b>Literaturverzeichnis</b>	<b>124</b>

# Kapitel 1

## Einleitung

Durch dieses Kapitel soll die Motivation für diese Arbeit vermittelt werden. Die Arbeit wird über die Zieldefinition klar abgegrenzt. Es wird auf den Aufbau der Arbeit eingegangen. Dadurch wird ein „kleiner“ inhaltlicher Einblick in die nachfolgenden Kapitel gegeben.

### 1.1 Motivation

Bevor ich meine Motivation zu dieser Arbeit beschreiben kann, muss ich etwas ausschweifen. Während ich mich mit dem Bereich „Semantic Web“ beschäftigte, bin ich um das Teilgebiet RDF<sup>1</sup> und Queries nicht herum gekommen. Grob gesagt wird RDF zum Speichern bzw. für die Darstellung von Informationen verwendet. Wie auch in anderen Bereichen ist das Ziel von Queries, dass Informationen effizient abgefragt werden können. Datums- und Zeitwerte werden in RDF durch einen gemeinsamen Datentyp dargestellt. Auf den ersten Blick ist dies auch sehr praktisch und effizient. Über kurz oder lang hatte ich so meine Probleme mit diesem Datentypen. Während der Arbeit mit dem „xsd:DateTime“ Datentyp stellte ich mir die folgenden Fragen:

*„Wie kann ich bei einem „xsd:DateTime“ Datentyp nur nach einem Datumswert suchen?“*

*„Wie kann ich bei einem „xsd:DateTime“ Datentyp nur nach einem Zeitwert suchen?“*

Hier dachte ich mir, dass sich sicherlich auch andere diese Frage stellen. Dies war das auslösende Ereignis meiner Motivation zu dieser Arbeit. Sehr selten kommt es vor, dass man genau weiss, wann man ein Dokument erstellt hat. Mir fällt kein Dokument oder ähnliches ein, wo ich folgende Aussage treffen kann: „Ich habe dieses Dokument am 01.12.2007 um 17:13:47 Uhr erstellt.“ So exakt müssen die Angaben für den Datums-

---

1. Resource Description Framework

und Zeitwerte Datentyp aber sein. Die Suchabfrage könnte man auch auf einen Bereich einschränken. Ein Beispiel für eine solche Abfrage ist: „Gib alle Dokumente aus, welche zwischen 01.01.2007 um 00:00:00 Uhr und zwischen 31.12.2007 00:00:00 Uhr erstellt wurden“. Mit dieser Abfrage ist es nicht möglich, dass alle Dokumente ausgegeben werden, welche auf einem Montag erstellt wurden. Daher sind Suchabfragen mit Bereicheinschränkungen alleine nicht ausreichend. Da der Mensch oft nur eine fragmentarische Erinnerung hat, ist die Ausgangssituationen für eine solche Suchabfrage selten so genau definiert. Einige Beispiele für realistische Ausgangssituationen sind folgende:

- Ich weiss, dass ich das Dokument zwischen 13:00 Uhr und 15:00 Uhr erstellt habe, weil ich es immer vor der Besprechung am Montag erstelle.
- Bei einer Besprechung wurden verschiedene Themen festgelegt. Ich weiss aber nur mehr, dass diese Besprechung an einem Mittwoch statt gefunden hat.
- Wenn ich nach Unterlagen zu einer bestimmten Lehrveranstaltung suche, stehe ich oft vor dem Problem, dass ich nicht mehr genau weiss, in welchem Jahr ich diese Lehrveranstaltung absolviert habe. Ich weiss aber noch, dass die Lehrveranstaltung immer an jedem 2. Donnerstag statt gefunden hat. Des Weiteren kann ich den Jahreszeitraum noch eingrenzen.

Je mehr ich mich mit der beschriebenen Problematik beschäftige, umso mehr stieg mein Interesse daran. Durch Khalid Latif und Shuaib Karim ist die Grundidee zu solchen variablen Datums- und Zeitabfragen entstanden. Diese Suchabfragen sind auch für das „SemanticLIFE Project“<sup>2</sup> von Vorteil, welches ein akuelles Forschungsprojekt ist.

## 1.2 Ziele

Durch die zuvor beschriebene Motivation ergibt sich ein primäres Ziel für diese Arbeit. Dies lässt sich wie folgt beschreiben:

*„Es sollen variable Suchabfragen für Datums- und Zeitwerte entwickelt werden.“*

Bevor das Ziel in Angriff genommen werden kann, muss ausreichend Hintergrundwissen erarbeitet werden. Dieses Hintergrundwissen stellt die Basis dar, damit das primäre Ziel erreicht werden kann. Hierzu gehört eine Beschreibung des Semantic Web und RDF, sowie eine Beschreibung von Queries. Um zu verdeutlichen, dass dieses Ziel nicht einfach aus der Luft gegriffen ist, werden auch aktuelle Initiativen und Applikationen behandelt. Das „SemanticLIFE Project“ wird in diesem Zusammenhang detaillierter beschrieben.

---

2. Siehe dazu auch <http://storm.ifs.tuwien.ac.at/>- Zugriffsdatum: 18.12.2007

Um zu erreichen, dass Datums- und Zeitabfragen innerhalb einer Query formuliert werden können, wird das Framework<sup>3</sup> Jena beschrieben. Durch dieses Framework werden verschiedene Funktionen für Applikationen zur Verfügung gestellt. Viele aktuelle Anwendungen verwenden das „Jena Framework“<sup>4</sup>, wofür es verschiedene Gründe gibt. Hierauf wird in den jeweiligen Kapiteln genauer eingegangen.

Für eine Suchabfrage müssen Datums- und Zeitwerte exakt definiert werden. Dies hängt mit der Definition des Datentyps „xsd:DateTime“ zusammen. Im Zuge dieser Arbeit wird dieser Datentyp beschrieben, wodurch verdeutlicht wird, wieso die entwickelten Erweiterungen sinnvoll sind. Für die Realisierung der flexiblen Datums- und Zeitabfragen werden unter anderem Filter Funktionen innerhalb des Frameworks Jena verwendet.

### 1.3 Aufbau der Arbeit

Grob lässt sich diese Arbeit in die drei Bereiche „Theorie“ und „Praxis“ sowie „Abschluss“ aufteilen. Zum ersten Bereich „Theorie“ zählen die Kapitel 2, 3, 4 und 5 sowie 6. Die darauf folgenden Kapitel 7 und 8 beschreiben, wie die Abfragen von flexiblen Datums- und Zeitwerten realisiert werden. Nachfolgend werden die Kapitel dieser Arbeit kurz vorgestellt.

#### **Kapitel 2 - Begriffsdefinition**

Dieses Kapitel ist die Basis der darauf folgenden Kapitel. Es werden hier die relevanten Begriffe wie z.B. „Semantik“, „Ontologie“ oder „Engine“ behandelt.

#### **Kapitel 3 - Semantic Web**

Über das Kapitel 3 wird ein Teil des Hintergrundwissens für die Praxis Kapitel erarbeitet. In diesem Zusammenhang werden die Ziele, sowie die Architektur des Semantic Web behandelt. Des Weiteren werden die aktuellen Initiativen auf diesem Gebiet beschrieben. Es wird unter anderem auf „Semantic Systems“ und „Semantic Web Services“ eingegangen.

#### **Kapitel 4 - RDF (Resource Description Framework)**

Im Kapitel 4 wird das Resource Description Framework beschrieben. Diese Thematik ist eine Grundlage für die Kapitel zur Implementierung. Im speziellen wird hier auf den Aufbau des Frameworks sowie auf die Abfragesprachen wie z.B. SPARQL oder RQL eingegangen. Um ein besseres Verständnis für diesen Themenbereich zu erlangen, wird die Verwendung, wie z.B. im World Wide Web (WWW) oder bei Web-Portalen, beschrieben. Entwicklungen bzw. Anwendungen, wie Dublin Core Metadata Initiative, werden hier auch beschrieben.

---

3. eine Grundstruktur bzw. ein Rahmenwerk

4. Im Folgenden weiter Jena genannt

**Kapitel 5 - Jena Framework**

Um Applikationen im Bereich „Semantic Web“ zu entwerfen und zu entwickeln, existieren einige Frameworks. Über ein Framework werden unterschiedliche Methoden und Funktionen zur Verfügung gestellt. In diesem Zusammenhang wird das Framework Jena behandelt. In diesem Kapitel fließen bereits Bereiche der zuvor behandelten Kapitel ein. Unter anderem wird auf Jena in Verbindung mit RDF und SPARQL eingegangen. Die Behandlung dieses Frameworks ist grundlegend für die Kapitel über die Implementierung.

**Kapitel 6 - SemanticLIFE Project**

Der Bereich „Semantic Web“ ist ein aktives Forschungsgebiet, in welchen viele Applikationen entwickelt werden. Das Kapitel 6 befasst sich mit einem aktuellen Forschungsprojekt auf diesem Gebiet, dem „SemanticLIFE Project“. Beim „SemanticLIFE Project“ wird versucht sich der Idee von Memex zu nähern, bei welchem eine individuelle Speicherung von Dokumenten möglich ist. Es wird die Architektur der Applikation beschrieben und es wird speziell auf die Datenabfrage eingegangen.

**Kapitel 7 & 8 - Praxis**

Die Kapitel 7 und 8 beschreiben wie das Ziel der flexiblen Datums- und Zeitabfragen erreicht wird. Diese Kapitel bauen auf das zuvor erarbeitete Hintergrundwissen auf. Im Kapitel 7 werden die Vorbereitungen für die Implementierung der Funktionalität beschrieben. Des Weiteren wird durch eine Anforderungsanalyse der Bedarf dieser Abfragen dargestellt. Der abschließende Teil dieses Kapitels widmet sich dem Software Design. Die Implementierung selbst, sowie die genaue Funktionalität, wird im Kapitel 8 beschrieben. Des Weiteren befasst sich dieses Kapitel mit der Evaluierung der entwickelten Suchabfragen sowie mit den Tests. Es wurden unterschiedliche Tests durchgeführt um die flexiblen Suchabfragen auf ihre Funktionalität zu testen. Ein wichtiger Aspekt bei der Entwicklung bestand darin, dass die Funktionalität mit geringem Aufwand in ein andere Anwendung eingebunden werden kann. Dies wird getestet durch die Verwendung der variablen Datums- und Zeitabfragen innerhalb der Applikation „Personal Information Box“.

**Kapitel 9 - Abschluss**

Das Kapitel 9 beinhaltet eine Zusammenfassung der zuvor behandelten Kapitel. Es werden die Schlussfolgerungen, welche aus dieser Arbeit resultieren, beschrieben. Im Ausblick wird auf mögliche Erweiterungen im Zusammenhang mit ungenauen bzw. unpräzisen Suchabfragen eingegangen.

## Kapitel 2

### Begriffsdefinition

In diesem Kapitel werden verschiedene Begriffe beschrieben, welche bei den nachfolgenden Kapitel benötigt werden.

#### 2.1 Semantik

*„ [Die Semantik ist] die Lehre von der inhaltlichen Bedeutung einer Sprache und ihrer Zeichen. In der Informatik bezeichnet die Semantik spezieller die eindeutige Beziehung zwischen einer (in einer Programmiersprache formulierten) Anweisung und der elementaren Funktion oder Zustandsänderung, die der Computer daraufhin ausführen soll. Jede Programmiersprache benötigt eine exakt definierte Semantik.“* (Brockhaus, 2002, S. 800)

Es gibt natürlich auch noch eine andere Definition für „Semantik“ im sprachwissenschaftlichen Sinn. Mit dem Semantic Web sollen nicht einfach nur Daten übertragen werden wie beim Internet, sondern es soll auch die Bedeutung von Daten vermittelt werden. Der Begriff „Semantic Web“ ist nach Pellegrini und Blumauer (2006)<sup>1</sup> nicht weit genug gefasst, da zur Zeit die drei Aspekte Semantik, Syntax und Pragmatik in diesem Bezug berücksichtigt bzw. diskutiert werden.

Ein Begriff, welcher im Zusammenhang mit dem Semantic Web oft vorkommt ist „semantische Qualität“. Dieser ist wie folgt definiert:

*„ Although the primary goal for semantic quality is a correspondence between the externalized ontology and the domain, this correspondence can neither be established nor checked directly.“* (Lanzenberger, Sampson, Rester, Naudet und Latour, 2007, S. 9)

---

1. Dieses Werk ist sehr empfehlenswert für EinsteigerInnen auf dem Gebiet des Semantic Web, da man sich dadurch einen sehr guten Überblick über diese Thematik verschaffen kann. Die Gliederung dieses Werks trägt auch sehr zur Verständlichkeit bei.

Sinngemäß bedeutet dies, dass obwohl die primäre Absicht für die semantische Qualität eine Ähnlichkeit zwischen der Ontologie und der Domäne ist, kann diese Ähnlichkeit weder gegründet noch direkt überprüft werden.

## 2.2 Syntax

*„ [Die Syntax beschreibt die] Art und Weise sprachliche Elemente zu Sätzen zu ordnen.“ (Brockhaus, 1990, S. 761)*

Die Syntax ist unter anderem ein Teilgebiet der Grammatik. In der Literatur zum Semantic Web kommt oft der Begriff „syntaktische Qualität“ vor. Dieser Begriff ist wie folgt definiert:

*„ Syntactic quality is the degree of correspondence between the ontology and the language extension.“ (Lanzenberger u. a., 2007, S. 8)*

Sinngemäß bedeutet dies, dass die syntaktische Qualität der Grad der Ähnlichkeit zwischen der Ontologie und einem bestimmten Vokabular ist.

## 2.3 Pragmatik

*„ Das Sprachverhalten, das Verhältnis zwischen sprachlichen Zeichen und interpretierendem Menschen untersuchende linguistische Disziplin.“ (Brockhaus, 1990, S. 627)*

Diese Definition für „Pragmatik“ stammt aus der Sprachwissenschaft. Im Zusammenhang mit dem Semantic Web kommt oft auch der Begriff „pragmatische Qualität“ vor, welcher wie folgt definiert ist:

*„ Pragmatic quality is the degree of correspondence between the ontology and the audience interpretation.“ (Lanzenberger u. a., 2007, S. 9)*

Sinngemäß bedeutet dies, dass die pragmatische Qualität der Grad der Ähnlichkeit zwischen der Ontologie und der Interpretation der AnwenderInnen und DomänenexpertInnen ist.

## 2.4 Daten & Information

Die zwei Begriffe „Daten“ und „Information“ hängen eng miteinander zusammen. Daher wird zu erst der Begriff „Daten“ definiert und anschließend der Begriff „Information“.

*„ Zeichen oder kontinuierliche Funktionen, die zum Zweck der Verarbeitung aufgrund von bekannten oder unterstellten Vereinbarungen „Information“ darstellen. Daten in diesem Sinn werden nach ihrer äußeren Form unterteilt in Zeichen (Buchstaben, Ziffern, Sonderzeichen), Bilder, Texte, Sprache und Muster.“ (Brockhaus, 2002, S. 212)*

Man unterscheidet hierbei noch nach der Lesbarkeit von Daten. Daten können von Maschinen, von Menschen oder von beiden gelesen werden. Der Begriff „Daten“ hat auch noch weitere Definitionen, je nachdem in welchem Zusammenhang der Begriff verwendet wird. Wenn dieser Begriff z.B. in einem Gesetzestext verwendet wird, dann hat dieser dort eine andere Bedeutung. (Brockhaus, 2002)

*„ Die zweckbestimmte Interpretation von Daten durch den Menschen. Daten bestehen zunächst nur aus Fakten und werden erst dann zu Information, wenn sie im Kontext betrachtet werden und eine Bedeutung für den Menschen übermitteln.“ (Microsoft, 2000, S. 362)*

Bei der Bewertung von Information wird zwischen dem syntaktischen, dem semantischen und pragmatischen Aspekt unterschieden. Beim syntaktischen Aspekt geht es um die strukturelle Eigenschaft der Information. Der Bedeutungsgehalt einer Information wird durch den semantischen Aspekt behandelt. Der Wert einer Information wird durch den pragmatischen Aspekt beschrieben. (Brockhaus, 2002)

## 2.5 Metadaten

*„ Bezeichnung für Daten, die andere Daten beschreiben, z.B. die Datenbankbeschreibung in einer Datenbank (beschreibt die Bedeutung und Struktur der Primärdatensätze), eine Indextabelle (gibt die Speicheradresse von Daten an) oder eine Metadatei. “ (Brockhaus, 2002, S. 574)*

Eine der Voraussetzungen für die Realisierung des Semantic Web ist eine Metadaten - Infrastruktur, welche man mit XML<sup>2</sup> alleine nicht mehr zu Stande bringen kann. Wie Metadaten im Semantic Web mit RDF dargestellt werden, wird in dieser Arbeit im Kapitel 4 noch beschrieben. (Pellegrini und Blumauer, 2006)

## 2.6 Semantic Web

*„ The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and commu-*

---

2. eXtensible Markup Language

*nity boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners.“*  
(Herman, 1994)

Sinngemäß bedeutet dies, dass durch das Semantic Web ein allgemeines Framework zur Verfügung gestellt wird, durch welches Daten über Anwendungs- und Unternehmensgrenzen hinaus sowie mit verschiedenen Communities gemeinsam genutzt und wiederverwendet werden können. Diese Bemühung wird geleitet von W3C<sup>3</sup> mit der Beteiligung von vielen ForscherInnen und KooperationspartnerInnen aus der Industrie.

## 2.7 Ontologie

*„ An ontology is an explicit specification of a conceptualization.“*  
(Gruber, 1993, S. 1)

Dies bedeutet sinngemäss, dass eine Ontologie eine explizite und formale Konzeptualisierung einer Domäne ist. Durch eine Ontologie wird eine Domäne beschrieben. Eine weitere Definition für Ontologie ist folgende:

*„ Ontologien wurden im Umfeld der Künstlichen Intelligenz entwickelt und sind die zentralen Bausteine des Semantic Web: Mit ihnen kann Wissen einer Domäne formal repräsentiert und prinzipiell unabhängig von Programmen wieder verwendet werden.“* (Pellegrini und Blumauer, 2006, S. 12)

Im Semantic Web werden dadurch Konzepte und ihre Beziehungen innerhalb einer Wissensdomäne beschrieben, wodurch Maschinen unterstützt werden, Inhalte im World Wide Web (WWW) zu interpretieren anstatt die Inhalte „nur“ darzustellen. (Pellegrini und Blumauer, 2006)

## 2.8 Taxonomie

*„ Properly structured taxonomies help bring substantial order to elements of a model, are particularly useful in presenting limited views of a model for human interpretation, and play a critical role in reuse and integration tasks. Improperly structured taxonomies have the opposite effect, making models confusing and difficult to reuse or integrate.“* (Guarino und Welty, 2000, S. 1)

---

3. World Wide Web Consortium

Sinngemäß bedeutet dies, dass genau strukturierte Taxonomien dabei helfen eine Ordnung in die Elemente eines Modells zu bringen. Sehr hilfreich sind Taxonomien bei der Darstellung von Teilansichten eines Modells für die Interpretation durch eine Person. Bei der Wiederverwendung und der Integration von Aufgaben spielen Taxonomien ebenfalls eine Rolle. Im Gegensatz dazu haben unstrukturierte Taxonomien den Effekt, dass ein Modell verwirrend wird und dass dieses dadurch schwer wiederverwendbar oder integrierbar ist. (Guarino und Welty, 2000)

Es gibt auch weitere Definitionen für „Taxonomie“. Bei Brockhaus (1990) ist dieser Begriff wie folgt definiert.

*„ Teilgebiet der Linguistik, auf dem man durch Segmentierung und Klassifikation sprachlicher Einheiten den Aufbau eines Sprachsystems beschreiben will (Sprachwissenschaft)“* (Brockhaus, 1990, S. 768)

## 2.9 Web Ontology Language (OWL)

*„ The OWL Web Ontology Language is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications.“* (Smith, Welty und McGuinness, 2004)

Sinngemäß bedeutet dies, dass mit OWL eine Sprache zur Verfügung gestellt wird, durch welche Klassen und deren Relationen zueinander beschrieben werden können. Dadurch ist es auch möglich, dass Aussagen über Klassen gemacht werden können. Diese kann sowohl in Dokumenten als auch in Applikationen eingesetzt werden. OWL ist eine Wissenrepräsentation, welche auch Schlußfolgerungsmechanismen unterstützt. (Smith u. a., 2004)

## 2.10 Uniform Resource Identifier (URI)

*„ A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource. “* (Berners-Lee, 2005, S. 1)

Sinngemäß bedeutet dies, dass eine Uniform Resource Identifier (URI) eine Zeichenfolge ist, durch welche eine abstrakte oder physikalische Ressource identifiziert werden kann. Der Zugriff auf eine Ressource muss nicht unbedingt über das Internet erfolgen. Eine Ressource kann alles mögliche sein wie z.B. ein Buch in einer Bibliothek. (Berners-Lee, 2005)

## 2.11 Engine

*„Zentraler Teil eines Programms für grundlegende Teilaufgaben (z.B. Grafik-Engine zur Ausgabe von Grafikdaten). Manchmal auch verkürzt für „Search Engine“ (Suchmaschine) gebraucht.“* (Brockhaus, 2002, S. 315)

Eine Engine wird auch als Prozessor oder als ein Programmteil bezeichnet, welcher für die Manipulierung sowie für die Verwaltung von Daten maßgebend ist. (Microsoft, 2000)

## 2.12 Digitale / Elektronische Signatur

*„elektronische Signatur: elektronische Daten, die anderen elektronischen Daten beigefügt oder mit diesen logisch verknüpft werden und die der Authentifizierung, also der Feststellung der Identität des Signators, dienen“* (Signaturgesetz, § 2, 2000)

Nach dem Signaturgesetz, § 2 (2000) wird eine sichere elektronische Signatur wie folgt definiert.

*„eine elektronische Signatur, die*

- a) ausschließlich dem Signator zugeordnet ist,*
- b) die Identifizierung des Signators ermöglicht,*
- c) mit Mitteln erstellt wird, die der Signator unter seiner alleinigen Kontrolle halten kann,*
- d) mit den Daten, auf die sie sich bezieht, so verknüpft ist, daß jede nachträgliche Veränderung der Daten festgestellt werden kann, so wie*
- e) auf einem qualifizierten Zertifikat beruht und unter Verwendung von technischen Komponenten und Verfahren, die den Sicherheitsanforderungen dieses Bundesgesetzes und der auf seiner Grundlage ergangenen Verordnungen entsprechen, erstellt wird“* (Signaturgesetz, § 2, 2000)

## 2.13 Aktives Datenbanksystem

*„Ein Datenbanksystem heißt aktiv, wenn es zusätzlich zu den üblichen DBS<sup>4</sup>-Fähigkeiten in der Lage ist, definierbare Situationen in der Datenbank (und wenn möglich darüber hinaus) zu erkennen und*

---

4. Datenbanksystem

*als Folge davon bestimmte (ebenfalls definierbare) Reaktionen auszulösen.* “ (Dittrich und Gatzju, 2000, S. 7)

Ein aktives Datenbanksystem muss alle Eigenschaften eines passiven Datenbanksystems haben, wie ein Datenmodell und ein Transaktionsmodell, die Möglichkeit zur Datenabfrage und zur Datenmanipulation, Datensicherung, Wiederanlauf und Datenschutz. Die zusätzlichen Funktionen eines passiven Datenbanksystems besitzt ein aktives Datenbanksystem eine Beschreibung von Situationen und Reaktionen. Eine Situation ist definierbar und muss erkannt werden. Die gewünschte Reaktion auf eine Situation ist definierbar und automatisch ausführbar. (Dittrich und Gatzju, 2000)

# Kapitel 3

## Semantic Web

Das Semantic Web ist ein aktives Forschungsgebiet. Daher werden viele unterschiedliche Applikationen in diesem Bereich entwickelt. Um das Prinzip des Semantic Web zu verstehen ist es notwendig, dass die grundlegenden Ideen sowie die Zielsetzungen beschrieben werden. Daher befasst sich der erste Teil dieses Kapitels mit dieser Thematik.

Grundlegend für das Verständnis des Semantic Web ist auch die Architektur, welche aus sieben Schichten besteht. Aus diesem Grund wird im zweiten Abschnitt auf diese Architektur eingegangen. In dieser Arbeit wird speziell die Schicht „RDF“ behandelt.

Der letzte Abschnitt dieses Kapitels widmet sich den aktuellen Initiativen und Einsatzfeldern in der Europäischen Union<sup>1</sup>, Österreich und bei betrieblichen Anwendungen.

### 3.1 Zielsetzung

Im World Wide Web (WWW) ist das Auffinden von Wissen mittlerweile schwierig geworden, da die Menge an Information stetig zunimmt. Um Informationen im Internet zu finden, werden Suchmaschinen, wie z.B. Google<sup>2</sup>, Yahoo<sup>3</sup>, AltaVista<sup>4</sup> usw., verwendet. All die verschiedenen Suchmaschinen besitzen die gleichen Probleme. Harmelen und Antoniou (2004) beschreiben diese Probleme wie folgt:

- Selbst wenn hauptsächlich relevante Seiten gefunden und angezeigt werden, sind diese nicht immer brauchbar. Vor allem nicht, wenn auch die wenig brauchbaren sowie die nicht brauchbaren Seiten angezeigt werden. Wenn die Anzahl der gefundenen Seiten zu hoch ist, dann kann dies ebenso von Nachteil sein, als wenn zu wenige Seiten gefunden werden.

---

1. Im Folgenden weiter EU genannt

2. Siehe dazu auch <http://www.google.com/> - Zugriffsdatum: 15.12.2007

3. Siehe dazu auch <http://www.yahoo.com/> - Zugriffsdatum: 15.12.2007

4. Siehe dazu auch <http://www.altavista.com/> - Zugriffsdatum: 15.12.2007

- Es kann vorkommen, dass man auf eine Suchanfrage kein Ergebnis bekommt. Des Weiteren kann es sein, dass nicht alle wichtigen und relevanten Seiten gefunden und angezeigt werden. Dieses Problem, kommt bei Suchmaschinen eher selten vor.
- Die Resultate sind abhängig vom Vokabular. Oft kommt es vor, dass man mit den initialen Suchwörtern nicht das gewünschte Ergebnis erhält. In diesem Fall verwenden die relevanten Dokumente eine unterschiedliche Terminologie von der ursprünglichen Frage. Hier sollten semantisch ähnliche Fragen auch ähnliche Resultate liefern.
- Resulte können auch einzelne Webseiten sein. Wenn Informationen benötigt werden, welche auf mehrere Dokumente verteilt sind, so ist es notwendig, dass mehrere Suchanfragen gestellt werden. Anschließend muss man alle Dokumente sammeln, manuell extrahieren und sie anschließend wieder zusammenfügen.

Durch das Semantic Web werden verschiedene Ansätze miteinander kombiniert. Ein Ansatz stammt aus dem Bereich der künstlichen Intelligenz, welcher sich damit beschäftigt, dass Maschinen auf semantischer Ebene arbeiten. Die Extraktion von Semantik aus Datenbeständen, welche nicht oder nur teilweise strukturiert sind, ist ein weiterer Ansatz. Dieser geht ebenfalls auf den Bereich der künstlichen Intelligenz zurück. Das Semantic Web geht auf die Verteiltheit von Information in großen Mengen ein, wie es auch beim World Wide Web (WWW) der Fall ist. (Tochtermann und Maurer, 2006)

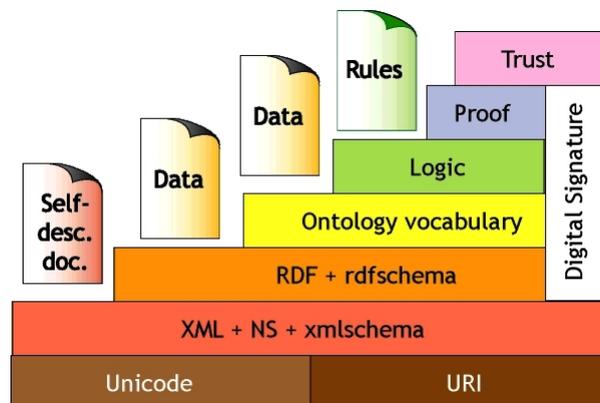
Die Zielsetzung des Semantic Web ist, dass Informationen auch von Maschinen gelesen und verarbeitet werden können. Es wird die Datenintegration und die Kombination von Daten unterstützt. Durch intelligente Agenten besteht auch die Möglichkeit, dass Aufgaben automatisiert werden können. Die Grundidee des Semantic Web ist, dass standardisierte Technologien definiert und verwendet werden mit welchen eine maschinenverarbeitbare Beschreibung von Information im World Wide Web (WWW) möglich ist. Die Schlüsseltechnologien beim Semantic Web beinhalten das Konzept der Metadaten, der Logik und Schlussfolgerungen, der Ontologien und der intelligenten Agenten. (Harmelen und Antoniou, 2004; Herman, 1994)

Das Semantic Web geht von dem Prinzip aus, dass jede Information mit einer URI identifiziert werden kann. Ein weiteres Prinzip ist, dass unvollständige Informationen akzeptabel sind und dass absolute Wahrheiten nicht benötigt werden. (Jacobs, 2004; Harmelen und Antoniou, 2004)

## 3.2 Architektur

Für einen besseren Überblick über diese Thematik wird an dieser Stelle die Architektur des Semantic Web, welche aus sieben Schichten besteht, behandelt. Abbildung 3.1 zeigt

diese Architektur, welche durch die entsprechenden Unterabschnitte beschrieben wird.



**Abbildung 3.1:** Architektur des Semantic Web (Quelle: Berners-Lee, 2000, W3C)

### 3.2.1 Schicht 1: Unicode & URI

Auf der untersten Schicht der Architektur des Semantic Web finden sich die Bereiche „Unicode“ und „URI“. Diese Schicht bildet die Grundlage für die nachfolgenden Schichten.

Zeichen aus verschiedenen Kulturen weltweit werden durch Unicode erfasst. Unicode ist ein internationaler Standard, welcher eine sehr umfangreiche Datenbank für Zeichen besitzt. In dieser Datenbank sind unter anderem Zeichen aus dem japanischen, chinesischen, griechischen, skandinavischen, kyrillischen, usw. vorhanden. Dieser Standard unterliegt einer fortlaufenden Erweiterung, was an den Versionen ersichtlich ist. Dadurch ist eine einheitliche Kodierung „aller“ Zeichen möglich. Auf Grund dessen ist es möglich, dass Maschinen die verschiedensten Zeichen lesen können. Unicode verwendet nicht nur 8 Bit pro Zeichen, wie es beim ASCII<sup>5</sup> - bzw. ANSI<sup>6</sup> - Zeichensatz verwendet wird, sondern es werden 16 Bit pro Zeichen verwendet. Die Anzahl der Programme und Anwendungen, welche diesen Standard unterstützen, nimmt immer mehr zu. (DATACOM, 2007; Brockhaus, 2002)

Durch URIs ist es möglich weltweit eindeutige Namen zu bilden. URIs besitzen im Wesentlichen die folgenden drei Unterarten:

1. Uniform Resource Locators (URL)
2. Uniform Resource Names (URN)
3. Uniform Resource Characteristic (URC)

5. American Standard Code for Information Interchange

6. American National Standards Institute

URLs identifizieren eine Ressource über ihren Zugriffmechanismus und über ihre Position innerhalb des Computer Netzwerkes. Bei URNs hingegen ist die Position der Ressource unabhängig, d.h. URNs bleiben erhalten auch wenn sich die Position innerhalb eines Netzwerkes ändert. URCs werden verwendet für die Angabe von Metadaten. Es müssen aber nicht alle drei Unterarten verwendet werden. In den letzten Jahren wurden auch noch andere Konzepte zur Benennung und Charakterisierung von Webseiten vorgestellt. Die zwei Unterarten URNs und URCs werden sehr viel seltener verwendet als die Unterart URL. URLs werden in Form von Webadressen oft verwendet. (Brockhaus, 2002; Moats, 1997; Berners-Lee, 2005)

### 3.2.2 Schicht 2: XML + NS + xmlschema

Durch die zweite Schicht ist es unter anderem möglich, strukturierte Dokumente zu erstellen und zu speichern. Hierfür wird XML, der Namespace (NS)<sup>7</sup> und das XML Schema verwendet. XML wird auch als Format für den Datenaustausch zwischen Applikationen verwendet. (Harmelen und Antoniou, 2004)

Wie auch Erlenkötter (2003) beschreibt, ist XML keine Programmiersprache, sondern eine Sprache, mit welcher Daten beschrieben werden. Genauer gesagt ist XML eine Metasprache<sup>8</sup>. XML wurde entwickelt, damit Daten und ihre Struktur in einem unabhängigen Format beschrieben werden können und diese natürlich auch gespeichert werden können. Die XML Tags<sup>9</sup> bestehen aus spitzen Klammern, in welchen dann der Text steht (z.B. `<vorname>Homer</vorname>`). Hiermit wird dann die Struktur und der Inhalt eines Dokumentes beschrieben. Es wird aber damit nicht die Bedeutung der Tags festgelegt. Zu XML gehören noch eine ganze Reihe weiterer Sprachen. Einige davon sind folgende:

- XInclude (XML Inclusions) - Hiermit wird beschrieben wie mehrere Dokumente zu einem einzigen Dokument zusammengefügt werden können.
- XPath - Hiermit wird beschrieben wie die logischen Bestandteile eines Dokumentes adressiert werden können.
- XPointer (XML Pointer Language) - Hiermit wird beschrieben wie man mit XPath Ausdrücken auf Teile von XML Dokumenten verweisen kann.
- XSLT (eXtensible Stylesheet Language Transformation) - Hiermit wird beschrieben wie ein XML Dokument in ein anderes Dokument, wie beispielsweise ein HTML Dokument, umgewandelt werden soll.

Der Namespace ist ein Konstrukt, welches von Programmiersprachen wie z.B. Java übernommen wurde. Durch ihn ist es möglich, Namen einer Dokumentenklasse oder einer Anwendung zusammenzufassen und die eindeutige Verwendung sicherzustellen.

---

7. im deut. Namensraum

8. Sprache zur Beschreibung von Sprachen

9. Markierungen bzw. Marken, die der Sprache ihren Namen gegeben haben

Sollten gleiche Namen vorhanden sein, so ist es dadurch möglich, dass gleiche Namen von einander unterschieden werden können. Bei einer Programmiersprache wie z.B. Java entspricht dies dem Konzept der `import` Anweisungen. Durch eine `import` Anweisung können zwei gleichnamige Klassen aufgrund der Zugehörigkeit zu verschiedenen Packages unterschieden werden. Eine genaue Beschreibung zu diesem Thema findet sich unter anderem bei Erlenkötter (2003).

XML Schema ist eine leistungsfähige und auch komplexe Technologie. Durch XML Schema wird ein Regelwerk definiert, durch welches der Inhalt sowie die Bedeutung eines XML Dokumentes festgelegt werden kann. Hiermit ist es möglich XML um Regeln, mit welchen man z.B. die Multiplizität eines Elements angeben kann, zu erweitern. Das XML Schema bietet auch die Möglichkeit, dass die Struktur des Dokumentes angegeben wird. Ein XML Dokument muss wohlgeformt sein, d.h. das Dokument muss der XML Syntax entsprechen. Bei der Validierung wird geprüft, ob das XML Dokument mit dem zugeordneten XML Schema konform ist. Ein gültiges XML Dokument muss wohlgeformt und valide sein. (Erlenkötter, 2003)

Natürlich bietet XML, der Namespace und das XML Schema noch weitere Möglichkeiten und Vorteile, welche hier nicht erwähnt wurden. Deshalb sei an dieser Stelle auf die jeweilige Fachliteratur, wie z.B. W3C und Quin (2003) oder Erlenkötter (2003), verwiesen. Eine detaillierte Beschreibung all dieser Themen würde an dieser Stelle den Rahmen sprengen.

### 3.2.3 Schicht 3: RDF + rdfschema

Zu dieser Schicht sei an dieser Stelle erwähnt, dass sie sich mit der Interpretation der Bedeutung von Tags für Maschinen befasst. Eine genaue Beschreibung dieser Thematik findet im Kapitel 4 statt.

### 3.2.4 Schicht 4: Ontology Vocabulary (Ontologien Vokabular)

Über die vierte Schicht werden weitere Vokabulare und Strukturen bereitgestellt. Wie auch Birkenbihl (2006) beschreibt wird für den Aufbau einer Ontologie die Möglichkeit benötigt, Klassen zu konstruieren und miteinander zu verknüpfen. Es wird über diese Schicht das RDF Schema um neue Klassen und Prädikate erweitert. Das Ziel dieser Schicht ist es, das Problem, Terminologien für einen bestimmten Zusammenhang zu erstellen, zu lösen. Weitere Ziele dieser Schicht sind, dass Eigenschaften besser eingeschränkt werden können und dass logische Charakteristiken von Eigenschaften sowie die Äquivalenz von Begriffen definiert werden können. Es werden dem RDF Schema verschiedene Ressourcen und Eigenschaften hinzugefügt, die es erlauben, durch:

- die Aufzählung der Instanzen
- den Durchschnitt mit anderen Klassen (was in Klasse A und in Klasse B ist)

- die Vereinigung mit anderen Klassen (was in Klasse A oder in Klasse B ist)
- die Kardinalitäten ( $n$  Instanzen aus, mehr/weniger als  $n$  Instanzen aus A)
- die Komplementäroperation (alles, was nicht in Klasse A ist)

neue Klassen zu konstruieren. Damit können auch Subklassen durch einschränkende Eigenschaftswerte der Instanzen gebildet werden. Eine der Sprachen für Web - Ontologien ist die „Web Ontology Language“<sup>10</sup>.

Zur Zeit nimmt die Semantik von OWL das logische Standardmodell der „Open world assumption“ an, dies ist eines der Grundaspekte von Ontologien. Die „Open world assumption“ geht davon aus, dass nicht automatisch angenommen werden kann, dass etwas nicht existiert, nur weil es nicht definiert ist. Wenn von einer Behauptung das Gegenteil nicht bewiesen oder nicht erreicht wurde, dann kann nicht automatisch angenommen werden, dass die Behauptung wahr ist. (Harmelen und Antoniou, 2004)

Ein weiterer Grundaspekt von Ontologien ist die „No Unique-Names Assumption“<sup>11</sup>, welcher ein wesentlicher Unterschied zur typischen Datenbank ist. Eine typische Datenbankapplikation geht davon aus, dass zwei Personen mit unterschiedlichen Namen auch unterschiedliche Menschen sind. Dies gilt natürlich auch für Eigenschaften, Klassen, und ähnlichem. Bei OWL hingegen ist dies anders, wenn hier zwei Personen einen unterschiedlichen Namen haben, dann kann es durch die Schlussfolgerung möglich sein, dass es sich bei diesen zwei unterschiedlichen Instanzen um eine einzige Person handelt. (Harmelen und Antoniou, 2004)

Bei OWL wird zwischen Objekteigenschaften und Datentypeneigenschaften unterschieden. Aufgrund der Komplexität wurde OWL in die drei Stufen „OWL Full“, „OWL Description Logic“<sup>12</sup> und „OWL Lite“ gegliedert. (Birkenbihl, 2006; Harmelen und Antoniou, 2004)

#### 3.2.4.1 OWL Full

OWL Full ist die gesamte Sprache, welche alle OWL Sprachprimitive verwendet. Es wird das Maximum an Möglichkeiten zur Verfügung gestellt und es kann RDF Schema voll genutzt werden. OWL Full kann als Erweiterung von RDF angesehen werden. Hiermit kann z.B. eine Klasse gleichzeitig als eine Sammlung von Instanzen und als eine Instanz behandelt werden. Es wird einer Ontologie erlaubt, die Bedeutung des vordefinierten RDF oder OWL Vokabulars zu erweitern. Diese Sprache ist allerdings so mächtig geworden, dass sie nicht mehr entscheidbar ist. Jedes OWL Dokument ist ein RDF Dokument und jedes RDF Dokument ist ein OWL Full Dokument, aber nur einige

---

10. Im Folgendem weiter OWL genannt

11. im deut. Annahme von nicht eindeutigen Namen

12. Im Folgendem weiter OWL DL genannt

RDF Dokumente sind gültige OWL DL oder OWL Lite Dokumente. (Birkenbihl, 2006; Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

#### 3.2.4.2 OWL DL

OWL DL kann als Erweiterung einer eingeschränkten Ansicht von RDF angesehen werden. OWL DL ist eine Unterklasse von OWL Full. Es wird ein Maximum an Ausdrucksfähigkeit angeboten, während die Vollständigkeit und Entscheidbarkeit behalten wird. Der Nachteil von OWL DL ist, dass die volle Vereinbarkeit mit RDF verloren geht. Ein RDF Dokument wird einerseits teilweise erweitert und andererseits eingeschränkt werden müssen, bevor es ein gültiges OWL DL Dokument ist. Jedes gültige OWL DL Dokument ist ein gültiges RDF Dokument. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

#### 3.2.4.3 OWL Lite

Ebenfalls als Erweiterung einer eingeschränkten Ansicht von RDF kann OWL Lite angesehen werden. OWL Lite ist eine Unterklasse von OWL DL, welche effizient zu realisieren ist. Für Klassifikationshierarchien mit einfachen Einschränkungen kann OWL Lite gut verwendet werden. Es werden die Kardinalitäten auf 0 und 1 eingeschränkt. Des Weiteren wird ein schneller Migrationspfad für Thesauri und andere Taxonomien zur Verfügung gestellt. OWL Lite hat eine niedrigere formelle Komplexität als OWL DL. Dadurch wird ein einfach implementierbares und nützliches Subset von OWL bereitgestellt. (Birkenbihl, 2006; Eckstein und Eckstein, 2003; Smith u. a., 2004; Harmelen und Antoniou, 2004)

### 3.2.5 Schicht 5: Logic (Logik)

Die Ebene „Logic“ erweitert die vorherige Schicht um die Möglichkeit, dass anwendungsspezifische Regeln formuliert werden können. Auf dieser Ebene sollen Daten so beschrieben werden, dass Maschinen diese verarbeiten können. Um eine Automatisierung zu erreichen, muss es möglich sein aus den Daten Schlussfolgerungen zu ziehen. (Harmelen und Antoniou, 2004; Wichmann, 2007)

### 3.2.6 Schicht 6: Proof (Beweis)

Diese Schicht der Semantic Web Architektur umfasst den aktuellen Schlussfolgerungsprozess. Es werden die Beweise in Websprachen dargestellt und es wird die Beweisführung validiert. (Harmelen und Antoniou, 2004)

### **3.2.7 Schicht 3 - 6: Digital Signature (Digitale Signatur)**

Parallel zur Schicht 3, Schicht 4 und Schicht 5 sowie zur Schicht 6 wird eine verlässliche digitale Signatur entwickelt zur jeweiligen Schicht. Eine digitale Signatur wird durch ein Verschlüsselungsverfahren erzeugt, mit welchem die Herkunft sowie die Echtheit von digitalen Daten belegt werden kann. Weiter wird beschrieben, dass die Kommunikation über das Internet gewisse Sicherheitsrisiken mit sich bringt. Es kann sein, dass die Datenübertragung zwischen Sender und Empfänger abgehört, abgefangen oder auch manipuliert wird. Des Weiteren kann es auch sein, dass gegenüber dem Kommunikationspartner falsche Angaben bezüglich Name oder Adresse gemacht werden. Für eine manipulationssichere Kommunikation und für eine Identifikation der Kommunikationspartner wird mit Hilfe der digitalen Signatur gesorgt. Mit einer digitalen Signatur wird aber kein Schutz für die Vertraulichkeit einer Nachricht gegeben. Wenn auch dies erreicht werden soll, so muss zusätzlich eine Verschlüsselung verwendet werden. (Brockhaus, 2002)

### **3.2.8 Schicht 7: Trust (Vertrauen)**

In dieser Schicht werden Daten auf ihre Vertrauenswürdigkeit geprüft. Die Prüfung der Daten auf ihre Vertrauenswürdigkeit ist ein wichtiges Verfahren. Das World Wide Web (WWW) kann nur dann sein volles Potential erreichen, wenn die Daten vertrauenswürdig sind. (Harmelen und Antoniou, 2004)

### **3.2.9 Weiterentwicklung**

Das Semantic Web ist ein aktives sowie auch dynamisches Forschungsgebiet. Dadurch kommt es laufend zu Weiterentwicklungen, auch bei der Architektur des Semantic Web. Abbildung 3.2 zeigt eine weiterentwickelte Architektur. Die Grundelemente, welche zuvor beschrieben wurden, sind auch hier enthalten. Das Schichtenmodell aus Abbildung 3.2 unterscheiden sich zu dem Schichtenmodell aus 3.1 durch die Anordnung bzw. durch den Übergang zwischen den einzelnen Schichten und durch eine stärkere Ausdifferenzierung.

## **3.3 Aktuelle Initiativen und Einsatzfelder**

In diesem Abschnitt wird auf die EU und die Semantic Systems Bezug genommen. Des Weiteren wird auch die Förderung von Semantic Systems innerhalb von Österreich beschrieben. In diesem Zusammenhang werden auch Semantic Systems bei betrieblichen Anwendungen behandelt.

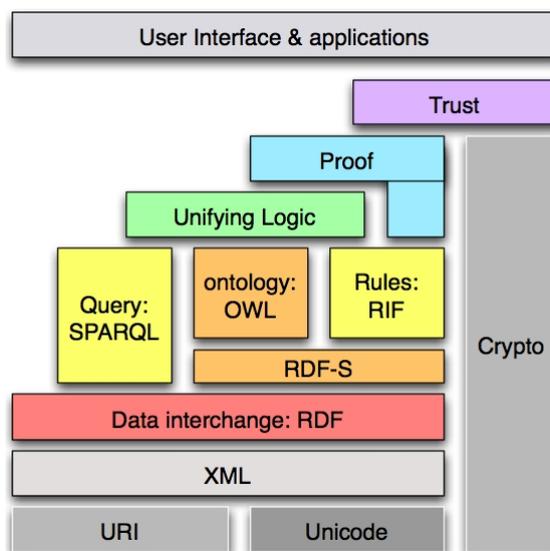


Abbildung 3.2: Weiterentwickelte Architektur des Semantic Web (Quelle: Bratt, 2006, W3C)

### 3.3.1 Semantic Systems / Technologien in der EU

Das Semantic Web und seine Technologien gewinnen immer mehr an Bedeutung. Deshalb ist es auch nicht verwunderlich, dass auch die EU darauf reagiert. Software Entwicklung wurde bisher eher breitflächig gefördert, aber Initiativen auf der europäischen Ebene zeigen das eine Fokussierung der „Forschung & Entwicklung“<sup>13</sup> immer mehr an Bedeutung gewinnt.

Die EU will ihre Produktivität und ihre Innovation durch verschiedene Maßnahmen erhöhen. Eine genaue Definition bzw. eine Festlegung dieser Maßnahmen findet in der „Lissabon-Strategie“ statt. In diesem Zusammenhang sollen die Ausgaben für F&E auf 3% des europäischen BIP<sup>14</sup> ansteigen. Hierdurch soll die Innovation in der gesamten europäischen Gemeinschaft verbessert werden. Mit der „Lissabon-Strategie“ will die EU sicherstellen, dass das Produktivitätsgefälle zwischen ihr und ihren Konkurrenten (wie z.B.: der USA) ausgeglichen bleibt. Natürlich hat die „Lissabon-Strategie“ noch weitere detaillierte Ziele, welche hier nicht beschrieben werden, da dies den Rahmen sprengen würde. Weitere Details hierzu findet sich unter anderem bei der Kommission der Europäischen Gemeinschaften (2003) und beim Bundesministerium für Finanzen (2007).

Rund um die „Lissabon-Strategie“ existiert ein Rahmenprogramm, welches die Arbeits-

13. Im Folgendem weiter F&E genannt

14. Bruttoinlandsprodukt

programme mit ihren thematischen Inhalten<sup>15</sup> und die Maßnahmen definiert. Diese sollen durchgeführt werden, damit die definierten Ziele der „Lissabon-Strategie“ erreicht werden. In diesem Rahmenprogramm werden unter anderem auch Projekte mit einem Bezug zu semantischen Technologien ausgeschrieben. Hier finden sich nun auch Arbeitsprogramme mit Aktionslinien, welche sich explizit mit Wissensmanagement, Ontologien oder web-basierten Diensten befassen bzw. diese nachfragen. Unter anderem wurde eine eigene Aktionslinie „Semantische Webtechnologien“ für Projekteinrichtungen geöffnet. In dieser Aktionslinie werden vier Schwerpunkte vorgegeben. Wohlkinger und Pellegrini (2006) beschreiben diese Aktionslinien wie folgt:

1. Kodierung und Strukturierung digitaler Inhalte, für die Definition und Erläuterung ihrer Semantik
2. Ableitung semantischer Attribute von webbasierten Inhalten
3. Semantikbasierte Werkzeuge für das Auffinden von Wissen sowie die intelligente Filterung und Profilerstellung
4. Intelligente und visuelle Schnittstellen, die semantische Informationsstrukturen nutzen

Im Laufe dieses Rahmenprogramms hat eine Abteilung innerhalb der Europäischen Kommission dieses Thema konkretisiert. Diese Abteilung nennt sich: Unit E2 „Knowledge and Content Technologies“. In verschiedenen thematischen Förderungsschwerpunkten finden sich immer mehr Projekte mit einem Bezug zu Semantic Systems. Einige solcher thematischen Förderungsschwerpunkten sind unter anderem „Networked business and governments“, „eSafety of road and air transports“ und „eHealth“. (Wohlkinger und Pellegrini, 2006)

In weiterer Folge dieses Rahmenprogramms zur „Lissabon-Strategie“ etablieren sich die semantischen Technologien immer weiter in Anwendungsgebieten, Geschäftsbereichen und in technologischen Forschungsgebieten. Der Anteil der Arbeitsprogramme mit Aktionslinien, welche den Bereich der „Semantic Systems“ mehr oder weniger stark thematisiert, steigt bis auf 36% der gesamten Arbeitsprogramme an. (Wohlkinger und Pellegrini, 2006)

### 3.3.2 Semantic Systems / Technologien in Österreich

Auf dem Gebiet der Informationstechnologie fördert das Impulsprogramm „FIT-IT“ des Bundesministeriums für Verkehr, Innovation und Technologie<sup>16</sup> anspruchsvolle Innovation und Technologieentwicklung im Rahmen kooperativer Forschung. Hierbei stehen visionäre interdisziplinäre Projekte im Mittelpunkt, welche als Ergebnis einen Funktionsnachweis der technologischen Lösung (z.B. ein Prototyp) haben. Eine Produktent-

---

15. thematische Inhalte werden in diesem Zusammenhang auch Aktionslinien genannt

16. Im Folgenden weiter BMVIT genannt

wicklung alleine wird von FIT-IT nicht gefördert. (Bundesministerium für Verkehr, Innovation und Technologie, 2007)

Seit 2006 werden vom Impulsprogramm FIT-IT fünf Programmlinien verfolgt. Diese Programmlinien sind folgende:

- Embedded Systems
- Semantische und intelligente Systeme und Dienste
- Systems on Chip
- Trust in IT Systems
- Visual Computing

Die Programmlinie „Semantische und intelligente Systeme und Dienste“ von FIT-IT hat das Ziel, technologische Voraussetzungen zu schaffen, um die Interaktion zwischen Mensch und Computer und zwischen Informationssystemen zu vereinfachen. Hier ist ein großes wirtschaftliches Potential vorhanden. Neue Technologien werden hier als Grundlage für die Entwicklung von zukünftigen Anwendungen von Semantic Systems gefördert. Dies geschieht in Kooperation mit der Spitzen-Wissenschaft und der innovativen Wirtschaft. Mit dieser Initiative will Österreich zu den weltweiten Spitzenreitern in wirtschaftlich und sozial verwertbaren semantischen Technologien werden. (Bundesministerium für Verkehr, Innovation und Technologie, 2007)

### 3.3.3 Semantic Systems / Technologien bei betrieblichen Anwendungen

Der Bedarf zur Beschreibung der Bestandteile und Prozesse von Dienstleistungen und Waren steigt an. Dies ist mitunter auf die steigende Komplexität der Dienstleistung und der Waren zurückzuführen. Hierfür werden beispielsweise semantische Technologien benötigt. Nicht nur Menschen brauchen Informationen damit sie handeln können, sondern auch Maschinen. Wenn Informationen, welche zum Handeln benötigt werden, sprachlich codiert werden, dann kann es auch vorkommen, dass unterschiedliche Bezeichnungen zu vagen Repräsentationen ihrer Bedeutung führen. (John und Drescher, 2006)

Semantische Technologien werden in der organisationalen Prozessunterstützung bevorzugt für die Mensch-Maschine-Kommunikation verwendet. Im organisationalen und betriebswirtschaftlichen Kontext können semantische Technologien dazu beitragen, dass die natürliche Sprache operationalisiert wird. Dies hängt damit zusammen, dass die Geschäftsprozesse auf Informations- und Kommunikationsprozessen basieren. Semantische Technologien unterstützen bei betrieblichen Anwendungen die unternehmerischen Wissens- und Geschäftsprozesse. Hierbei werden sie unter anderem für die Analyse und Extraktion, als auch für die Modellierung und Verknüpfung, sowie für die Personalisierung und Qualifizierung bestehender Daten verwendet. Daher bezeichnen semantische Technologien in diesem Zusammenhang technische Verfahrensweisen zur Übermittlung

und Verarbeitung von Bedeutungen. Um zu einem gemeinsamen Verständnis von Prozessen und in der jeweiligen Anwendungsdomäne verwendeten Termini zu gelangen, können also semantische Technologien und Methoden in betrieblichen Anwendungen eingesetzt werden. (John und Drescher, 2006)

John und Drescher (2006) beschreiben, dass die semantischen Technologien, welche zurzeit eingesetzt werden auf den Prinzipien der Informationsverarbeitung aus den folgenden Bereichen basieren:

- kognitive Systemmodellierung
- künstliche Intelligenz
- Datenbankmodellierung
- Information Retrieval

Semantische Technologien können zur Repräsentation von deklarativem als auch prozeduralem Wissen eingesetzt werden. Explizit ausgedrücktes begriffliches Wissen spielt für Informationsprozesse sowie Wissensprozesse eine wichtige Rolle. Durch den gegenseitigen Austausch von Semantik werden diese verstärkt integriert. Dies soll eine Verbesserung des Informations- und des Kommunikationsprozesses zur Folge haben. Die Ergebnisse der semantischen Technologien bzw. der Prozesse stehen dann als Handlungsgrundlage zur Verfügung. Die Verwendung bzw. die Einführung von semantischen Technologien wird von den meisten Unternehmen noch als zu aufwendig betrachtet. Dies hängt damit zusammen, dass zum Aufbau einer Wissensbasis umfangreiche Modellierungsarbeiten und auch Wartungsarbeiten notwendig sind. In diesem Zusammenhang ist es für die Unternehmen wichtig ein Maß zu entwickeln, mit welchem sie den Aufwand und Nutzen für die Investition in ein semantisch basiertes System abschätzen können. (John und Drescher, 2006)

### 3.3.4 Semantic Web Services

Unter Semantic Web Services versteht man Webseiten, welche nicht nur statische Informationen liefern, sondern die BenutzerInnen durch Interaktion auch mit einbeziehen. Es werden hierbei über das World Wide Web (WWW) Funktionen bzw. Programme als Dienst für die BenutzerInnen angeboten. Normalerweise unterscheidet man zwischen einfachen und komplexen Semantic Web Services. (Harmelen und Antoniou, 2004; Eckstein und Eckstein, 2003)

Bei Harmelen und Antoniou (2004) wird beschrieben, dass bei einfachen Web Services keine komplexe Interaktion mit dem bzw. der BenutzerIn verlangt wird, ausser einfachen Antworten. „Informations-Bereitstellung-Services“, wie z.B. ein Flugfinder oder ein Service welcher die Postleitzahl zu einer gegebenen Adresse findet, sind Beispiele für einfache Semantic Web Services. Aus einfachen Web Services werden dann komplexe Semantic Web Services aufgebaut. Diese komplexen Web Services verlangen im

Gegensatz zu einfachen Web Services häufig eine andauernde Interaktion mit dem bzw. der BenutzerIn. Ein Beispiel wäre hier die Interaktion von einem bzw. einer BenutzerIn mit einem Online Musik Geschäft. Hier wären unter anderem folgende Interaktionen möglich:

- Suchen nach CDs oder Musiktiteln mit verschiedenen Kriterien
- Lesen von Bewertungen und das Hören von Samples<sup>17</sup>
- Artikel in den virtuellen Warenkorb geben
- Kreditkarteninformationen angeben

Bei Web Services wird zurzeit noch die Interaktion mit dem bzw. der BenutzerIn benötigt. Das Ziel hierbei ist es, dass das Auffinden, der Aufruf und die Beobachtung von Web Services automatisch durchgeführt wird. Dies soll mit Hilfe von Maschinen interpretierbaren Beschreibungen von Services zur Verfügung gestellt werden. Webseiten sollten im Stande sein, eine Reihe von grundlegenden Klassen und Eigenschaften zu verwenden. (Harmelen und Antoniou, 2004)

Auf dem Gebiet der Semantic Web Services werden sehr effektiv Techniken aus der künstlichen Intelligenz eingesetzt. Zu einem solchen Service gehören ein „Service Profil“<sup>18</sup>, ein „Service Modell“<sup>19</sup> und ein „Service Fundament“<sup>20</sup>. (Harmelen und Antoniou, 2004)

Das Service Profil ist die Spezifikation eines Services. Dadurch werden die Voraussetzungen, welche erfüllt sein müssen, damit der Service genutzt werden kann, beschrieben. Des Weiteren werden die Funktionen des Services beschrieben. Dies ist dafür notwendig, dass ein Service z.B. von einem Agent gefunden werden kann. Wenn ein Agent einen bestimmten Service sucht, so kann er durch das Service Profil feststellen, ob dieser Service für seinen Zweck passend ist. (Harmelen und Antoniou, 2004)

Ein Service Model beschreibt wie ein Service arbeitet. Es wird beschrieben, was geschieht, während das Service ausgeführt wird. Dies kann für einen Agenten, welcher einen Service sucht, von Interesse sein, um Services zusammenzusetzen, damit eine komplexe Aufgabe durchgeführt werden kann. Diese Informationen sind auch dafür wichtig, dass die Ausführung des Services kontrolliert werden kann. (Harmelen und Antoniou, 2004)

Ein Service Fundament spezifiziert wie ein Agent auf diesen Service zugreifen kann. Normalerweise spezifiziert ein Service Fundament ein Kommunikationsprotokoll und die Port Nummern, welche verwendet werden, wenn der Service kontaktiert wird. (Harmelen und Antoniou, 2004)

---

17. ein Ausschnitt einer Musik Aufnahme

18. im engl. service profile

19. im engl. service model

20. im engl. service grounding

## Kapitel 4

### RDF (Resource Description Framework)

RDF besteht im Wesentlichen aus drei Teilen. Diese Teile sind das „RDF Datenmodell“, die „RDF Syntax“ und das „RDF Schema“. Diese Teile werden in den jeweiligen Abschnitten beschrieben.

Des Weiteren werden ein paar Vertreter der „Query Sprachen“<sup>1</sup> für RDF beschrieben. Anschließend wird auf die Verwendung von RDF, sowie auf die Entwicklungen Bezug genommen.

#### 4.1 RDF Datenmodell

RDF besteht aus einer XML ähnlichen Syntax und einem grafischen Modell zur Darstellung und Annotation von Metadaten. Im Gegensatz zu XML muss RDF zusätzliche Anforderungen erfüllen. Wie bereits im Abschnitt 3.2.2 beschrieben wurde, bietet XML zwar die Möglichkeit, Daten flexibel in einer einheitlichen Sprache zu strukturieren, aber es gibt hier keine Möglichkeit, dass die Bedeutung der Tags und der Daten maschinenlesbar definiert wird. Hier wird die Bedeutung von den Tags und Daten nur vom Menschen interpretiert. Wenn man zu den Daten selbst noch Informationen speichern will und diese mit anderen Informationen in Beziehung setzen will, dann wird noch ein anderer Ansatz benötigt. Um dies zu erreichen, wurde zuerst eine einfache Möglichkeit gesucht um Ressourcen zu beschreiben. Diese Möglichkeit ist durch RDF gegeben. (Birkenbihl, 2006; Eckstein und Eckstein, 2003)

Das RDF Modell wird als ein grafisches Modell beschrieben, welches aus Knoten und Kanten besteht. Die grundlegenden Konzepte von RDF sind „*Ressource*“<sup>2</sup>, „*Eigenschaften*“<sup>3</sup> und „*Aussagen*“<sup>4</sup>. (Eckstein und Eckstein, 2003)

- 
1. im deut. Abfragesprache
  2. im engl. resources
  3. im engl. properties
  4. im engl. statements

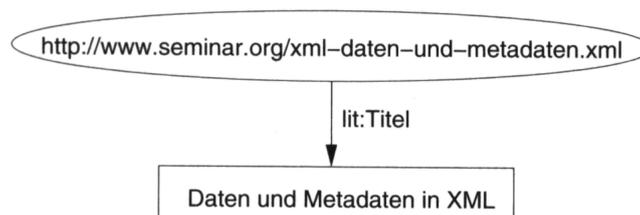
Eine Ressource wird definiert, als ein Objekt, über welches man „sprechen“ möchte. Ressourcen sind durch URIs eindeutig identifizierbar. Im grafischen Modell werden Ressourcen durch eine Ellipse abgebildet. Beispiele für Ressourcen sind z.B. einzelne Web-Seiten, eine Sammlung von Web-Seiten, Objekte auf welche nicht direkt über das Web zugegriffen wird (z.B. Bücher, Hotels, Zimmer), usw. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

Eine Eigenschaft beschreibt eine Ressource und die Beziehung zwischen Ressourcen. Eigenschaften im grafischen Modell werden durch eine benannte Kante abgebildet, welche die Knoten für die Ressource und den Wert der Eigenschaft verbindet. Durch Rechtecke wird der Wert der Eigenschaft abgebildet. Jede Eigenschaft besitzt eine spezifische Bedeutung, welche definiert ist durch:

- ihre erlaubten Werte
- die Typen von Ressourcen, die damit beschrieben werden können
- ihre Beziehungen zu anderen Eigenschaften

Eine Aussage besteht aus einer Ressource, einer Eigenschaft und einem Wert. Durch die Verknüpfung von Aussagen ergibt sich ein gerichteter Graph, welcher aus Ressourcen und ihren Eigenschaften besteht. Die Ressourcen werden in Beziehung zueinander gesetzt, indem sie durch eine oder mehrere Eigenschaften miteinander verbunden werden. Alles was auf die gleiche URI zeigt, verwendet das gleiche Konzept. (Birkenbihl, 2006; Harmelen und Antoniou, 2004; Eckstein und Eckstein, 2003)

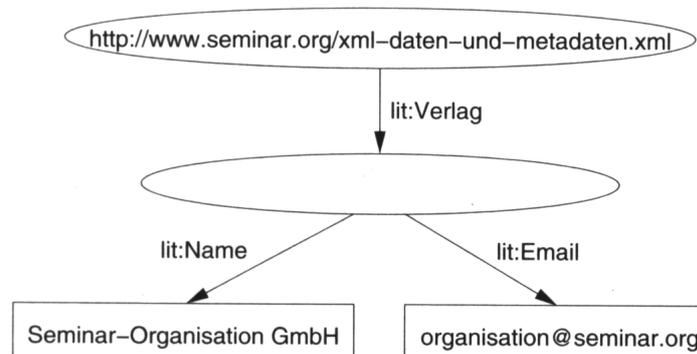
Die Abbildung 4.1 zeigt ein einfaches Beispiel für eine RDF Aussage. In der Ellipse wird die Ressource durch die URI „<http://www.seminar.org/xml-daten-und-metadaten.xml>“ dargestellt. Die Eigenschaft dieser Ressource wird durch die benannte Kante „*lit:Titel*“ beschrieben. Im Rechteck wird der Wert „*Daten und Metadaten in XML*“ der Eigenschaft beschrieben. Diese Aussage kann man in einem Satz formulieren, welcher dann wie folgt lautet: *Das Objekt mit der URI „<http://www.seminar.org/xml-daten-und-metadaten.xml>“ hat den Titel „Daten und Metadaten in XML“.* (Eckstein und Eckstein, 2003)



**Abbildung 4.1:** Einfache RDF Aussage (Quelle: Eckstein und Eckstein, 2003, S. 240)

Es besteht auch die Möglichkeit, strukturierte Werte für die Beschreibung zu verwenden. Hierfür wird eine leere Ellipse verwendet, welcher die einzelnen Werte als Rechtecke

zugeordnet werden. Die Zuordnung wird über benannte Kanten vorgenommen. Die Abbildung 4.2 zeigt eine RDF Aussage mit strukturierten Werten zur Beschreibung. Die Eigenschaft „Verlag“ wird als Ressource dargestellt, welche mit den zwei Eigenschaften „Name“ und „Email“ und den entsprechenden Werten beschrieben wird. (Eckstein und Eckstein, 2003)



**Abbildung 4.2:** RDF Aussage mit strukturierten Werten zur Beschreibung (Quelle: Eckstein und Eckstein, 2003, S. 240)

Es ist möglich Aussagen in der Form von „*Subjekt*“<sup>5</sup>, „*Prädikat*“<sup>6</sup> und „*Objekt*“<sup>7</sup> (*s p o*) zu treffen. Das vorherige Beispiel kann wie folgt interpretiert werden:

„Das XML Dokument „<http://www.seminar.org/xml.daten-und-metadaten.xml>“ besitzt einen Verlag, der durch den genannten Namen „Seminar-Organisation GmbH“ und die genannte Email Adresse „organisation@seminar.org“ beschrieben wird.“

(Eckstein und Eckstein, 2003)

## 4.2 RDF Syntax

Die RDF Syntax bietet verschiedene Konstrukte für diverse Anwendungsbereiche an, welche in den entsprechenden Unterabschnitten beschrieben werden.

### 4.2.1 RDF Basis Syntax

Die Basis Syntax für RDF basiert auf XML. Bei einem RDF Dokument ist das Wurzelement immer `RDF`, welchem `rdf:` für den RDF Namensraum vorangestellt sein sollte. Dies ist im Listing 4.1 dargestellt.

- 
- 5. wird für eine Ressource verwendet
  - 6. wird für eine Eigenschaft verwendet
  - 7. wird für die Werte verwendet

```

1 <rdf:RDF
2   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
4   xmlns:uni="http://www.mydomain.org/uni-ns#">
5   <rdf:Description rdf:about="949352">
6     .....
7   </rdf:Description>
8 </rdf:RDF>

```

**Listing 4.1:** RDF Basis Syntax (Quelle: Harmelen und Antoniou, 2004, S. 70)

Ein RDF-Element kann beliebig viele beschreibende Elemente besitzen. Ein beschreibendes Element ist das Description-Element und das Type-Element. Innerhalb des Description - Elements `rdf:Description` findet die Ressourcen Beschreibung statt. Eine Ressource wird zuvor von dem Attribut `ID` oder `about` referenziert. Das `ID` Attribut wird verwendet, wenn die Ressource keine URI besitzt. Dadurch wird eine Ressource definiert. Wenn eine Ressource beschrieben werden soll, welche im World Wide Web (WWW) verfügbar ist oder welche zumindest über eine URI identifizierbar ist, dann wird das `about` Attribut verwendet und als Wert wird die URI gesetzt. Durch dieses Attribut wird die Definition einer Ressource erweitert. Das Listings 4.2 zeigt die Verwendung dieser Attribute. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

```

1 <rdf:Description rdf:about="CIT1111">
2   .....
3 </rdf:Description>
4
5 <rdf:Description rdf:ID="#949318">
6   .....
7 </rdf:Description>

```

**Listing 4.2:** Verwendung der Attribute „`rdf:about`“ und „`rdf:ID`“ (Quelle: Harmelen und Antoniou, 2004, S. 73)

Der Inhalt eines Description-Elements kann aus beliebig vielen Eigenschaften bestehen. Die Eigenschaften werden mit Eigenschaftselementen<sup>8</sup> beschrieben. Diese können frei definierbare Namen besitzen. Es wird zwischen verschiedenen Arten von Eigenschaftselementen unterschieden. Diese sind folgende:

8. im engl. property element

- beliebiger, beschreibender XML-Text
- Description-Element
- `resource` Attribute

Beim XML-Text muss als Wert eine URI vorhanden sein. Hiermit wird angegeben, dass eine andere Ressource der Wert für diese Eigenschaft ist. So kann vermieden werden, dass Ressourcen redundant beschrieben werden. Das Attribut `resource` besitzt als Wert eine URI. Hier ist eine andere Ressource der Wert für diese Eigenschaft. Im Listing 4.3 werden die Eigenschaftselemente dargestellt. Das Listing 4.3 enthält zwei Description-Elemente, welche durch `<rdf:Description>` erkennbar sind. Dabei enthält das erste Description-Element die zwei Eigenschaftselemente `<uni:name>` und `<uni:title>`, welche beide einen String als Inhalt besitzen. Das zweite Description-Element enthält das Eigenschaftselement `<uni:courseName>`. Durch die Zeile `<uni:isTaughtBy rdf:resource="#949318"/>` wird ein Verweis auf das erste Description-Element gegeben. In diesem Beispiel bedeutet dies, dass der Kurs „Diskrete Mathematik“ von „Bart Simpson“ unterrichtet wird. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

```
1 <rdf:Description rdf:ID="#949318">
2   <uni:name>Bart Simpson</uni:name>
3   <uni:title>Professor</uni:title>
4 </rdf:Description>
5
6 <rdf:Description rdf:about="CIT1111">
7   <uni:courseName>Diskrete Mathematik</uni:courseName>
8   <uni:isTaughtBy rdf:resource="#949318"/>
9 </rdf:Description>
```

**Listing 4.3:** Verwendung der Eigenschafts-Elemente (Quelle: Harmelen und Antoniou, 2004, S. 73)

Eine Ressource kann durch die Verwendung des Type-Elements einem Typ zugeordnet werden. Dadurch kann man einer Ressource explizit allgemeine Eigenschaften einer Klasse zuordnen. Das Type-Element ist wie das Description Element ein beschreibendes Element und besitzt auch die Attribute `ID` und `about`. Ein Type-Element kann beliebig viele Eigenschaftselemente enthalten. Das Listing 4.4 zeigt ein Description-Element, welches das Attribute `type` enthält. Die Ressource wird durch die Verwendung von `<rdf:type rdf:resource="&uni;course"/>` typisiert. Dadurch wird angegeben, dass die Ressource eine Lehrveranstaltung ist. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

```
1 <rdf:Description rdf:about="CIT1111">  
2   <rdf:type rdf:resource="&uni;course"/>  
3   <uni:courseName>Diskrete Mathematik</uni:courseName>  
4   <uni:isTaughtBy rdf:resource="949318"/>  
5 </rdf:Description>
```

**Listing 4.4:** Verwendung des Type-Elementes (Quelle: Harmelen und Antoniou, 2004, S. 74)

Eine Verschachtelung der verschiedenen Elemente ist natürlich auch möglich. Eine detaillierte Erläuterung hierzu findet sich bei Eckstein und Eckstein (2003) sowie bei Harmelen und Antoniou (2004).

#### 4.2.2 Container Elemente

Es wird oft die Möglichkeit benötigt, dass mehrere ähnliche Beschreibungen zusammengefasst werden, z.B. wenn zwei Autoren bei einem Buch angegeben werden. Hierfür gibt es das Konzept der Container-Elemente. Dieses Konzept kann überall dort verwendet werden, wo auch Beschreibungen erlaubt sind. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

Es existieren hier die Container Arten „Sequenz“, „Multimenge“ und „Alternative“. Eine Sequenz ist eine Liste, bei welcher die Elemente geordnet sind. Durch den Tag `rdf:seq` wird eine Sequenz angegeben. Eine Multimenge ist wie eine Liste, aber mit dem Unterschied, dass es hier keine Rolle spielt, ob die Elemente geordnet sind oder nicht. Die Elemente können auch mehrfach vorkommen. Durch den Tag `rdf:Bag` wird eine Multimenge angegeben. Eine Alternative lässt die Auswahl eines ihrer Elemente zu. Mindestens ein Element muss in einer Alternative gespeichert sein und die Anfangs- und die Endmarke müssen zusammen passen. Durch den Tag `rdf:Alt` wird eine Alternative angegeben. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

Optional besitzen die Container-Elemente ein `id` Attribut, mit welchem der Container identifiziert und referenziert werden kann. Ein Containerinhalt besteht aus `li` Elementen. Wie auch beim Eigenschaftselement gibt es auch beim `li` Element unterschiedliche Arten. Eine dieser Arten besitzt nur ein `resource` Attribut, welches mit einer URI auf eine andere definierte Ressource verweist. Die andere Art enthält einen String, eine Beschreibung oder einen weiteren Container. Im Listing 4.5 ist ein Beispiel für die Container-Elemente `rdf:Alt` und `rdf:Bag` sowie `rdf:seq` dargestellt. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

```

1 <rdf:Alt>
2   <rdf:li rdf:resource="949352" />
3   <rdf:li rdf:resource="949318" />
4 </rdf:Alt>
5
6 <rdf:Bag rdf:ID="DBcourses">
7   <rdf:li rdf:resource="CIT1111" />
8   <rdf:li rdf:resource="CIT3112" />
9 </rdf:Bag>
10
11 <rdf:Seq>
12   <rdf:li>Homer Simpson</>
13   <rdf:li>Marge Simpson</>
14 </rdf:Seq>

```

**Listing 4.5:** Verwendung von Container-Elementen (Quelle: Harmelen und Antoniou, 2004, S. 77)

### 4.2.3 Aussagen modellieren

Öfters möchte man Aussagen über Aussagen treffen. Über den Typ `rdf:statement` wird dies erreicht. Mit den Elementen `subject`, `predicate`, `object` und `type` kann man einfache Satzstrukturen nachbilden. Dabei entspricht das `subject` Element der Ressource, das `predicate` Element der Eigenschaft, das `object` Element dem Eigenschaftswert und das `type` Element ist das Statement. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004)

Ein Beispiel hierfür wäre die folgende Beschreibung:

```

<rdf:Description rdf:about="949352">
  <uni:name>Homer Simpson</uni:name>
</rdf:Description>

```

Diese kann modelliert werden zu:

```

<rdf:Statement rdf:about="StatementAbout949352">
<rdf:subject rdf:resource="949352"/>
<rdf:predicate rdf:resource="&uni;name"/>
<rdf:object>Homer Simpson</rdf:object>
</rdf:Statement>

```

### 4.3 RDF Schema

RDF Schema<sup>9</sup> ist eine Sprache zur Definition von Metadaten und einfachen Ontologien. RDF bietet „nur“ die Möglichkeit, einfache Aussagen in Bezug auf Ressourcen zu machen. Durch RDF Schema werden Klassen und Eigenschaften festgelegt, durch welche es dem bzw. der AnwenderIn möglich ist, eine Anwendung einheitlich zu spezifizieren. Im Anhang A befindet sich ein Überblick über die Klassen und Eigenschaften. RDF Schema gibt aber nicht vor, welche Klassen, Subklassen und Eigenschaften für welche Vokabulare definiert werden. Aus dem RDF Modell und der zugehörigen Syntax erhält man Dokumente, welche sich an zusätzliche Regeln bezüglich der Struktur und Benennung von Elementen und Attributen halten. Durch eine geschickte Namenswahl erhält man hier zwar hilfreiche Informationen, aber für die automatische Weiterverarbeitung ist dies noch zu wenig. (Eckstein und Eckstein, 2003; Harmelen und Antoniou, 2004; Birkenbihl, 2006)

Klasse-Unterklasse-Beziehungen und Eigenschaft-Untereigenschaft-Beziehungen sind die wesentlichen Konzepte von RDF Schema. Bei der Klasse-Unterklasse-Beziehung werden in Klassen Objekte mit den gleichen Eigenschaften zusammengefasst und bestimmen andererseits Begriffe. Unter einer Unterklasse versteht man eine spezielle Untermenge einer Oberklasse. Entsprechendes gilt für die Eigenschaft- Untereigenschaft-Beziehung. (Eckstein und Eckstein, 2003)

Durch RDF Schema können Begriffe semantisch zu einander in Beziehung gesetzt werden. Des Weiteren ist es möglich, von einem Objekt weitere RDF Klassen und die Definition einer Klasse von einer Subklasse abzuleiten. Man kann auch Klassen Eigenschaften, Werte-, und Gültigkeitsbereiche zuordnen. Im Gegensatz zu einem RDF Dokument muss ein RDF Schema Dokument die Namensraumdeklaration für ein RDF Schema enthalten. `rdf` bezeichnet den RDF Namensraum und `rdfs` bezeichnet den Namensraum von RDF Schema. (Eckstein und Eckstein, 2003)

#### 4.3.1 Klassen

RDF Schema stellt die drei zentrale Klassen `rdfs:Resource`, `rdfs:Class` und `rdf:Property` zur Verfügung. (W3C, Brickley und Guha, 2004; Eckstein und Eckstein, 2003)

---

9. Abgekürzt auch „RDFS“

Ressourcen können gruppiert werden in so genannte „Klassen“, welche mit Hilfe von `rdfs:Class` abgebildet werden. Unter einer „Instanz“ versteht man ein Mitglied einer Klasse. Bei RDF wird zwischen Klassen und einem Set von Instanzen unterschieden. Jede Klasse, die in RDF gebildet wird, ist eine Instanz der Klasse `rdfs:Class`. (W3C u. a., 2004; Eckstein und Eckstein, 2003)

Alles was in RDF beschrieben ist, wird „Ressource“ genannt und ist eine Instanz der Klasse `rdfs:Resource`. Alle anderen Klassen sind Subklassen von der Klasse `rdfs:Resource`, welche eine Instanz der Klasse `rdfs:Class` ist. (W3C u. a., 2004; Eckstein und Eckstein, 2003)

Die Eigenschaft `rdf:Property` ist eine Teilmenge von `rdfs:Resource`. Jede Eigenschaft, welche in RDF gebildet ist, ist eine Instanz der Klasse `rdf:Property`. (W3C u. a., 2004; Eckstein und Eckstein, 2003)

Wenn in RDF Schema eine Ressource als Klasse definiert wird, dann kann diese in einem RDF Dokument als Typ in der Form von einem Typelement und `type` Attributen verwendet werden. Des Weiteren können Ressourcen, welche als Eigenschaften in RDF Schema definiert wurden, in einem RDF Dokument in Form von Eigenschaftselementen und Eigenschaftsattributen verwendet werden. (Eckstein und Eckstein, 2003)

### 4.3.2 Eigenschaften

RDF Schema stellt zentrale Eigenschaften zur Verfügung, welche `rdf:type` sowie `rdfs:subClassOf` und `rdfs:subPropertyOf` sind. So genannte „Utility Properties“<sup>10</sup> sind die Eigenschaften `rdfs:seeAlso`, `rdfs:isDefinedBy` sowie die Eigenschaft `rdfs:comment` und `rdfs:label`. Für Einschränkungen werden die Klassen `rdfs:range` und `rdfs:domain` zur Verfügung gestellt. Alle diese Eigenschaften sind Instanzen der Klasse `rdfs:Property`. (Harmelen und Antoniou, 2004; W3C u. a., 2004; Eckstein und Eckstein, 2003)

Mit der Eigenschaft `rdf:type` kann nur eine Ressource als Typ angegeben werden, welche als Klasse definiert ist. Wenn kein Typ angegeben ist, dann wird standardmäßig die Klasse `Resource` angenommen, da dies die allgemeinste Klasse ist. (Harmelen und Antoniou, 2004; W3C u. a., 2004; Eckstein und Eckstein, 2003)

`rdfs:subClassOf` ist eine Eigenschaft, mit welcher eine neue Klasse zu einer Unterklasse wird. Diese neue Klasse wird einer bestehenden Klassenhierarchie hinzugefügt. Diese Eigenschaft dürfen nur Instanzen der Klasse `rdfs:class` besitzen. Eine Klasse darf eine Unterklasse von sich selbst bzw. von ihren eigenen Unterklassen sein. Hiermit kann die Äquivalenz von Begriffen spezifiziert werden. Ein Beispiel für die Eigenschaft ist `A rdfs:subClassOf B`. Diese Eigenschaft ist transitiv, d.h. wenn B

---

10. Hilfseigenschaften

eine Unterklasse von A ist und C eine Unterklasse von B ist, dann ist C auch eine Unterklasse von A. (Harmelen und Antoniou, 2004; W3C u. a., 2004; Eckstein und Eckstein, 2003)

Das Grundprinzip der Eigenschaft `rdfs:subPropertyOf` entspricht dem von der Eigenschaft `rdfs:subClassOf` mit dem Unterschied, dass es sich hierbei um Eigenschaften handelt und nicht um Klassen. Diese Eigenschaft ist ebenfalls transitiv. (Harmelen und Antoniou, 2004; W3C u. a., 2004; Eckstein und Eckstein, 2003)

Mit der Eigenschaft `rdfs:seeAlso` können zusätzliche Informationen zu einer Ressource angegeben werden. `rdfs:isDefinedBy` ist eine Eigenschaft, durch welche auf eine definierende Ressource verwiesen werden kann. Diese Eigenschaft sowie die Eigenschaft `rdfs:seeAlso` kann bei jeder Ressourcen verwendet werden und kann als Wert eine beliebige Ressource besitzen. (Harmelen und Antoniou, 2004; W3C u. a., 2004; Eckstein und Eckstein, 2003)

Definierte Klassen und Eigenschaften können in ihrer Verwendbarkeit eingeschränkt werden. Dafür gibt es die Eigenschaften `rdfs:domain` und `rdfs:range`. Die Eigenschaft `rdfs:domain` wird verwendet um festzulegen, dass Ressourcen mit eingeschränkter Eigenschaft beschrieben werden dürfen, die zu der angegebenen Domäne gehören. Durch die Eigenschaft `rdfs:range` werden die Werte auf die Objekte der angegebenen Klasse eingeschränkt. (Harmelen und Antoniou, 2004; W3C u. a., 2004; Eckstein und Eckstein, 2003)

Durch die Eigenschaft `rdfs:label` kann ein für Menschen intuitiv verständlicher Name einer Ressource zur Verfügung gestellt werden. Dies kann z.B. für den Namen eines Knoten in einer grafischen Darstellung eines RDF Dokumentes verwendet werden. (Harmelen und Antoniou, 2004; W3C u. a., 2004)

`rdfs:comment` ist eine Eigenschaft, durch welche es möglich ist, dass eine verbale Beschreibung zu einer Ressource oder zu einem Konzept erstellt werden kann. Normalerweise sind dies längere Texte. (Harmelen und Antoniou, 2004; W3C u. a., 2004; Eckstein und Eckstein, 2003)

Die Abbildung 4.3 zeigt einen Ausschnitt aus der semantischen Definition von RDF Schema in RDF Schema.

## 4.4 Query Sprachen für RDF

XML ist eine Sprache, welche auf einem niedrigeren Abstraktionslevel arbeitet als RDF. Deshalb werden andere „Query Sprachen“ für RDF benötigt. Eine geeignete Query Sprache muss nicht nur die RDF Syntax verstehen, sondern auch das RDF Datenmodell, und die Semantik des RDF Vokabulars. Des Weiteren sollte eine Query Sprache



auch die Semantik des RDF Diagramms verstehen. Einige der Query Sprachen werden in diesem Abschnitt beschrieben. (Harmelen und Antoniou, 2004)

#### 4.4.1 RQL (RDF Query Language)

Eine der Query Sprachen für RDF ist „RDF Query Language“<sup>11</sup>. `Class` ist eine Basis Query von RQL, mit welcher alle Klassen abgefragt werden können. Eine andere Basis Query ist `Property`, mit welcher alle Eigenschaften abgefragt werden können. Um die Instanzen einer Klasse, z.B. der Klasse `course`, abzufragen wird die Query `course` verwendet. (Harmelen und Antoniou, 2004)

Die Query `course` retourniert auch alle Instanzen der Subklassen von `course`. Wenn man aber die geerbten Instanzen nicht im Ergebnis haben möchte, so muss die Query `^course` verwendet werden. (Harmelen und Antoniou, 2004)

Um die Eigenschaften abzufragen, z.B. von `involves`, wird die Query „`involves`“ verwendet. Das Ergebnis von dieser Query enthält auch alle Subeigenschaften von `involves`. Wenn man die geerbten Eigenschaften nicht im Ergebnis haben möchte, so muss die Query „`^involves`“ stattdessen verwendet werden. (Harmelen und Antoniou, 2004)

Es gibt auch die Möglichkeit Daten über eine „Select-From-Where“ Query abzufragen. Wobei `Select` die Anzahl und die Anordnung der abgefragten Daten spezifiziert. Die `From` Klausel wird verwendet um durch das Datenmodell zu navigieren. In der `Where` Klausel werden die Einschränkungen der Query angegeben. Ein Beispiel hierfür ist folgende Query. (Harmelen und Antoniou, 2004)

```
SELECT X, Y
FROM {X}phone{Y}
```

Durch diese Query können alle Telefonnummern abgefragt werden. `X` und `Y` sind hierbei Variablen und durch `{X}phone{Y}` wird ein „Ressource-Eigenschaft-Wert“ Triple repräsentiert. Um alle Telefonnummer zu allen ProfessorenInnen zu erhalten, wird folgende Query verwendet. (Harmelen und Antoniou, 2004)

```
SELECT X, Y
FROM lecturer{X}.phone{Y}
```

Bei dieser Query werden in `lecturer{X}` alle Instanzen der Klasse `lecturer` gesammelt und in der Variable `{X}` gespeichert. Im zweiten Teil `phone{Y}` werden alle Tripel mit dem Prädikat `phone` gespeichert. Bei dieser Query wird ein „impliziter

---

11. Im Folgenden weiter RQL genannt

Join“ verwendet, welcher durch den Punkt (.) angezeigt wird. Durch diesen Join wird die zweite Abfrage auf die Triple eingeschränkt, welche bereits in der Variablen X gespeichert sind. Einen „expliziten Join“ wird durch die folgende Query dargestellt. Diese liefert die Namen von allen Kursen, bei welchen der bzw. die Vortragende mit der ID 949352 unterrichtet. (Harmelen und Antoniou, 2004)

```
SELECT N
FROM course{X}.isTaughtBy{Y}, {C}name{N}
WHERE Y="949352" and X=C
```

Mit RQL ist es auch möglich Schema Informationen abzufragen. Dabei haben Schema Variablen für Klassen den Präfix \$ und für Eigenschaften den Präfix @. Ein Beispiel hierfür ist folgende Query. (Harmelen und Antoniou, 2004)

```
SELECT X, $X, Y, $Y
FROM {X:$X}phone{Y:$Y}
```

Diese Query liefert als Ergebnis alle Ressourcen und Werte von Tripel mit der Eigenschaft phone, oder mit einigen Subeigenschaften und Klassen. (Harmelen und Antoniou, 2004)

#### 4.4.2 RDQL (RDF Data Query Language)

Eine weitere Query Sprache für RDF ist „RDF Data Query Language“<sup>12</sup>. Diese Sprache ist SQL<sup>13</sup> sehr ähnlich, da diese Sprache auch „Select-From-Where“ Queries verwendet. (Badertscher, 2006)

RDQL besteht aus einem Mustergraphen, welcher mit einer Liste bestehend aus Tripel Mustern ausgedrückt wird. Jedes dieser Tripel Muster besteht aus Variablen, URIs und Literalen<sup>14</sup>. Eine Query kann Einschränkungen in Bezug auf die Werte der Variablen und auf das Ergebnis haben. Ein Beispiel für eine RDQL Query ist die folgende. (Seaborne, 2004)

```
SELECT ?x
WHERE (?x,
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,
<http://example.com/someType>)
```

12. Im Folgenden weiter RDQL genannt

13. Structured Query Language

14. Eine Zeichenkette, welche der Wert sein kann

Durch dieses Tripel Muster werden alle Aussagen im Graphen erfasst, welche das Prädikat `<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>` und das Objekt `<http://example.com/someType>` haben. `?x` zeigt eine Variable an, dabei ist `x` der Name der Variablen und durch `?` wird angegeben, dass es sich um eine Variable handelt. Eine etwas komplexere Query zeigt das folgende Beispiel. (Seaborne, 2004)

```
SELECT ?resource
FROM   <http://example.org/someWebPage>
WHERE  (?resource info:age ?age)
AND    ?age >= 24
USING  info FOR <http://example.org/peopleInfo#>
```

Wenn mehrere Argumente in der „Select“ Klausel verwendet werden sollen, dann werden diese durch einen Beistrich (,) von einander getrennt. Bei dieser Query werden auch Einschränkungen durch die „Where“ Klausel verwendet. Hier können mehrere Einschränkungen verwendet werden, welche dann durch das Schlüsselwort AND miteinander verknüpft werden. (Seaborne, 2004; Badertscher, 2006)

Bei einer RDQL Query wird ein RDF Graph so behandelt als ob es „nur“ Daten sind. Bei RDQL wird keine Unterscheidung zwischen einem berechneten Tripel und einem Basis Tripel gemacht. Wenn bei einer Query Transitivität gewünscht wird, so muss dies per Hand in die Query integriert werden, da eine explizite Transitivität nicht unterstützt wird. Eine RDQL Anwendung ist das Jena Framework, welches im Kapitel 5 beschrieben wird. Weitere Anwendungen sind unter anderem „RDFStore“<sup>15</sup>, „Sesame“<sup>16</sup>, „3Store“<sup>17</sup>, usw. (Seaborne, 2004; Badertscher, 2006)

#### 4.4.3 SPARQL (SPARQL Protocol and RDF Query Language)

Eine andere Query Sprache für RDF ist „SPARQL“. Diese Sprache basiert auf dem Vergleichen von Graphen Mustern. Dabei enthalten diese Graphen Muster Tripel, welche ähnlich den RDF Tripel sind. Durch die Kombination von Tripel Mustern enthält man ein Basis Graph Muster, bei welchen ein genauer Vergleich mit einem Graphen erforderlich ist. Die Queries können auch auf RDF Literalen durchgeführt werden. Eine Query besteht hierbei aus einer „Select“ Klausel und einer „Where“ Klausel. In der „Select“ Klausel werden die Variablen angegeben, welche im Ergebnis enthalten sind und in der

15. <http://rdfstore.sourceforge.net/> - Zugriffsdatum: 17.12.2007

16. <http://www.openrdf.org/> - Zugriffsdatum: 17.12.2007

17. <http://sourceforge.net/projects/threestore/> - Zugriffsdatum: 17.12.2007

„Where“ Klausel stellt das Basis Graphen Muster zur Verfügung, welches mit dem Daten Graphen verglichen wird. Die nachfolgende Query zeigt wie ein Buchtitel von einem gegebenen Datengraphen gefunden werden kann. Das Basis Graph Muster enthält hier ein einfaches Tripel Muster mit der einzelnen Variablen `?title`. (Prud'hommeaux und Seaborne, 2007)

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1>
  <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Dies Query liefert genau ein Ergebnis als Resultat zurück. Aber eine Query kann kein, ein oder mehrere Ergebnisse zurückliefern. Ein Beispiel für eine Query, welches mehrere Ergebnisse enthält ist das nachfolgende. (Prud'hommeaux und Seaborne, 2007)

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
{
  ?x foaf:name ?name .
  ?x foaf:mbox ?mbox
}
```

Diese Query liefert die Namen und die E-Mail Adressen. Über PREFIX wird angegeben, auf welchen Daten die Query durchgeführt wird. Bei diesem Beispiel werden nachfolgenden Daten verwendet. (Prud'hommeaux und Seaborne, 2007)

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Charles Montgomery Burns" .
_:a foaf:mbox <mailto:cmb@example.com> .
_:b foaf:name "Waylon Smithers" .
_:b foaf:mbox <mailto:waylon@example.org> .
```

SPARQL unterstützt vier verschiedene Formen von Queries. Eine davon ist das „Select“ Query, welches alle oder eine Teilmenge der Variablen, welche in das Query Muster passen, liefert. Das „Construct“ Query hat einen RDF Graphen als Ergebnis, welcher durch Substituieren von Variablen in einer Menge von Tripel Templates konstruiert wurde.

Einen RDF Graphen, welcher die gefundenen Ressourcen beschreibt, liefert das „Describe“ Query. Das „Ask“ Query hat als Ergebnis wahr oder falsch, dadurch wird angezeigt, ob ein Query Muster passt oder ob es nicht passt. (Prud'hommeaux und Seaborne, 2007)

#### 4.4.4 OWL-QL (Ontology Web Language Query Language)

Eine Query Sprache für RDF ist „Ontology Web Language Query Language“<sup>18</sup>, welche eine Weiterentwicklung der Sprache „DQL“<sup>19</sup> ist. Hiermit soll eine generische Query Sprache für das Semantic Web zur Verfügung gestellt werden. Diese Sprache ist ein Protokoll, welches die Kommunikation zwischen dem Webserver und dem Client beschreibt, was eine Besonderheit unter den Query Sprachen ist. Das ist notwendig damit die Anfrage-Antwort Dialoge realisiert werden können. Es wird hierbei eine semantische Beziehung zwischen einer Frage, der dazupassenden Antwort und der verwendeten Wissensbasis bzw. der Wissensbasen genau festgelegt. (Badertscher, 2006; Wichmann, 2007; Fikes, McCool und McGuinness, 2005)

Indem ein OWL-QL basierender Agent eine Frage an einen anderen OWL-QL basierenden Agenten sendet, kann ein Frage-Antwort Dialog ausgelöst werden. Eine Frage ist ein Objekt, welches unbedingt ein Query Muster enthält. Optional besteht die Möglichkeit durch ein Antwort Muster das Format der Antwort festzulegen. Zu einer Frage können keine, eine oder mehrere Antworten vorhanden sein. Diese Sprache wurde zwar für OWL spezifiziert, aber es besteht die Möglichkeit, dass diese Query Sprache auch für RDF und „DAML+OIL“ angepasst werden kann. (Wichmann, 2007; Fikes u. a., 2005)

### 4.5 Verwendung von RDF

Da RDF zur Verarbeitung von Metadaten eingesetzt wird, gibt es verschiedene Verwendungsmöglichkeiten hierfür. In diesem Abschnitt werden einige dieser Verwendungsmöglichkeiten beschrieben.

#### 4.5.1 World Wide Web (WWW) & Suchmaschinen

Im World Wide Web (WWW) ist eine unvorstellbar große Masse an Information vorhanden. Das Problem hierbei ist aber das Finden der „richtigen“ Informationen. Hierfür gibt es natürlich viele Suchmaschinen, welche dabei helfen. Allerdings ist es nicht immer leicht die Informationen zu finden, welche man gerade sucht. Sehr oft werden

---

18. Im Folgenden weiter OWL-QL genannt

19. DAML Query Language

hierbei auch Informationen gefunden und angezeigt, welche nur bedingt mit dem gewünschten Ergebnis zu tun haben. Dies hängt damit zusammen, dass Suchmaschinen sehr einfach parametrisiert sind. Dies kann durch RDF verbessert werden. Dafür ist es notwendig das HTML<sup>20</sup> Seiten, genauer gesagt der HTML Header, mit RDF Daten annotiert werden. Zurzeit würde dies am Suchverfahren selbst nichts ändern, aber die wichtigsten Informationen würden sich am Anfang des Dokumentes befinden. Dadurch können Suchmaschinen eine hohe Wertigkeit vergeben, wenn gerade diese Informationen mit dem Suchbegriff zusammenpassen. Es würden hier Metadaten für die Suchmaschinen verwendet werden. (Eckstein und Eckstein, 2003)

#### **4.5.2 Semantisch unterstützte Suchmaschinen**

In naher Zukunft können Suchmaschinen auch auf semantischen Definitionen Bezug nehmen, besonders auf das RDF Schema mit den beschreibenden Begriffshierarchien. Durch diese Begriffshierarchien werden Beziehungen zwischen den Begriffen beschrieben. Dadurch werden auch Seiten gefunden, welche nicht direkt den gesuchten Begriff enthalten, sondern welche Begriffe enthalten, die zum gesuchten Begriff in Beziehung stehen. (Eckstein und Eckstein, 2003)

#### **4.5.3 Zentralisierte Beschreibungen spezieller Ressourcen**

Beschreibungen können für spezielle Ressourcen, wie z.B. Literatur, Messdaten einer Anwendungsdomäne oder Software, auf einem zentralen Server abgelegt werden. Dadurch entsteht die Möglichkeit, dass die Beschreibungen und die Begriffshierarchien so gespeichert werden, dass man diese effizient abfragen kann. Dadurch kann man in den Begriffshierarchien selbst suchen um Begriffe zu finden. Andererseits steht hier dann auch eine Schlüsselwortsuche zur Verfügung, durch welche mit verwandten Begriffen gearbeitet werden kann. (Eckstein und Eckstein, 2003)

#### **4.5.4 Web-Portale**

Eckstein und Eckstein (2003) beschreiben, dass über ein Web-Portal verschiedene Informationen bzw. Inhalte zu einem bestimmten Thema präsentiert werden. Im Bereich des E-Commerce bzw. des E-Business können Web-Portale eine Vermittlerfunktion zwischen:

- HändlerInnen
- HerstellerInnen bzw. Zulieferfirmen
- KäuferInnen bzw. Unternehmen

---

20. Hypertext Markup Language

übernehmen. Hier werden verschiedene Informationen gesammelt und verschiedene Services wie z.B. Rating Services zur Verfügung gestellt. Einfache Indizes und Meta-Tags in HTML sind hier bei weitem nicht mehr ausreichend. Um diese Funktionalität zu bieten, werden semantische Technologien benötigt. (Eckstein und Eckstein, 2003)

## 4.6 Entwicklungen / Anwendungen

In diesem Abschnitt werden RDF basierende Entwicklungen bzw. Anwendungen vorgestellt.

### 4.6.1 Dublin Core Metadata Initiative (DCMI)

Die „Dublin Core Metadata Initiative“<sup>21</sup> hat die Entwicklung von RDF sehr stark beeinflusst und gehört auch zu den bekanntesten Entwicklungen auf dem Gebiet der Metadaten. Die Entwicklung von DCMI begann im Jahre 1995. Das Ziel ist die Entwicklung und Verbreitung von interoperablen domänenunabhängigen Metadaten-Vokabularen, damit man Ressourcen so beschreiben kann, dass intelligentere Suchmaschinen entwickelt werden können. Zu Beginn wurde der Dublin-Core-Standard unabhängig von RDF entwickelt, da aber die EntwicklerInnen des Dublin-Core-Standards auch bei RDF mitgearbeitet haben, entstand in weiterer Folge auch ein Dublin-Core-Standard in RDF. (Eckstein und Eckstein, 2003; DCMI, 2007)

#### 4.6.1.1 Dublin Core Vokabular

Es werden hier eine Menge von Haupteigenschaften, so genannte Elemente, definiert, durch welche Dokumenten, Datenquellen oder ähnliches beschrieben werden sollen. Daraus erhält man die grundlegenden Metadaten. Zusätzlich besteht die Möglichkeit, dass einige Elemente weiter verfeinert werden können, dadurch wird die Beschreibung auch exakter. Des Weiteren werden zu einigen Eigenschaften auch ein Kodierungsschemata angegeben. Dadurch werden die erlaubten Werte der Eigenschaft eingeschränkt bzw. können die Werte semantisch eingeordnet werden. Es werden 15 Eigenschaften im „Element Set“ für die Beschreibung von Ressourcen, besonders für Web Ressourcen, behandelt. Ein Überblick über diese Eigenschaften findet sich im Anhang B. (Eckstein und Eckstein, 2003; DCMI, 2007)

---

21. Im Folgenden weiter DCMI genannt

#### 4.6.1.2 Dublin Core in RDF

Der Dublin Core in RDF ist geteilt in den einfachen und den qualifizierten Dublin Core. Beim einfachen Dublin Core sieht die DTD<sup>22</sup> für das RDF nur vor, dass die „Dublin-Core-Elemente“ in beliebiger Reihenfolge beliebig oft innerhalb des Elementes `rdf:Description` vorkommen dürfen. Dabei dürfen aber die „Dublin-Core-Elemente“ keine weiteren Elemente enthalten, es dürfen aber die Attribute `xml:lang` und `rdf:resource` enthalten sein. (Eckstein und Eckstein, 2003)

Beim qualifizierten Dublin Core werden die zwei Qualifizierungsarten „Element - Verfeinerung“ und „Kodierungsschemata“ unterschieden, welche vollständig über das RDF-Schema entwickelt werden. Bei einer „Element-Verfeinerung“ wird z.B. das Element `description` als Eigenschaft mit der `rdfs:subPropertyOf` Eigenschaft verfeinert zu `abstract`. Beim „Kodierungsschemata“ sind die RDF Syntax bzw. Schema Elemente enthalten. Des Weiteren steht hier auch das RDF-Container Konzept zur Verfügung. Dadurch ist es möglich ein RDF Dokument gut mit den Mitteln von Dublin Core auszudrücken. (Eckstein und Eckstein, 2003)

#### 4.6.2 Composite Capability/Preference Profiles (CC/PP)

„Composite Capability/Preference Profiles“<sup>23</sup> wird verwendet um die Funktionen von Clients sowie die Präferenzen des bzw. der BenutzerIn zu beschreiben. Die Entwicklung von mobilen Clients wie z.B. PDAs<sup>24</sup>, führte zu neuen Anforderungen für Applikationen im World Wide Web (WWW). Da auch bei diesen mobilen Clients die Fähigkeiten und die Präferenzen des bzw. der BenutzerIn beschrieben werden sollen, wurde das auf RDF basierende Metadatenformat CC/PP spezifiziert. Dabei wird RDF für die Erzeugung der Profile verwendet. (Eckstein und Eckstein, 2003; Klyne, Reynolds, Woodrow, Ohto, Hjelm und Tran, 2004)

Um die gewünschte Form einer Ressource zu bestimmen, welche an den Client übertragen werden sollen, enthält ein CC/PP Profil eine Anzahl von Attribut-Wert-Paaren. Bei CC/PP werden die Ressourcenarten „Eigenschaften“ und „Literele“ unterschieden. Eine Ressource bei CC/PP ist eine Unterklasse von `rdfs:Resource`, welche wiederum in `Profile`, `Proxy-profile`, `Proxy-behavior` und `Component` unterteilt ist. Wobei ein `Profile` ein `Request-profile` oder ein `Client-profile` sein kann. (Eckstein und Eckstein, 2003; Klyne u. a., 2004)

Ein Client wird genauer durch die Komponenten „Hardware-Plattform“, „Software-Plattform“ und „Browser-User-Agent“ beschrieben. Angaben zum Betriebssystem, zu

---

22. Document Type Definition

23. Im Folgenden weiter CC/PP genannt

24. Personal Digital Assistants

der vorhandenen und zu der ladbaren Software kann durch die Komponenten „Software-Plattform“ gemacht werden. Für die Komponente „Hardware-Plattform“ können Angaben zu den verschiedenen Parametern des Displays und der sonstigen Konfigurationen gemacht werden. Durch die Komponente „Browser-User-Agent“ werden verschiedene Informationen, wie z.B. der Name, die Version, usw., über den Browser auf dem Client angegeben. Zu allen Komponenten existieren Voreinstellungen, auf welche Bezug genommen werden kann.

Ein CC/PP Profil enthält ein oder mehrere Komponenten mit einem oder mehreren Attributen. CC/PP besitzt ein Attribut-Vokabular für Druck und Anzeige, welche die Attribute der Klasse `component` sind mit der Ausnahme des Attributes `defaults`. Durch die Attribute der Komponenten werden die Client Fähigkeiten und die Präferenzen des bzw. der BenutzerIn beschrieben. Diese Attribute besitzen als Wertebereich Literale. Es existieren dabei URIs, Texte, ganze und rationale Zahlen. Das Vokabular von CC/PP ist nicht auf spezielle Anwendungsdomänen ausgerichtet. (Eckstein und Eckstein, 2003; Klyne u. a., 2004)

Es können aber nicht nur die Client Komponenten beschrieben werden, sondern auch die Proxies<sup>25</sup> des Clients. Über einen Proxy können die Inhalte, welche zu einem Client übermittelt werden, gefiltert oder auch modifiziert werden. CC/PP Client Informationen können hierbei die Proxy Informationen erweitern. Wenn mehrere Proxies hinter einander bis zum Client durchlaufen werden, dann bilden sie eine Kette. Hier wird dann mit dem Anfangsprofil `Request-profile` begonnen, welches die zwei Eigenschaften `nextProfile` und `proxyProfile` enthält. Solange Proxies in der Kette vorhanden sind, ist das nächste Profil ein Anfrageprofil. Wenn die Kette dann mit dem Client beendet wird, dann wird das Clientprofil `Client-profil` verwendet, welches als Eigenschaft `Component` besitzt, in welchen dann die Komponenten des Clients spezifiziert werden. (Eckstein und Eckstein, 2003; Klyne u. a., 2004)

---

25. Darunter versteht man Zwischenknoten innerhalb eines Netzwerks zwischen Client und Server, welche bei jeder Verbindung verwendet werden.

## Kapitel 5

### Jena Framework

Das „Jena Framework“<sup>1</sup> wurde von den „Hewlett-Packard Labs“ während des „Semantic Web Research“ entwickelt. Jena ist eine Java API<sup>2</sup> für Semantic Web Anwendungen. Es werden hierbei Klassen und Interfaces<sup>3</sup> bereitgestellt, durch welche Ressourcen, Eigenschaften, Klassen und alle weiteren Elemente von RDF, RDF Schema und OWL erzeugt werden können. (Hewlett-Packard Development Company, 2007b)

Im ersten Abschnitt werden allgemeine Themen von Jena, wie z.B. die Verwendung von Datenbanken, beschrieben. Mit der Verwendung der RDF API beschäftigt sich der zweite Abschnitt. In den letzten zwei Abschnitten wird dann die „SPARQL“ Query Engine und die Verwendung der Ontologie API eingegangen. An den passenden Stellen werden auch Codefragmente für eine bessere Verständlichkeit angegeben.

#### 5.1 Allgemeines

Im Jena Framework sind folgende „Features“ inkludiert:

- Eine RDF API
- Lesen und Schreiben von RDF in RDF/XML, N3<sup>4</sup> und N-Triples
- Eine OWL API
- „In-memory“ und persistentes Speichern
- Eine SPARQL Query Engine

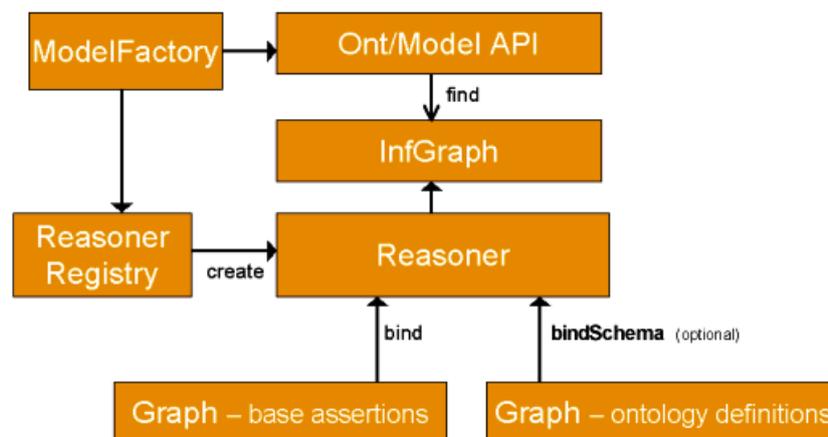
(Hewlett-Packard Development Company, 2007b)

- 
1. Im Folgenden weiter Jena genannt
  2. Application Programming Interface - Schnittstelle zur Anwendungsprogrammierung
  3. Schnittstellen
  4. Auch Notation 3 genannt

N3 sowie N-Triples sind Formate für RDF Daten. N3 ist eine Sprache, welche eine kompakte und lesbare Alternative zur XML Syntax von RDF ist. Durch diese Sprache ist auch eine größere Ausdrucksfähigkeit gegeben. N-Triples ist ein Format für die Verschlüsselung von RDF Graphen. (Berners-Lee, 2006; Grant und Beckett, 2004)

Es können auch verschiedene relationale Datenbanken für das persistente Speichern von RDF Daten verwendet werden. Dabei sind Implementierungen für die Datenbanken „HSQLDB“<sup>5</sup>, „MySQL“<sup>6</sup>, „PostgreSQL“<sup>7</sup>, „Apache Derby“<sup>8</sup>, „Oracle“<sup>9</sup> und „Microsoft SQL Server“<sup>10</sup> verfügbar. (Hewlett-Packard Development Company, 2007b)

Das Inferenz Subsystem von Jena wurde so entworfen, dass es mehreren Inferenz Engines erlaubt, sich mit Jena zu verbinden. Solche Engines werden dazu verwendet, dass man zusätzliche RDF Behauptungen von einem Grundgraphen ableiten kann. Die Hauptverwendung dieses Mechanismus ist die Unterstützung der Verwendung von Sprachen wie RDFS und OWL, welche es zusätzlichen Fakten erlauben aus Beispieldaten und Klassenbeschreibungen abgeleitet zu werden. Abbildung 5.1 zeigt den Aufbau des Inferenzsystems. (Reynolds, 2007)



**Abbildung 5.1:** Aufbau des Inferenzsystems (Quelle: Reynolds, 2007).

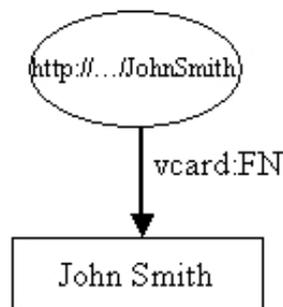
5. <http://www.hsqldb.org/> - Zugriffsdatum: 17.12.2007
6. <http://www.mysql.com/> - Zugriffsdatum: 17.12.2007
7. <http://www.postgresql.org/> - Zugriffsdatum: 17.12.2007
8. <http://db.apache.org/derby/> - Zugriffsdatum: 17.12.2007
9. <http://www.oracle.com/index.html> - Zugriffsdatum: 17.12.2007
10. <http://www.microsoft.com/germany/sql/default.msp> - Zugriffsdatum: 17.12.2007

## 5.2 RDF

Jena stellt unter anderem auch eine Programmierumgebung für RDF und RDF Schema zur Verfügung. Die Verwendung von RDF in Jena wird in diesem Abschnitt beschrieben.

### 5.2.1 RDF Graph

Jena kann für die Erstellung und die Bearbeitung von RDF Graphen verwendet werden. Ein einfacher RDF Graph ist in Abbildung 5.2 dargestellt. Diese werden in Jena „Modell“ genannt und sie werden durch das „Model Interface“ dargestellt. (McBride, 2006)



**Abbildung 5.2:** Einfacher RDF Graph (Quelle: McBride, 2006)

Um den Graphen aus der Abbildung 5.2 in Jena zu erzeugen, ist der Code aus dem Listing 5.1 notwendig. (McBride, 2006)

```
1 static String personURI = "http://somewhere/JohnSmith";
2 static String fullName = "John Smith";
3
4 Model model = ModelFactory.createDefaultModel();
5
6 Resource johnSmith = model.createResource(personURI);
7
8 johnSmith.addProperty(VCARD.FN, fullName);
```

**Listing 5.1:** Erzeugung eines Graphen in Jena (Quelle: McBride, 2006)

Durch die ersten zwei Zeilen werden die Konstanten `personURI` und `fullName` definiert. Anschließend wird über die Klasse `ModelFactory` ein leeres Modell erzeugt.

In Jena sind verschiedene Modell Interfaces für unterschiedliche Modells vorhanden, bei diesem Beispiel wird ein „memory-based“ Modell verwendet. Anschließend wird die Ressource `johnSmith` durch die `createResource()` angelegt und eine Eigenschaft wird durch `addProperty()` hinzugefügt. Hierbei wird durch die Konstante `VCARD` angegeben, dass das „VCARD“ Schema verwendet wird. Es werden in Jena konstante Klassen auch für andere bekannte Schemas, wie für RDF und RDF Schema, sowie für Dublin Core und DAML zur Verfügung gestellt. (McBride, 2006)

Es werden auch Methoden für das Lesen und das Schreiben von RDF als XML von Jena zur Verfügung gestellt. Dies ist nützlich um ein RDF Modell in einer Datei abzuspeichern, um später wieder darauf zuzugreifen. Um ein RDF Modell abzuspeichern, steht die Methode `write()` zur Verfügung. Um ein RDF Modell von einer XML Datei zu lesen, ist die Methode `read()` verwendbar. (McBride, 2006)

Das RDF Package `com.hp.hpl.jena.rdf.model` ist besonders wichtig für AnwendungsentwicklerInnen. Dieses Package enthält die Interfaces für die Modelle, die Ressourcen, die Eigenschaften, die Literale, die Aussagen und für alle anderen Schlüsselkonzepte von RDF. Des Weiteren ist hier auch das `ModelFactory` für die Erzeugung eines Modells enthalten. Damit die Applikation unabhängig von der Implementierung bleibt, ist es am besten, wenn überall wo es möglich ist, Interfaces an stelle von spezifischen Klassen verwendet werden. (McBride, 2006)

### 5.2.2 RDF Aussagen

Eine RDF Aussage besteht aus einem „Subjekt“, einem „Prädikat“ und einem „Objekt“. Dies wird oft auch „Tripel“ genannt. Die Methode `listStatements()` wird im Jena Modell Interface definiert, welche einen Iterator über alle Aussagen im Modell zurück liefert. Dieses Interface liefert auch Methoden für „Subjekte“, „Prädikate“ und „Objekte“ von einer Aussage. Eine weitere nützliche Methode ist `getObject`. Diese Methode liefert ein Objekt `RDFNode` zurück, welches eine Superklasse von `Resource` und `Literal` ist. Das Objekt einer Aussage kann sowohl eine `Resource` als auch ein `Literal` sein. Durch die Verwendung der Methode `instanceof` wird festgestellt, ob es sich beim Objekt um eine `Resource` oder um ein `Literal` handelt. Das Listing 5.2 zeigt die Verwendung von `listStatements()`. Die Methode `toString()` wird hierbei für die Ausgabe des Subjektes, des Prädikates und des Objektes verwendet. Um bei der Ausgabe unterscheiden zu können, ob es sich um eine Ressource oder um ein Literal handelt, wird ein Literal zwischen („“) ausgegeben. (McBride, 2006)

```
1 // Auflisten der Aussagen
2 StmtIterator iter = model.listStatements();
```

```
3
4 while (iter.hasNext()) {
5     // die nächste Aussagen erhalten
6     Statement stmt = iter.nextStatement();
7
8     // das Subjekt erhalten
9     Resource subject = stmt.getSubject();
10
11    // das Prädikat erhalten
12    Property predicate = stmt.getPredicate();
13
14    // das Objekt erhalten
15    RDFNode object = stmt.getObject();
16
17    // Ausgabe des Subjekts, des Prädikates und des Objekts
18    System.out.print(subject.toString() + " ");
19    System.out.print(predicate.toString() + " ");
20
21    if (object instanceof Resource) {
22        // Objekt ist eine Ressource
23        System.out.print(object.toString());
24    } else {
25        // Objekt ist ein Literal
26        System.out.print "\"" + object.toString() + "\"");
27    }
```

**Listing 5.2:** Auflisten von Aussagen (Quelle: McBride, 2006)

### 5.2.3 Navigieren durch ein Modell

Durch die URI einer Ressource kann das Ressource Objekt von dem Modell abgefragt werden. Hierfür wird die Methode `Model.getResource(String uri)` verwendet. Eine Vielzahl von Methoden für den Zugriff auf die Eigenschaften einer Ressource wird im `Resource` Interface definiert. Eine Eigenschaft einer Ressource kann über die Methode `Resource.getProperty(Property p)` abgefragt werden. Bei dieser Methode wird die gesamte Aussage retourniert. Um z.B. den Wert der `VCARD.N` Eigenschaft abzufragen ist der Code aus dem Listing 5.3 notwendig. (McBride, 2006)

```
1 Resource name = ( Resource )
2 vcard . getProperty ( VCARD . N ) . getObject ( ) ;
```

*Listing 5.3:* Wert einer Eigenschaft abfragen (getObject) (Quelle: McBride, 2006)

Eine Aussage kann eine Ressource oder ein Literal sein. Jena bietet hierfür spezifische Funktionen an, so dass keine extra Klassencastings oder Typüberprüfungen durchgeführt werden müssen. Das Listing 5.4 zeigt wie der Wert einer Eigenschaft abgefragt werden kann. Des Weiteren wird die Abfrage für den literalen Wert einer Eigenschaft gezeigt. (McBride, 2006)

```
1 Resource name = vcard . getProperty ( VCARD . N ) . getResource ( ) ;
2
3 String fullName = vcard . getProperty ( VCARD . FN ) . getString ( ) ;
```

*Listing 5.4:* Wert einer Eigenschaft abfragen (getResource) (Quelle: McBride, 2006)

Es besteht die Möglichkeit, dass eine Eigenschaft mehr als einmal vorkommt. Dann kann der Aufruf `Resource.listProperties(Property p)` verwendet werden um einen Iterator zu erhalten. Der Iterator enthält Objekte vom Typ `Statement`. Durch die Methode `Resource.listProperties()` ohne Parameter werden alle Eigenschaften retourniert. Wenn z.B. alle Nicknames<sup>11</sup> ausgegeben werden sollen, so könnte der Code wie im Listing 5.5 aussehen. (McBride, 2006)

```
1 StmtIterator iter = vcard . listProperties ( VCARD . NICKNAME )
```

*Listing 5.5:* Auflisten der Eigenschaften (Quelle: McBride, 2006)

#### 5.2.4 Suchen in einem Modell

In der Kern-API von Jena wird nur eine beschränkte Query Primitive unterstützt. Die Methode `Model.listStatements()`, welche alle Aussagen in einem Modell auflistet, ist eine grobe Form des Suchens innerhalb eines Modells. Das gilt auch für ähnliche Methoden, wie z.B. für die Methode `Model.listSubjectsWithProperty(Property p, RDFNode o)`. (McBride, 2006)

---

11. Spitznamen

Bei der Querymethode `model.listStatements (Selector s)` ist dies hingegen anders. Diese Methode gibt einen Iterator über alle Aussagen zurück, welche sich im Modell `s` befinden. Um eine Aussage mit allen Subjekten, Prädikaten und Objekten zu erhalten, wird der `SimpleSelector(subject, predicate, object)` verwendet. Wird hier bei einem Parameter der Wert `null` übergeben, so passt dies zu allem, z.B. wenn `null` als Parameter von `subject` übergeben wird, so werden alle Subjekte der Modelle zurück geliefert. (McBride, 2006)

### 5.2.5 Operationen auf ein Modell

Um ein Modell als ganzes zu bearbeiten werden in Jena die Operationen „*union*“, „*intersection*“ and „*difference*“ zur Verfügung gestellt. Durch „*union*“ werden zwei Modelle zu einem vereinigt. Es werden hierbei die Aussagen, welche in jedem Modell repräsentiert werden, vereinigt. Die Abbildung 5.3 zeigt zwei getrennte Modelle. Diese Modelle sollen durch die Verwendung der Operation „*union*“ zu einem Modell vereinigt werden. Wie das Modell nach der Durchführung von „*union*“ aussieht wird in Abbildung 5.4 dargestellt. (McBride, 2006)

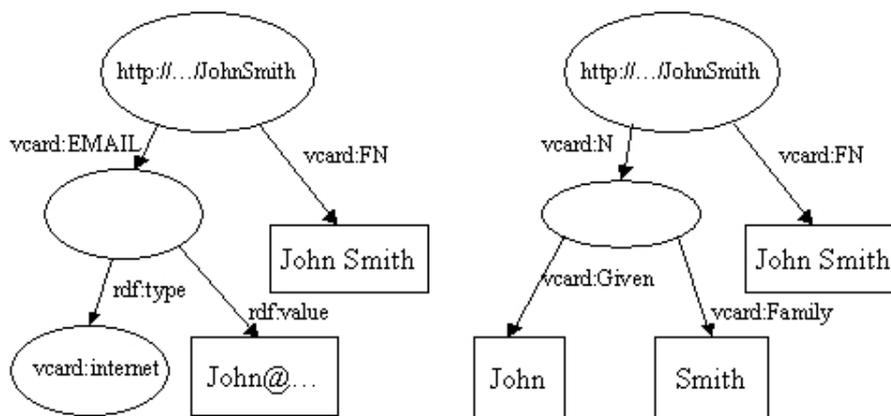
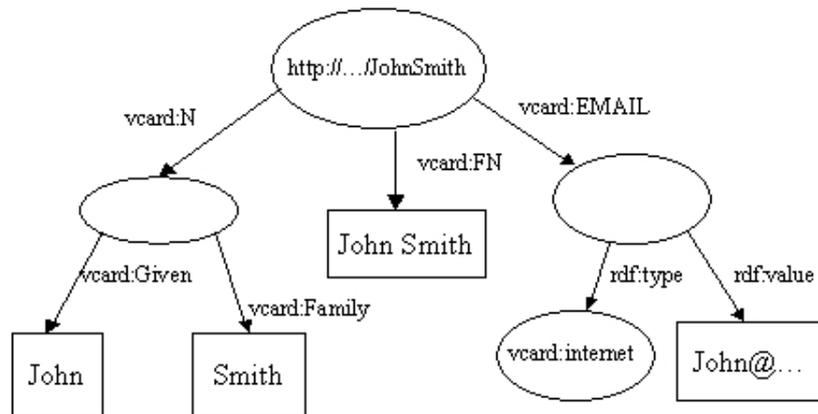


Abbildung 5.3: Zwei Modelle bevor die Operation „Union“ ausgeführt wird (Quelle: McBride, 2006)

Mit „*intersection*“ werden zwei Modelle miteinander verglichen. Von diesen Modellen wird dann ein „neues“ Modell erzeugt, welches alle Aussagen von den zwei Modellen enthält, welche in beiden Modellen vorkommen. Durch „*Difference*“ wird ein neues, unabhängiges Modell erzeugt. Dadurch werden zwei Modelle miteinander verglichen. Das „neue“ Modell enthält dann alle Aussagen welche nicht in beiden Modellen vorkommen. (McBride, 2006; Hewlett-Packard Development Company, 2007b)



**Abbildung 5.4:** Modell nach dem die Operation „Union“ ausgeführt wurde (Quelle: McBride, 2006)

### 5.3 SPARQL

Über das Modul „ARQ“ ist die Query Sprache „SPARQL“ in Jena verfügbar. Die ARQ Engine kann im Gegensatz zu SPARQL auch Queries, welche in RDQL geschrieben sind, parsen. SPARQL Queries können erzeugt und ausgeführt werden über die Klassen im Package `com.hp.hpl.jena.query`. Eine genauere Beschreibung der Sprache SPARQL findet sich im Abschnitt 4.4.3. (McCarthy, 2005)

### 5.4 Ontologie API

Die Jena „Ontologie API“ ist „sprach-neutral“. Um die Unterschiede zwischen den verschiedenen Darstellungen von jeder der vielen verschiedenen Ontologie Sprachen darzustellen, hat jede Ontologie Sprache ein Profil. Dieses Profil listet die erlaubten Konstrukte und die URI's von den Klassen und Eigenschaften auf. Das Profil ist an ein „Ontologie Modell“ gebunden. Das generelle Modell erlaubt den Zugriff auf Aussagen in einer Kollektion von RDF Daten. Die Eigenschaften, welche in der Ontologie Sprache definiert sind, passen zu den Zugriffsmethoden. (Hewlett-Packard Development Company, 2007b)

Ein Ontologie Modell ist eine Erweiterung des Jena RDF Models, welches weitere Fähigkeiten besitzt um Ontologie Datenquellen zu behandeln. Ein Ontologie Modell wird über das `Jena ModelFactory` erzeugt. Um eine Ontologie zu beschreiben, wird der Ausdruck „Dokument“ verwendet. Durch die Verwendung der `read` Methode wird ein Ontologie Dokument in das Ontologie Modell geladen. Dabei hat jedes Ontologie Modell einen zugehörigen Dokumentenmanager, welcher beim Verarbeiten und beim

Behandeln des Ontologie Dokuments hilft. Unter einem Dokumentenmanager versteht man eine Hilfsklasse, welche beim Importieren von Ontologie Dokumenten hilft. Es existiert ein globaler Dokumentenmanager, der automatisch vom Ontologie Modell verwendet wird. (Hewlett-Packard Development Company, 2007b)

## Kapitel 6

### SemanticLIFE Project

Durch das „SemanticLIFE“ Projekt wird versucht, sich der Vision von Vannevar Bush zu nähern. Im Jahre 1945 kreierte er die Idee von Memex als:

*„A device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory“*

(Bush, 1945)

Sinngemäß handelt es sich bei der Idee von Memex um ein Gerät, durch welches eine individuelle Speicherung von allen Büchern, Aufzeichnungen und Kommunikationsprotokollen möglich ist. Dabei wird das Gerät so entworfen, dass es bei Abfragen eine hohe Performance und gute Flexibilität aufweist. Dieses Gerät ist eine mechanische Ergänzung zum menschlichem Gedächtnis.

In diesem Kapitel wird das Projekt „SemanticLIFE“ beschrieben. Zuerst wird ein Überblick über das „Semantic Life“ Projekt gegeben. Anschließend wird die System Architektur beschrieben. Zuletzt wird auf die Query Verarbeitung innerhalb des „Semantic Life“ Systems eingegangen.

#### 6.1 Überblick

Das „SemanticLIFE“ System wurde entworfen um Lebensinformationen zu speichern, zu managen und abzufragen. Dadurch ist es möglich, dass man Daten gewinnen und speichern kann, während Kommentare zu Email Nachrichten, durchsuchten Webseiten, Telefonanrufen, Bildern, Kontakten und anderen Ressourcen hinzugefügt werden können. Das „SemanticLIFE“ System stellt einen intuitiven und effektiven Suchmechanismus zur Verfügung, welcher auf der gespeicherten Semantik basiert. Das Ziel dieses Projektes ist es, ein „Personal Information Management (PIM)“<sup>1</sup> über eine menschliche

---

1. im deut. „persönliches Informationsverwaltungssystem“ - im Folgenden weiter PIM genannt

Lebenszeit zu erstellen, welches Ontologien als Basis für die Darstellung der Inhalte verwendet. Das Projekt beinhaltet Aspekte aus den Bereichen „Human Computer Interface“, „Datenbanken“, „Data Mining“, „Security“, „Device Engineering“ usw. Bei der Entwicklung versucht man soviel wie möglich von anerkannten Standards zu verwenden. Das Projekt wird als „Open Source“ angeboten. (Ahmed, Hoang, Karim, Khusro, Lanzenberger, Latif, Michlmayr, Mustofa, Tho, Rauber, Schatten und Tjoa, 2004)

Lebende Systeme besitzen verschiedene Eigenschaften wie z.B. die Selbstregulierung von Prozessen, die Fortpflanzung oder das Wachstum. Ontologien von persönlichen Lebensereignissen enthalten Informationen über Objekte bzw. Ereignisse in unserem Leben wie z.B. Dokumente, Personen, Plätzen, Organisationen, Veranstaltungen und Aufgaben. In der physischen Welt werden Entitäten üblicherweise durch physische oder semantische Mittel mit einander verbunden. Unter einer Verbindung durch semantische Mittel wird die Bedeutung durch die menschliche Interaktion mit der physischen Welt hinzugefügt. Lebensobjekte in diesem System können hier als Informationsobjekte verstanden werden, die mit anderen Informationsobjekten nach ihrer semantischen Bedeutung mit einander verbunden werden. (Ahmed u. a., 2004)

Die PIM Systeme können den Menschen bei einigen alltäglichen Problematiken unterstützen, welche z.B. auftreten bei einer hohen Datenkomplexität, einer fragmentarischen Erinnerung, usw. Deshalb ist es von Vorteil, wenn das System in der Lage ist, ein Ereignis von einem bzw. einer BenutzerIn zu erfassen und zu definieren. Anschließend führt das System den dazu passenden Vorgang aus. Ahmed u. a. (2004) beschreiben, dass bei diesem Prozess mehrere Subprozesse involviert sind. Diese sind folgende:

- Erfassen von Ereignissen und damit verbundenen Informationen
- Prozess Situationen werden mit Aktionen verbunden wie z.B. bei einem aktiven Datenbanksystem
- Extrahieren von Metadaten von einem Ereignis oder der bzw. die BenutzerIn gibt die Daten manuell mit der semantischen Bedeutung ein
- Speichern der Daten inklusive des semantischen Kontexts als Ontologie
- Der bzw. die BenutzerIn kann die Daten abfragen

Typische Anwendungen für ein solches PIM kann mit den folgenden zwei Beispielen beschrieben werden. Ein bzw. eine StudentIn sucht ein Buch, welches von einem bestimmten Autor geschrieben worden ist. Der bzw. die Studierende weiß aber nur einen Teil des Titels, oder dass der Autor ein Absolvent einer bestimmten Universität ist. Als Ergebnis will der bzw. die Studierende z.B. einen Link zu einem Online Buchgeschäft haben. Der bzw. die Studierende soll KonsumentIn bei diesem Online Buchgeschäft sein, und das Buch soll lieferbar sein. Ein weiteres Beispiel sind WissenschaftlerInnen, welche in einem bestimmten Fachbereich arbeiten. Diese WissenschaftlerInnen sind daran interessiert mit anderen WissenschaftlerInnen in Kontakt zu treten. Gründe hierfür könnten sein, dass dieselben Interessen bzw. gleichartige Probleme vorhanden

sind, dass auf ähnlichen Konferenzen veröffentlicht wird oder dass kürzlich Aktivitäten auf einem gemeinsamen Forschungsgebiet statt gefunden haben. Das Ergebnis könnte hier die Webseiten oder die E-Mail Adressen der relevanten WissenschaftlerInnen sein. (Ahmed u. a., 2004)

Solche Probleme können nur gelöst werden, wenn mehrere Informationsquellen wie z.B. Newsgroups, Konferenzjournale, wissenschaftliche Datenbanken, E-Mail Boxen, Webseiten, usw. abgefragt werden. Das System muss verstehen, dass Entitäten die unterschiedlich gekennzeichnet sind, im semantischen Sinn identisch sind. Es muss auch im Stande sein, spezielle Sachverhalte zu verstehen und zu lösen, wie z.B. das einige Ergebnisse nur gültig sind in einem bestimmten Zeitintervall oder in einer bestimmten Sprache usw. (Ahmed u. a., 2004)

### 6.1.1 Beispiele für Personal Information Management (PIM) Projekte

Im Bereich der PIM Systeme sind Projekte bzw. Programme verfügbar, welche auf „Semantic Web“ Technologien basieren. Exemplarisch werden nachfolgend die Programme „MyLifeBits“, „Haystack“, „Snippet Manger“ und „LifeStreams“ kurz beschrieben.

Das Programm „MyLifeBits“ ist ein System zur Speicherung von diversen digitalen Medien. Erfasst und digital abgespeichert werden z.B. Artikel, Bücher, Karten, CD's, Briefe, Notizzettel, Papers<sup>2</sup>, Fotos, Bilder, Präsentationen, Amateurvideos, aufgezeichneten Vorträgen und Sprachaufzeichnungen. Das Programm konzentriert sich nicht auf die semantische Strukturierung oder auf das Auswerten des resultierenden Datenpools, sondern auf die Abfrage der gespeicherten Daten. (Gemmell, Bell, Lueder, Drucker und Wong, 2002; Hoang, Tjoa und Nguyen, 2006)

Ein weiteres Beispiel ist „Haystack“, welches ein Informationsmanagement System ist. „Haystack“ ist eine Plattform für die Erzeugung, die Organisation und die Darstellung von Informationen durch die Verwendung von Ontologien. Das Hauptziel ist eine qualitativ hochwertige Datenabfrage zur Verfügung zu stellen. Durch RDF basierte Datenmodelle kann der bzw. die BenutzerIn neue Objektattribute definieren, welche dabei helfen Informationen zu kategorisieren und abzufragen. Des Weiteren hilft dies beim Erzeugen von neuen Beziehungen zwischen den Objekten. (Hoang u. a., 2006; Huynh, Karger und Quan, 2002)

Durch das Programm „Snippet Manger“ kann eine Gruppe von BenutzernInnen kleinere Informationen abspeichern, wie z.B. Notizen, Bookmarks<sup>3</sup>, Nachrichten, usw. Diese Informationen werden mit den anderen GruppenmitgliederInnen geteilt, welche dann die Informationen kommentieren und strukturieren können. Dieses Programm baut auf

---

2. eine wissenschaftliche Publikation

3. im deut. Lesezeichen

das Programm „e-Person“ auf. „e-Person“ ist eine persönliche Informationsinfrastruktur in einem Netzwerk, welches von einem bzw. einer BenutzerIn verwendet wird um persönlichen Informationen zu speichern und abzufragen. Das Infrastruktur Design ist hierbei ein Hauptmerkmal, da für die Darstellung der gespeicherten Informationen RDF verwendet wird. (Banks, Cayzer, Dickinson und Reynolds, 2002; Hoang u. a., 2006)

„LifeStreams“ ist ein Programm für das Organisieren von persönlichen Informationen. Die Architektur von „LifeStreams“ basiert auf einer einfachen Datenstruktur. Es wird ein zeitlich geordneter Stream<sup>4</sup> von Dokumenten als Speichermodell verwendet. Diese Informationen können dann organisiert, lokalisiert, zusammengefasst und überwacht werden. (Banks u. a., 2002)

## 6.2 System Architektur

In den folgenden Unterabschnitten werden die Komponenten der System Architektur des „SemanticLIFE“ Projektes beschrieben. In Abbildung 6.1 ist diese Architektur dargestellt. Das „SemanticLIFE“ Projekt basiert auf einer hoch modularen Architektur. Durch die Verwendung von „Plug-in“ Mechanismen bleibt das System flexibel und erweiterbar. (Hoang u. a., 2006)

### 6.2.1 Erwerb von Daten

Zur Zeit ist der Großteil der verfügbaren Web-Applikationen und Frameworks so entwickelt worden, dass BenutzerInnen die Daten selbst analysieren und interpretieren müssen. Wenn aber komplexe Probleme gelöst werden sollen, welche angesammelte Informationen von unterschiedlichen Ressourcen benötigen, macht dies den Problemlösungsprozess zeitintensiv, inkonsistent, unzuverlässig und ist daher ungelegen / unbequem. Dadurch besteht ein Bedarf an einem „Wohl-Definierten“ Interface<sup>5</sup> zum Datenspeicher im World Wide Web (WWW) sowie zum persönlichen Datenspeichern. Deshalb müssen alle Informationsobjekte auf eine ontologische Weise gespeichert werden. Informationsobjekte können verschiedene Arten von Dokumenten, E-Mails, Bildern, Audio- oder Videostreams sein. (Ahmed u. a., 2004)

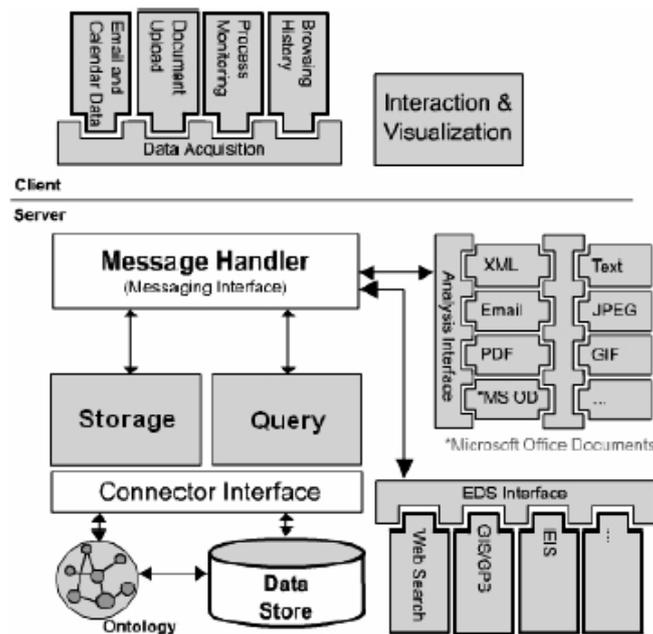
Im „SemanticLIFE“ System ist ein Modul für den Erwerb von Daten implementiert. Dabei sind grundsätzlich die folgenden drei unterschiedlichen Typen von Datenquellen bedeutend:

1. Daten werden automatisch erworben und in einem semantischen Datenspeicher gespeichert,

---

4. im deut. Datenkette

5. im deut. Schnittstelle



**Abbildung 6.1:** Architektur des „SemanticLIFE“ Systems (Quelle: Hoang u. a., 2006, S. 2)

2. Daten werden manuell durch den bzw. die BenutzerIn erworben,
3. Daten werden durch externe Datenquellen erworben und werden dann in den semantischen Datenspeicher importiert.

Es gibt Daten Repositories<sup>6</sup>, bei welchen es nicht sinnvoll ist, diese ins System zu importieren. Dies ist der Fall, wenn sich die Daten Repositories schnell ändern oder diese eine riesige Datenmenge beinhalten und hoch strukturiert sind. Beispiele dafür sind Literaturdatenbanken, Unternehmensinformationssysteme, Firmendatenbanken, Web Suchmaschinen usw. Solche externen Datenquellen werden bei Bedarf abgefragt und eine ontologische Darstellung wird durch ein „Plug-in“ im System generiert. (Ahmed u. a., 2004)

Das Importieren von Daten erfolgt über ein standardisiertes Interface, welches die Daten in einer (XML) Nachricht zusammenfasst. Dieses nachrichtenorientierte Design hat den Vorteil von „loose coupling“<sup>7</sup>, d.h. dass verschiedene Module leicht miteinander verbunden und kontrolliert werden können. Dies geschieht über eine zentrale Nachrichten Queue<sup>8</sup> und einen „Message Handler“. Allgemein können über eine Queue Daten zwischen verschiedenen Modulen ausgetauscht werden. Genau gesagt ist eine Queue eine

6. ähnlich wie ein Depot  
 7. im deut. lose Koppelung  
 8. im deut. Wartschlange

mehrelementige Datenstruktur. Die Nachrichten Queue führt hier auch Standard Operationen wie das Hinzufügen von Zeitstempeln bei bestimmten Operationen durch. Es wird auch ein Protokollmechanismus erzeugt, welcher das Analysieren des Systems im Fehlerfall ermöglicht. Der „Message Handler“ leitet die erworbenen Daten bzw. Nachrichten weiter zum Analyse Modul. (Ahmed u. a., 2004; Hoang u. a., 2006; Microsoft, 2000)

### 6.2.2 Analyse Modul

Das Analyse Modul erhält die Nachrichten bzw. Daten über den „Message Handler“. Anschließend werden dann von den eingehenden Nachrichten bzw. Daten effizient die Metadaten extrahiert. Durch diesen Mechanismus kann, abhängig vom Nachrichtentyp, mehr als ein Analyse Modul für die Verarbeitung einer Nachricht verwendet werden. Das Analyse Modul besteht aus mehreren „Plug-in“ Mechanismen für Extrahierungsmethoden und Indexing Techniken. Während der Analyse werden keine Daten entfernt oder modifiziert, damit die Verlaufsgeschichte erhalten bleibt. (Ahmed u. a., 2004; Hoang u. a., 2006)

Ein Informationsobjekt kann wiederum andere Informationen enthalten wie z.B. Bilder, Videoclips innerhalb von HTML oder eine PDF Datei. Deshalb muss ein Informationsobjekt mit allen weiteren enthaltenen Informationsobjekten analysiert werden, damit die Metadaten extrahiert werden können. (Ahmed u. a., 2004)

Zurzeit ist die automatische Informationsextraktion auf Nachrichten und Datenquellen beschränkt, die gewisse maschinenlesbare Strukturen zur Verfügung stellen. Zum Beispiel ist es äußerst schwierig eine strukturierte Information, wie z.B. eine Beschreibung über den Inhalt eines Bildes oder eines Films zu extrahieren. Durch das Hinzufügen von manuellen „Annotationen“<sup>9</sup> zum Informationsobjekt wird die Qualität verbessert. Dies ist eine Ergänzung zur Inhaltsanalyse und zur automatischen Metadaten Extraktion. (Ahmed u. a., 2004)

Wenn der Analyse Prozess beendet ist, dann wird die ursprüngliche Nachricht gemeinsam mit den erhaltenen Metadaten zurück zur Nachrichten Queue gesendet. Dadurch findet dann eine Weiterleitung an das Speicherungs-Modul statt. (Ahmed u. a., 2004)

### 6.2.3 Indexing und Speicherungs-Modul

Das Indexing und Speicherungs-Modul ist verantwortlich für das Extrahieren der XML basierten Metadaten Informationen aus einer Nachricht und für das Speichern im semantischen Datenspeicher. Nach dem Empfang einer Nachricht vom „Message Handler“ werden die folgenden Schritte durchgeführt:

---

9. im deut. Bemerkungen

1. Es findet eine Syntaxanalyse der Nachricht statt und es werden die Metadaten extrahiert.
2. Es werden die passenden Funktionen für das Schreiben in das spezifische Ontologie Framework über das „Connector Interface“ aufgerufen.
3. Es werden die Indizes aktualisiert.

Eine grundlegende Überlegung bei der Speicherung ist, dass die Größe der Speichermedien kontinuierlich steigt. Dadurch ist es möglich, dass alle Daten behalten werden können, welche ins System eingegeben wurden. Das ist eine Basis Funktion des Systems. (Ahmed u. a., 2004)

Eine wichtige Funktion dieses Moduls ist das Erzeugen von Verbindungen mit verwandten Entitäten. Solche Verbindungen können automatisch oder durch menschliche Interaktion erzeugt werden. Die im System gespeicherten Verbindungen sollten auch in der Lage sein, sich selbst zu aktualisieren und zu erneuern. Zusätzlich erlaubt diese „Nachrichten / Ereignis orientierte“ Architektur Probleme zu beheben, für welche normalerweise ein aktives Datenbanksystem verwendet wird. Ein aktives Datenbanksystem kann für datenbankinterne und datenbankexterne Anwendungsgebiete eingesetzt werden. Beispiele für solche Anwendungsgebiete sind Berechnung von Sichten, Zugriffskontrolle, Datenauswertungen, Finanzapplikationen, usw. (Ahmed u. a., 2004; Dittrich und Gatzju, 2000)

Im Indexing und Speicherungs-Modul werden die Informationensobjekte ontologisch strukturiert abgespeichert. Dabei werden die Daten in der Form von RDF Triples mit ihren Ontologien und Metadaten abgespeichert. Dieser Speicher wird in weiterer Folge auch „Metastore“ genannt. (Hoang u. a., 2006)

#### 6.2.4 Ontologie Repository Standard

OWL<sup>10</sup> und „Topic Maps“<sup>11</sup> sind zwei unterschiedliche Technologien um Ontologien zu erzeugen. Beide Technologien können für das Ontologie Repository verwendet werden, aber mit verschiedenen Abstraktionsniveaus. Es ist eine Interoperabilität zwischen diesen Technologien gegeben. Zur Zeit ist es noch nicht klar, welche der zwei Technologien in der Endversion des Systems verwendet wird. (Ahmed u. a., 2004)

In diesem Zusammenhang ist der Vergleich von Ontologie Algorithmen interessant. Bei Guggenberger (2008) werden Vergleichstechniken zwischen Ontologie Algorithmen beschrieben. Es werden zwei Ontologie Algorithmen analysiert und ausgeführt. Anschließend sollen die Ergebnisse beider Algorithmen in einem Resultat vereinigt werden. Dadurch sollen die Stärken des jeweiligen Algorithmus optimal eingesetzt werden.

---

10. Siehe dazu auch Smith u. a. (2004)

11. Siehe dazu auch <http://www.topicmaps.org/> - Zugriffsdatum: 18.12.2007

### 6.2.5 Interaktive Informationsabfrage

Die interaktive Informationsabfrage beschäftigt sich hauptsächlich mit der Such- und Query Funktionalität. Strukturierte Datenbanksysteme, wie Relationale oder Objektorientierte Datenbanksysteme, stellen normalerweise einen Querymechanismus zur Verfügung, welcher Queries auf hoch strukturierte Daten erlaubt. Das „SemanticLIFE“ System muss auch im Stande sein, mit „schwachen“ Queries zu arbeiten, welche durch das System in spezifischere Queries transformiert wird. (Ahmed u. a., 2004)

Im „SemanticLIFE“ Projekt ist es notwendig, dass Informationen von vielen Informationssystemen abgefragt werden können. Deshalb muss das „Query Verarbeitungs System“<sup>12</sup> dieses Projektes in der Lage sein, dass Query Abfragen auch „ungenau“ formuliert werden können. Wenn die Daten mit den semantischen Informationen gespeichert sind, ist es möglich eine starke „ungenau Suche“ zur Verfügung zu stellen. Der Begriff „ungenau“ hat in diesem Zusammenhang die zwei Bedeutungen. Zum einen sind die Ziele einer Query undefiniert. Zum anderem wird das Ziel der Query spezifiziert, aber es existieren Zweideutigkeiten innerhalb der Query, aufgrund der Heterogenität der verschiedenen Datenquellen. Aus diesem Grund muss das System in der Lage sein, diese Probleme während der Query Erzeugung zu beheben. Um dem bzw. der BenutzerIn ein besseres Verständnis der gespeicherten Daten zu geben, verwendet ein Teil des Query Moduls System Metadaten und globale Ontologien. Das Query Modul unterstützt sozusagen den bzw. die BenutzerIn bei der Erstellung von „genauen“ Queries. Um Datums- und Zeitwerte abzufragen ist ebenfalls eine „ungenau“ Suche wünschenswert. Wenn Datums- und Zeitwerte verwendet werden, dann müssen diese exakt definiert werden. Oft ist es der Fall, dass man den Datums- oder den Zeitwert nicht so genau angeben kann wie es laut Datentypdefinition gefordert ist. Auf diese Thematik wird in den weiteren Kapiteln eingegangen. (Ahmed u. a., 2004; Hoang u. a., 2006)

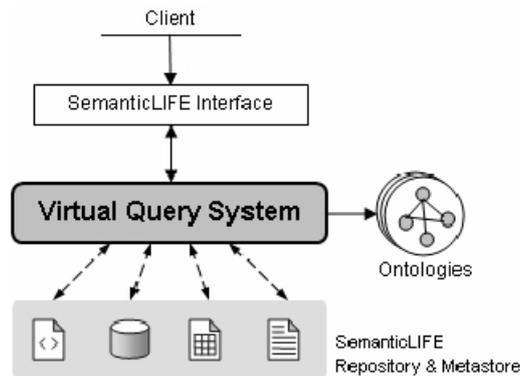
Die erhaltenen Query Ergebnisse benötigen eine Analyse und eine Klassifizierung um exaktere Resultate abzuleiten, welche zu den Anforderungen und zu den Präferenzen des bzw. der BenutzerIn passen. Um dies durchzuführen muss das System sich auf die Regeln und die Präferenzen des bzw. der BenutzerIn beziehen. Dazu werden Daten von internen und externen Datenquellen herangezogen. (Ahmed u. a., 2004; Hoang u. a., 2006)

Nach der Analyse und der Klassifikation werden die Ergebnisse in einer (XML) Nachricht zusammengefasst. Dieses Ergebnis kann dem bzw. der BenutzerIn dann über die Präsentationsschicht des Systems oder einem dazwischen liegenden Server, der das Ergebnis verarbeitet, präsentiert werden. Abbildung 6.2 zeigt den Query Prozess innerhalb des „SemanticLIFE“ Systems. Das „Virtual Query System“ wird im Abschnitt 6.3 beschrieben. Dieses System kann nicht nur die zuvor genannten Aufgaben behandeln,

---

12. im engl. query processing system

sondern es kann dem bzw. der BenutzerIn auch einen Überblick über die relevanten gespeicherten Informationen anzeigen. Dadurch spezifiziert der bzw. die BenutzerIn exaktere Queries basierend auf den Informationen und Daten, welche im System gespeichert sind. Das Query Benutzer Interface muss so implementiert sein, dass der bzw. die BenutzerIn die Möglichkeit besitzt, die Queries in einer wiederholenden Weise zu schreiben. (Ahmed u. a., 2004; Hoang u. a., 2006)



**Abbildung 6.2:** Query Prozess im „SemanticLIFE“ System (Quelle: Hoang u. a., 2006, S. 3)

### 6.2.6 Erweiterbarkeit und Persistenz von Information und Semantik

Die Basis des „SemanticLIFE“ Systems basiert auf der Analyse und der Extraktion der Metadaten. Neue Datenquellen kommen mit der Zeit hinzu und müssen durch das System behandelt werden. Es ist eine zeitintensive Aufgabe einen Support für neue Datenquellen hinzuzufügen, wenn das System eine feste Koppelung mit seinen Komponenten hat. Es wird daher eine leichtgewichtige asynchrone Nachrichten basierte Lösung benötigt, bei welcher neue Module in das System integriert werden können, ohne dass irgendwelche Änderungen im existierenden Code durchgeführt werden müssen. (Ahmed u. a., 2004)

Des Weiteren ist die Kompatibilität ein sehr wichtiges Thema bei Systemen, welche für die Gewinnung von Lebensinformationen und Metadaten entworfen wurden. Deshalb wird das „SemanticLIFE“ Projekt als „Open Source“ entwickelt. Dadurch können unterschiedliche EntwicklerInnen und BenutzerInnen am oder mit dem System arbeiten. Es wird für jedes Modul evaluiert, ob ein offener Standard für die Darstellung der Daten, für die Semantik oder für den Datenaustausch bereits existiert. (Ahmed u. a., 2004)

### 6.2.7 Client Interaktion

Der „Message Handler“ stellt einen standardisierten Kommunikationsmechanismus für unterschiedliche Arten von Clients zur Verfügung. Zurzeit ist das System an individuellen BenutzerInnen orientiert. Das Server System muss ein Benutzer Management und eine Authentifizierung implementieren. Deshalb muss auch jede Nachricht die Benutzerinformationen enthalten, z.B. von welchem bzw. welcher BenutzerIn eine Query ausgeführt worden ist. Alle Verarbeitungsmodule, wie das Indexing und Speicherungs-Modul oder die interaktive Informationsabfrage, müssen dann die Nachrichten filtern oder die Ergebnisse der Query entsprechend den Benutzerrechten und Rollen abfragen. Dabei können auch zusätzliche Profile von verschiedenen Typen, wie von BenutzerInnen, Geräten, Applikationen, Aufgaben und Interaktions Objekten enthalten sein. (Ahmed u. a., 2004)

## 6.3 Virtuelles Query System (VQS)

Das Formulieren von eindeutigen Queries ist eine anspruchsvolle Aufgabe, da der bzw. die BenutzerIn nicht den Überblick über die Semantik der im System gespeicherten Daten besitzt. Dieses Problem soll durch das „virtuelle Query System“<sup>13</sup> gelöst werden. Dazu werden die im System vorhandenen Daten grafisch über eine Ansicht in der Benutzeroberfläche dargestellt. Wenn der bzw. die BenutzerIn darüber bescheid weiß, welche Informationen im System vorhanden sind, dann können auch genauere virtuelle Queries auf Grund dieser Daten spezifiziert werden. Eine virtuelle Query wird im XML Format kodiert. Das VQS muss die Benutzerangaben in einem geeigneten Format erhalten, damit diese weiterverarbeitet werden können. In diesem XML Format wird unter anderem auch angegeben, in welchem Format das Ergebnis der Query Abfrage retourniert werden soll. Das VQS kann als Ergebnis eine Text Datei, eine XML Datei, eine RDF Datei bzw. einen RDF Graph oder ein „Query Antwort Objekt“ zurückliefern. Die verschiedenen Query Ergebnisformate werden im Abschnitt 6.4.4 genauer beschrieben. In den nachfolgenden Abschnitten wird das Design und der Workflow<sup>14</sup> des VQS beschrieben. (Hoang u. a., 2006)

### 6.3.1 VQS Design

Das VQS wurde so entworfen, dass es als ein unabhängiges „Plug-in“ Framework innerhalb des „SemanticLIFE“ Projektes verwendet werden kann.

---

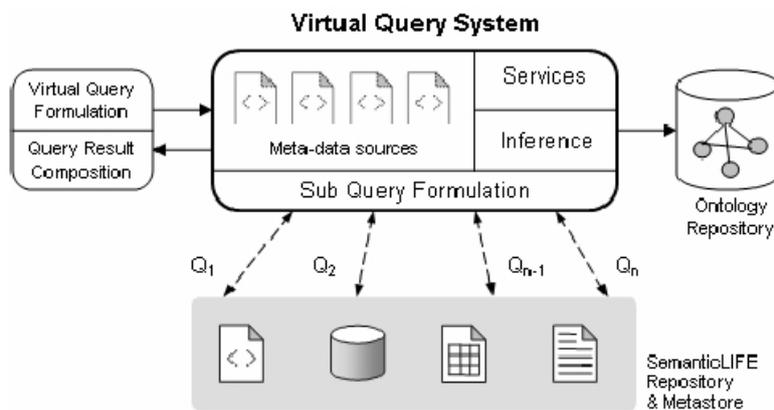
13. Im Folgenden weiter VQS genannt

14. im deut. Arbeitsablauf

Dabei besteht das VQS aus den folgenden sechs Modulen:

1. „Meta-data Source“<sup>15</sup> Modul
2. „Ontology Repository“ Modul
3. „Sub-Query Formulation“<sup>16</sup> Modul
4. „Services“ Modul (bestehend aus den Services „Ontology Mapping“, „Query Caching“ und „Query Refinement“<sup>17</sup>)
5. „Ontology-based Inference“<sup>18</sup> Modul
6. „Virtual Query User Interface“<sup>19</sup> Modul (bestehend aus den Submodulen „Virtual Query Formulation“<sup>20</sup> und „Query Result Composition“<sup>21</sup>)

In der Abbildung 6.3 sind diese Module abgebildet und es wird auch dargestellt, wie diese Module zusammenarbeiten. Dabei werden durch  $Q_i$  ( $i = 1..n$ ) die „echten“ Query Abfragen dargestellt, welche verwendet werden, um den Metastore abzufragen. (Hoang u. a., 2006)



**Abbildung 6.3:** Virtuelles Query System (VQS) im „SemanticLIFE“ System (Quelle: Hoang u. a., 2006, S. 3)

### 6.3.1.1 „Meta-data Source“ Modul

Das Modul „Meta-data Source“ enthält die Quellen für die Metadaten des VQS. Dieses Modul dient als „virtuelle“ Datenquelle für den bzw. die BenutzerIn. Dadurch erhält der

- 
15. im deut. Meta-Daten Quelle
  16. im deut. Sub-Abfrage Formulierung
  17. im deut. Abfrage Verfeinerung
  18. im deut. Ontologie basierende Folgerung
  19. im deut. virtuelles Benutzer Interface für Abfragen
  20. im deut. virtuelles Abfrage Formulierung
  21. im deut. Anordnung des Abfrage Ergebnis

bzw. die BenutzerIn einen Überblick über die im System gespeicherten Daten, und somit können dann auch präzisere virtuelle Queries formuliert werden. Das VQS sammelt die Metadaten von den Datenquellen des „SemanticLIFE Repository“ und vom Metastore. Auf diesen Metadaten Quellen wird ein Analyse Prozess sowie eine statistische Berechnung durchgeführt um dadurch weitere Informationen zu erhalten.. Anschließend werden die erhaltenen Informationen in diesem Modul gespeichert und weitergegeben. In der weiteren Verarbeitung der Query wird auch auf dieses Modul, als „Abbild“ über die Datenbasis im System, verwiesen. (Hoang u. a., 2006)

#### 6.3.1.2 „Ontology Repository“ Modul

Das „Ontology Repository“ Modul bildet den Kern des VQS. Dieses Modul enthält die Ontologien, welche im VQS System verwendet werden. (Hoang u. a., 2006)

#### 6.3.1.3 „Sub Query Formulation“ Modul

Ein wichtiger Teilbereich des VQS ist das „Sub Query Formulation“ Modul. Ausgehend von der virtuellen Query von einem bzw. einer BenutzerIn erzeugt dieses Modul Sub-Queries basierend auf der globalen Ontologie für die spezifischen Datenquellen. Diese Sub-Queries sind in der Abbildung 6.3 durch  $Q_i$  ( $i = 1..n$ ) dargestellt. Des Weiteren führt dieses Modul auch Schlussfolgerungen basierend auf der Benutzeranfrage durch, um weitere mögliche Sub-Queries zu erhalten. Nach diesem Prozess werden die „echten“ Queries für jede Datenquelle, basierend auf einer RDF Query Sprache, erzeugt. Dadurch kann dieses Modul mit Zweideutigkeiten eines Ausdrucks innerhalb einer Benutzeranfrage umgehen. (Hoang u. a., 2006)

#### 6.3.1.4 „Services“ Modul

Das „Services“ Modul des VQS besteht aus drei Services. Diese Services sind das „Ontology Mapping“ Service, das „Query Caching“ Service und das „Query Refinement“ Service. Es wird ein „Ontology Mapping“ Mechanismus benötigt, da es einfacher ist eine existierende Ontologie abzubilden, als eine neue Ontologie anzulegen. Dieses Service ist der Mechanismus, mit welchem eine lokale Ontologie in eine globale Ontologie abgebildet wird. Durch das „Query Caching“ Service wird die Systemleistung des VQS verbessert, indem die Queries für eine bestimmte Zeit im Speicher zwischen gespeichert werden. Dabei wird zwischen den zwei Caching Methoden „Query Caching“ und „Result Caching“ unterschieden. Beim „Query Caching“ wird der Prozess der Erzeugung der Sub-Queries zwischen gespeichert und beim „Result Caching“ werden die Ergebnisse der virtuellen Query zwischen gespeichert. Das „Query Refinement“ Service ist für die Ontologie basierende Query Verarbeitung wichtig. Dies stellt einen interaktiven Weg für das VQS dar um mit mehrdeutigen Benutzer Queries umzugehen. Das

„Query Refinement“ Service vom VQS ist ein halbautomatisierter Prozess. Während dieses Prozesses bekommt der bzw. die BenutzerIn eine Liste mit Verbesserungsvorschlägen. Durch diese Verbesserungen würden sich die Mehrdeutigkeiten verringern. (Hoang u. a., 2006)

#### 6.3.1.5 „Ontology-based Inference“ Modul

Ein weiteres Modul des VQS ist das „Ontology-based Inference“ Modul. Durch dieses Modul kann das System die virtuellen Queries von einem bzw. einer BenutzerIn analysieren und auswerten. Während dessen werden die Sub-Queries basierend auf der „Inference“ Ontologie erzeugt. (Hoang u. a., 2006)

#### 6.3.1.6 „Virtual Query User Interface“ Modul

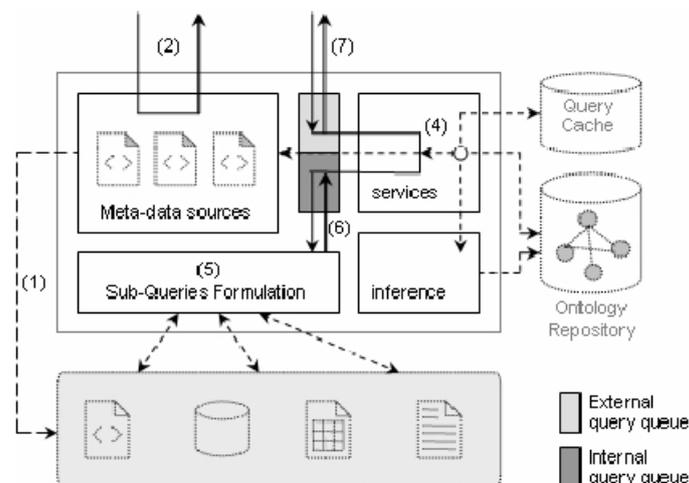
Das Modul „Virtual Query User Interface“ besteht aus den zwei Submodulen „Virtual Query Formulation“ und „Query Result Composition“. Durch das „Virtual Query User Interface“ werden dem bzw. der BenutzerIn die im System vorhandenen Daten graphisch angezeigt. Dadurch wird der bzw. die BenutzerIn bei der Formulierung der virtuellen Query unterstützt. Das „Query Result Composition“ Modul führt die Integration und das Zusammenfassen der Sub-Query Ergebnisse durch. Diese Ergebnisse werden dem bzw. der BenutzerIn dann in weiterer Folge präsentiert. (Hoang u. a., 2006)

### 6.3.2 VQS Workflow

Abbildung 6.4 zeigt den Workflow innerhalb des VQS. Dabei werden Tätigkeiten vom System durch gestrichelte Linien bzw. Pfeile dargestellt. Normale Linien bzw. Pfeile stellen eine Interaktion zwischen dem bzw. der BenutzerIn und dem VQS dar. Es wird eine interne und eine externe Queue verwendet. Die externe Queue wird für das Empfangen von Queries und für das Retournieren der Ergebnisse verwendet. Innerhalb des Systems wird die interne Queue verwendet. Diese interne Queue wird für das Empfangen von verarbeiteten Queries und für das Retournieren der Ergebnisse verwendet. Die nummerierten Schritte aus Abbildung 6.4 werden nachfolgend beschrieben. Diese Schritte werden chronologisch durchgeführt. (Hoang u. a., 2006)

Wenn das VQS im „SemanticLIFE“ System installiert ist, dann wird Schritt (1) ausgeführt. Während dieses Schrittes werden die Metadaten vom Metastore gesammelt und es werden die benötigten statische Berechnungen durchgeführt. Die Ergebnisse daraus werden dann im „Meta-data Source“ Modul gespeichert. (Hoang u. a., 2006)

Schritt (2) beschreibt die Interaktion vom VQS mit dem bzw. der BenutzerIn. Der bzw. die BenutzerIn kann die Informationen aus Schritt (1) durch die Formulierung von geeigneten Queries abrufen. Diese Queries stellen die Verbindung zwischen dem bzw. der



**Abbildung 6.4:** Workflow des Virtuellen Query System (VQS) im „SemanticLIFE“ System (Quelle: Hoang u. a., 2006, S. 5)

BenutzerIn und dem VQS dar. Über das Query Interface, welches in Abbildung 6.2 dargestellt ist, werden von dem bzw. der BenutzerIn die Queries eingegeben. Über dieses Interface können dem bzw. der BenutzerIn die Informationen aus den Metadaten dargestellt werden. Dadurch können mehrdeutige Abfragen besser vermieden werden. (Hoang u. a., 2006)

Der nachfolgende Schritt (3) ist in der Abbildung 6.4 nicht eingezeichnet. Durch diesen Schritt werden die zuvor spezifizierten Queries an das VQS gesendet. Ab diesem Schritt werden die virtuellen Queries ausgewertet. Dies geschieht durch die Ontologie basierenden Services des VQS. Diese Services werden dann im Schritt (4) aufgerufen und ausgeführt. (Hoang u. a., 2006)

Im Schritt (4) wird auf das „Query Caching“, das „Ontology Repository“ und auf das „Inference“ Modul zugegriffen. Das VQS „Query Caching“ Service ist verantwortlich für die Verarbeitung der Query. Als erstes wird geprüft ob, diese Query im „Query Cache“ gefunden wird. Wenn dies der Fall ist, dann wird das Query Ergebnis gleich zum bzw. zur BenutzerIn retourniert. Wenn aber die Abfrage des „Query Cache“ kein passendes Ergebnis liefert, dann wird eine Analyse und Evaluierungprozess gestartet um weitere Informationen zu erhalten. Über die interne Query Queue wird dann die Query mit den analysierten Informationen zum Schritt (5), der „Sub-Query Formulation“, weitergeleitet. Im Schritt (5) werden Sub-Queries für jede spezifizierte Datenquelle generiert. (Hoang u. a., 2006)

In den Schritten (6) und (7) werden dann die Ergebnisse der Query Abfragen an den bzw. die BenutzerIn übermittelt. Die Ergebnisse werden vom „Query Result Composi-

tion“ Modul zusammengefasst, bevor die Ergebnisse dem bzw. der BenutzerIn angezeigt werden. Die Übertragung der Ergebnisse erfolgt über die interne und die externe Queue. (Hoang u. a., 2006)

## 6.4 Virtuelle Query Sprache (VQL)

Die „Virtual Query Language“<sup>22</sup> vom „SemanticLIFE“ System wird vom VQS verwendet, um Informationen aus dem „SemanticLIFE“ System abzufragen. Die VQL ist ein „Front-End“ Ansatz für benutzerorientierte Informationsabfragen. Für das Semantic Web existieren viele unterschiedliche Query Sprachen, welche einen ausdrucksstarken Mechanismus zur Verfügung stellen. Mit diesen Query Sprachen ist es aber schwierig Queries für „durchschnittliche“ BenutzerInnen und für abstrakte Informationsabfragen zu erstellen. Daher ist VQL vorgesehen als eine einfache Query Sprache. (Hanh und Tjoa, 2006)

Im Zusammenhang mit VQS und dem „SemanticLIFE“ Projekt hat VQL verschiedene Ziele. Eines dieser Ziele ist, dass VQL die BenutzerInnen bei der Navigation durch das System unterstützt. Dies geschieht über semantische Links. Durch semantische Links wird eine Verbindung zwischen Instanzen und Objekten innerhalb der Datenquellen angezeigt. Dadurch werden ausdrucksstarke Queries, welche auf Ontologien basieren, zur Verfügung gestellt. Ein anderes Ziel ist, dass VQL die Kommunikation zwischen den Query Modulen und anderen Komponenten vereinfacht. Dadurch werden auch die Komponenten des „SemanticLIFE“ Systems unabhängiger. Des Weiteren wird durch VQL die Portabilität des Systems verbessert. Zur Zeit verwendet das VQS eine RDF Query Sprache für die „Back-End“ Datenbank. Wenn man zukünftig diese Query Sprache ändern möchte, dann müssen keine anderen Komponenten des Systems angepasst werden. (Hanh und Tjoa, 2006)

Der VQL Parser übersetzt die VQL Query in die entsprechende RDF Query und greift auf den Metastore zu, um die entsprechenden Ergebnisse zu erhalten. Anschließend werden die Ergebnisse in das gewünschte Format umgewandelt und die erhaltenen Ergebnisse werden zurück zum bzw. zur BenutzerIn gesendet. (Hanh und Tjoa, 2006)

### 6.4.1 VQL Query Aufbau

Eine VQL Query ist ein XML Dokument. Auf dem ersten Level einer VQL Query wird der Querybody angelegt. Dieser besteht aus dem `<query>` Element mit dem Attribut `type`. Im `type` Attribut muss die Art der Query spezifiziert werden. Es wird hier zwischen Query Typen `data`, `schema`, `itql1` und `itql2` unterschieden. Diese werden im Kapitel 6.4.2 beschrieben. (Hanh und Tjoa, 2006)

---

22. im deut. virtuelle Abfrage Sprache - im Folgenden weiter VQL genannt

Im zweiten Level der Query werden die benötigten Subelemente angegeben. Abhängig von der Art der Query werden unterschiedliche Elemente verwendet. Für eine Query vom Typ `data` werden die Elemente `<params>`, `<sources>` und `<relations>` benötigt. Diese Elemente besitzen Kindelemente auf dem dritten Level. Listing 6.1 zeigt eine Query vom Typ `data`. Für eine Query vom Typ `schema` oder vom Typ `itql1` wird nur das `<statement>` Element benötigt. Listing 6.2 zeigt eine Query vom Typ `schema`. Für eine Query vom Typ `itql2` werden die Elemente `<select>`, `<from>`, `<where>` benötigt. Listing 6.3 zeigt eine Query vom Typ `itql2`. Bei allen Query Arten wird auf diesem Level über das Element `<resultformat>` das Format für das Query Ergebnis angegeben. (Hanh und Tjoa, 2006)

Im dritten Level der Query sind die VQL Daten Queries. Die Elemente auf diesem Level sind Kindelemente der Elemente `<params>`, `<sources>` und `<relations>`. (Hanh und Tjoa, 2006)

## 6.4.2 VQL Query Typen

Es werden die Query Typen `data`, `schema`, `itql1` und `itql2` unterstützt. Bei allen Query Typen wird das Query Ergebnis Format durch das Element `<resultformat>` angegeben. Alle Query Typen unterstützen dieselben Ergebnisformate. Es kann als Ergebnis eine Text Datei, eine XML Datei, eine RDF Datei bzw. ein RDF Graph oder ein „Query Antwort Objekt“ zurückgeliefert werden. Dadurch ist der bzw. die BenutzerIn recht flexibel bei der Weiterverarbeitung der Ergebnisse. (Hanh und Tjoa, 2006; Hoang u. a., 2006)

### 6.4.2.1 VQL Query Typ „data“

Der Query Typ `data` wird üblicherweise für die Informationsabfrage verwendet. Generell besteht diese Query aus den vier Teilbereichen „Parameter“, „Datenquelle“, „Constraints“<sup>23</sup> und „Query Ergebnis Format“. Im Listing 6.1 ist eine Query vom Typ `data` dargestellt. (Hanh und Tjoa, 2006)

```

1 <query type=" data ">
2   <params>
3     <param show=" 1 "
4       name=" s1:messageTimeStamp ">2007-11-01>/param>
5     <param show=" 0 "
6       name=" s2:messageTimeStamp ">2007-12-18>/param>

```

23. im deut. Einschränkungen

```

7  </params>
8  <sources>
9    <source name="fileupload">FileUpload</source>
10   <source
11     name="browsingsession">BrowseSession</source>
12 </sources>
13 <relations>
14   <relation id="1" param="s1" source="">dt:gt</relation>
15   <relation id="2" param="s2" source="">dt:lt</relation>
16 </relations>
17 <resultformat>xml</resultformat>
18 </query>

```

**Listing 6.1:** Beispiel einer XML basierenden VQL Query vom Typ „data“ (Quelle: Hanh und Tjoa, 2006, S. 5)

Der erste Teilbereich „Parameter“ wird durch das Element `<params>` angegeben. Hier werden die unterschiedliche Parameter spezifiziert um die gesuchten Informationen zu erhalten. Das Element `<params>` besitzt Kindelemente vom Elementtyp `<param>`. Bei diesem `<param>` Element werden die Attribute `show` und `name` benötigt. Das Attribut `show` kann die Werte „0“ oder „1“ haben. Durch „1“ wird angegeben, dass das Ergebnis dieses Parameters im Query Ergebnis angezeigt werden soll und durch „0“ wird das Ergebnis nicht angezeigt. Das Attribut `name` besteht aus den zwei Teilen Variablenname und Meta-Information, welche durch (:) getrennt sind. Dieses Element kann dann die zwei optionalen Attribute `order` und `exclude` besitzen. Das Attribut `order` wird für das Sortieren verwendet, wobei es die Werte „0“ oder „1“ haben kann. Dadurch wird angegeben, ob das Ergebnis sortiert werden soll oder nicht. Das Attribut `exclude` wird verwendet um bestimmte Informationen vom Query Ergebnis Resultat auszuschließen. (Hanh und Tjoa, 2006)

Der zweite Teilbereich Datenquelle wird durch Element `<sources>` angegeben. Mit diesem Element werden die Datenquellen angegeben, aus welchen die Informationen abgefragt werden sollen. Die Parameter, welche im Teilbereich `<params>` angegeben sind, müssen in einem Zusammenhang mit den Datenquellen aus diesem Teilbereich stehen. (Hanh und Tjoa, 2006)

Das Element `<sources>` besitzt Kindelemente vom Elementtyp `<source>`. Im Element `<source>` werden die Namen der betroffenen Datenquellen für die Anforderung des bzw. der BenutzerIn angegeben. Dieses Element hat das Attribut `name`. Durch dieses Attribut wird der Name der Datenquelle angegeben. (Hanh und Tjoa, 2006)

Im dritten Teilbereich Constraints wird durch das Element `<relations>` die Einschränkungen für die VQL Query angegeben. Es werden die Relationen zwischen den

Datenquellen, den Paramtern mit den VQL Operatoren kombiniert. (Hanh und Tjoa, 2006)

Das Element `<relations>` besitzt Kindelemente vom Elementtyp `<relation>`. Im `<relation>` Element werden Einschränkung für die VQL Query angegeben. Dabei werden die Attribute `id`, `param`, `source` und `or` benötigt, wobei das letzte Attribut optional ist. Das Attribut `id` hat einen numerischen Wert. Durch das Attribut `param` wird der Variablenname angegeben. Im Attribut `source` wird der Name der Datenquelle angegeben. Dieser Wert kann auch leer bleiben, falls genau eine Datenquelle existiert. Um eine „ODER Bedingung“ zu erzeugen, bekommt das Attribut `or` den Wert des Attributes `id` von einem anderen `<relation>` Element. Dadurch wird ein „ODER“ Ausdruck zwischen dem Wert aus dem Attribut `id` und dem Wert aus dem Attribut `or` erzeugt. Der Operator für die Einschränkungen wird als Wert im `<relation>` Element angegeben<sup>24</sup>. (Hanh und Tjoa, 2006)

Der vierte Teilbereich Query Ergebnis Format wird durch das `<resultformat>` Element angegeben. (Hanh und Tjoa, 2006)

#### 6.4.2.2 VQL Query Typ „schema“

Der Query Typ `schema` besteht aus dem Bereich `<statement>` und aus dem Bereich `<resultformat>`. Im Listing 6.2 ist eine Query vom Typ `schema` dargestellt. (Hanh und Tjoa, 2006)

```

1 <query type="schema">
2   <statement>
3     select $s, $p $o
4     from rmi:/192.168.168.174/slife#BaseModel
5     where $s $p $o
6   </statement>
7   <resultformat>xml</resultformat>
8 </query>

```

**Listing 6.2:** Beispiel einer XML basierenden VQL Query vom Typ „schema“ (Quelle: Hanh und Tjoa, 2006, S. 5)

Das Attribut `type` des Elements `<query>` gibt an, dass es sich hier um den Query Typ `schema` handelt. Das Element `<statement>` enthält eine RDF Query. Diese Query enthält im `<select>` und `<where>` Element die Variablen `$s`, `$p` und `$o`. Dadurch

24. Die Operatoren werden im Abschnitt 6.4.3 beschrieben

wird angegeben, dass das Subjekt ( $\$s$ ), das Prädikat ( $\$p$ ) und das Objekt ( $\$o$ ) von dem Wert aus den `<from>` Element ausgegeben werden soll. Im zweiten Teilbereich wird das Query Ergebnis Format durch das Element `<resultformat>` angegeben<sup>25</sup>. Zur Zeit werden schema Queries dem bzw. der BenutzerIn über Query Templates zur Verfügung gestellt. (Hanh und Tjoa, 2006)

#### 6.4.2.3 VQL Query Typ „iTQL“

Bei den „iTQL“ Query Typen werden RDF Queries in VQL Queries verwendet. Es wird hier zwischen den zwei Query Typen „iTQL type 1“ und „iTQL type 2“ unterschieden. (Hanh und Tjoa, 2006)

Das Format vom Query Typ „iTQL type 1“ (`itql1`) ist ähnlich zum Format des Query Typs `schema` aus dem Listing 6.2. Wobei im Element `<query>` beim Attribut `type` als Wert `itql1` an Stelle von `schema` angegeben werden muss. Bei diesem Query Typ wird die komplette RDF Query in ein Element eingebettet. (Hanh und Tjoa, 2006)

Im Format vom Query Typ „iTQL type 2“ (`itql2`) wird jeder Teil der RDF Query in ein eigenes VQL Query Element eingebettet. Im Listing 6.3 ist eine Query vom Typ „iTQL type 2“ dargestellt. (Hanh und Tjoa, 2006)

```

1 <query type="itql2">
2   <select>$s $p $o</select>
3   <from>
4     rmi://192.168.168.174/slife#BaseModel
5   </from>
6   <where>$s $p $o</where>
7   <resultformat>xml</resultformat>
8 </query>

```

**Listing 6.3:** Beispiel einer XML basierenden VQL Query vom Typ „iTQL type 2“ (Quelle: Hanh und Tjoa, 2006, S. 7)

Das Attribute `type` des Elements `<query>` gibt an, dass es sich hier um den Query Typ `itql2` handelt. Für eine Query vom Typ `itql2` werden die Elemente `<select>`, `<from>`, `<where>`, `<orderby>`, `<limit>` und `<offset>` benötigt, wobei die letzten drei Elemente optional sind. Im Listing 6.3 werden im `<select>` sowie im `<where>` Element die Variablen  $\$s$ ,  $\$p$  und  $\$o$  verwendet. Dadurch wird angegeben, dass das Subjekt ( $\$s$ ), das Prädikat ( $\$p$ ) und das Objekt ( $\$o$ ) von dem Wert aus

25. Die verschiedenen Ergebnisformate werden im Abschnitt 6.4.4 beschrieben

den `<from>` Element ausgegeben werden soll. Abschließend wird das Query Ergebnis Format durch das Element `<resultformat>` angegeben. (Hanh und Tjoa, 2006)

### 6.4.3 VQL Query Operatoren & Ausdrücke

In VQL werden die logischen Operatoren AND, OR und NOT unterstützt. NOT ist in jedem `<param>` Element durch das `exclude` Attribut definiert. AND und OR Operatoren werden im `<relation>` Element definiert. OR wird durch das `or` Attribute definiert, z.B. `or="2"` bedeutet, dass die aktuelle Einschränkung mit der Einschränkung der `id=2` kombiniert wird, durch die Verwendung des logischen OR Operators. AND wird impliziert in einer Relation ohne `or` Attribut. (Hanh und Tjoa, 2006)

Es werden auch Vergleichsoperatoren in VQL unterstützt. Diese Vergleichsoperatoren werden im `<relation>` Element der VQL Query verwendet. Es werden die Vergleichsoperatoren `equal`, `gt`, `lt`, `dt:gt`, `dt:lt`, `str:match` und `ft:match` unterstützt. (Hanh und Tjoa, 2006)

`equal` ist eine Vergleichsoperation, welche verwendet wird, indem das `<relation>` Element leer bleibt. Der Operator `gt` bedeutet „Größer-Als“<sup>26</sup>, welcher für das Vergleichen von Basis Datentypen wie z.B. `String` oder `numbers` verwendet wird. Der Operator `lt` bedeutet „Kleiner-Als“<sup>27</sup>, welcher für das Vergleichen von Basis Datentypen wie z.B. `String` oder `numbers` verwendet wird. Der Operator `dt:gt` und der Operator `dt:lt` werden verwendet für den Vergleich von Datumswerten oder Zeitwerten. Beim Operator `dt:gt` muss der gesuchte Wert zeitlich nach dem angegebenen Datumswert oder Zeitwert liegen. Der Operator `dt:lt` wird verwendet, wenn der gesuchte Wert zeitlich vor dem angegebenen Datumswert oder Zeitwert liegen muss. Mit dem `str:match` Operator ist es möglich innerhalb einer Zeichenkette nach einem bestimmten Zeichenmuster zu suchen. Der leistungsfähigste Operator ist `ft:match`. Dieser stützt sich auf die Voll-Text Indizes des „SemanticLIFE“ Metastore. Dieser wird verwendet für die Voll-Text Suche innerhalb von Daten wie z.B. suchen im Inhalt einer Datei oder eines Email Anhangs. (Hanh und Tjoa, 2006)

Die Ausdrücke für die Bedingungen der Query werden mit Hilfe der Booleschen Operatoren OR und AND formuliert. Zuerst werden die „ODER Bedingungen“ mit einander kombiniert. Dabei verwendet VQL das `or` Attribut vom `<relation>` Element als Booleschen OR Operator. Anschließend wird der Boolesche AND Operator für das Kombinieren im finalen Ausdruck verwendet. Die Reihenfolge der `<relations>` Elemente ist wichtig für die Kombination der Ausdrücke. (Hanh und Tjoa, 2006)

---

26. im engl. Greater-Than

27. im engl. Less-Than

#### 6.4.4 VQL Query Ergebnis Formate

VQL kann die Ergebnisse der Query Abfrage als Text Datei, XML Datei, RDF Datei bzw. RDF Graph oder „Query Antwort Objekt“ zurückliefern. Dies erhöht die Flexibilität für das weitere Arbeiten mit diesen Ergebnissen. Innerhalb einer VQL Query wird das gewünschte Ergebnisformat auf dem zweiten Level durch das `<resultformat>` Element angegeben. (Hanh und Tjoa, 2006)

Innerhalb einer VQL Query wird durch `<resultformat>xml</resultformat>` angegeben, dass eine XML Datei zurückgeliefert werden soll. Das VQL Query Ergebnis Format XML ist dem W3C Format für die SPARQL Query Ergebnisse ähnlich<sup>28</sup>. Dieses XML Format besteht aus zwei Teilen. Im ersten Teil werden die Query Parameter und die zusätzlich benötigten Informationen aufgelistet. Der zweite Teil ist eine Liste mit den gefundenen Instanzen. (Hanh und Tjoa, 2006)

Durch `<resultformat>text</resultformat>` wird innerhalb der VQL Query angegeben, dass eine Text Datei zurückgeliefert werden soll. Das Ergebnisformat Text ist strukturiert aufgebaut. Es wird jeweils die Variable und der Werte von jedem Element als Paar in der Form `VARIABLEN_NAME = VALUE` ausgegeben. Diese Paare sind durch (;) miteinander verbunden. Jeweils eine Zeile ist für ein Element. (Hanh und Tjoa, 2006)

Innerhalb einer VQL Query wird durch `<resultformat>rdf</resultformat>` angegeben, dass eine RDF Datei bzw. ein RDF Graph zurückgeliefert werden soll. Dieses Ergebnisformat ist für „Semantic Web“ Client Applikationen entworfen worden. Bei diesen Applikationen werden Metadaten bevorzugt. Aus dem Query Ergebnis wird dann ein RDF Graph erzeugt, bevor das Ergebnis dem bzw. Benutzer retourniert wird. (Hanh und Tjoa, 2006)

Durch `<resultformat>object</resultformat>` wird innerhalb einer VQL Query angegeben, dass ein „Query Antwort Objekt“ zurückgeliefert werden soll. Aufgrund der Kommunikation zwischen den Komponenten innerhalb des „SemanticLIFE“ Systems kann es vorkommen, dass die Komponenten die Query Ergebnisse ohne eine Formatumwandlung benötigen. VQL serialisiert dann die Ergebnisse und sendet anschließend die serialisierten „Query Ergebnis Objekte“ zurück zur fragenden Komponente. (Hanh und Tjoa, 2006)

---

28. Siehe dazu auch Beckett und Broekstra (2007)

# Kapitel 7

## Praxis: Analyse & Design

Dieses Kapitel befasst sich mit den Vorbereitungen zur Implementierung. Dabei werden die Vorüberlegungen und die Rahmenbedingungen für die Implementierung beschrieben. Durch die Beschreibung von Anwendungsfällen soll der Nutzen der Implementierung verdeutlicht werden. Der letzte Abschnitt dieses Kapitels beschreibt die Konzeption und das Software Design. In diesem werden auch die Design Entscheidungen begründet.

### 7.1 Vorbereitungen

Dieser Abschnitt beschreibt die Gründe für eine Erweiterung von Query Abfragen im Bezug auf Datums- und Zeitwerte. In diesem Zusammenhang werden die existierenden Datentypen für Datums- und Zeitwerte beschrieben.

#### 7.1.1 Vorüberlegungen und Rahmenbedingungen

In RDF Dokumenten werden unter anderem auch XSD<sup>1</sup> Datentypen verwendet. Zur Darstellung von Datums- und Zeitwerten gibt es in XSD die Datentypen „Date“, „Time“ und „DateTime“. Diese drei Datentypen verlangen eine exakte Angabe zu dem Datum und zu der Zeit.

Der Datentyp „Date“ wird zur Darstellung von einem Datumswert verwendet. Dabei ist dieser Datentyp in der Form „YYYY-MM-DD“ spezifiziert. Durch den Platzhalter „YYYY“ wird das Jahr, durch „MM“ wird das Monat und durch „DD“ wird der Tag angegeben. Das Zeichen (-) dient als Trennzeichen zwischen den einzelnen Werten. Alle diese Informationen müssen angegeben werden, wenn dieser Datentyp verwendet wird. Im Listing 7.1 wird ein Beispiel für die Verwendung des „Date“ Datentyps gezeigt. (W3Schools, 2007)

---

1. XML Schema Definition

Um Zeitwerte darzustellen gibt es den Datentyp „Time“. Spezifiziert ist der Datentyp in der Form „hh:mm:ss“. Dabei wird durch „hh“ die Stunde, durch „mm“ die Minuten und durch „ss“ die Sekunden angegeben. Durch das Zeichen (:) werden die einzelnen Zeitwerte von einander getrennt. Wenn dieser Datentyp verwendet wird, dann müssen alle diese Informationen angegeben werden. Im Listing 7.1 wird die Verwendung dieses Datentyps dargestellt. (W3Schools, 2007)

Um Datums- und Zeitwerte gemeinsam darzustellen existiert der Datentyp „DateTime“. Spezifiziert ist der Datentyp in der Form „YYYY-MM-DDThh:mm:ss“. Hierbei wird durch „YYYY“ das Jahr, durch „MM“ das Monat und durch „DD“ der Tag angegeben. Mit dem Zeichen (-) werden die Datumswerte voneinander getrennt. Durch die Verwendung des Platzhalters „T“ wird angezeigt, dass die Sektion für das Zeitformat beginnt. Bei den Zeitwerten werden durch „hh“ die Stunden, durch „mm“ die Minuten und durch „ss“ die Sekunden angegeben. Das Zeichen (:) wird zur Trennung der einzelnen Zeitwerte verwendet. Wie bei den Datentypen „Date“ und „Time“ müssen auch hier alle genannten Informationen angegeben werden um den „DateTime“ Datentyp zu verwenden. Im Listing 7.1 wird gezeigt wie dieser Datentyp verwendet wird. (W3Schools, 2007)

Bei allen drei beschriebenen Datentypen kann auch die Zeitzone angegeben werden. Dieser Wert wird in UTC<sup>2</sup> Zeit angegeben. Die UTC Zeit wird unter anderem auch für die Synchronisierung von Computersystemen in Internet verwendet. Die Zeitzone wird durch „Z“ angegeben. Es kann aber auch die lokale Zeit verwendet werden, indem „+hh:mm“ bzw. „-hh:mm“ verwendet wird. Durch (+) oder (-) wird die Differenz zur Weltzeit angegeben. Im Listing 7.1 wird die Verwendung dieser drei Möglichkeiten gezeigt. Die Angabe von „-06:00“ bedeutet, dass von der Ortszeit sechs Stunden abgezogen werden müssen um die Weltzeit zu erhalten. Wenn „+06:00“ angegeben wird, dann bedeutet dies das zur Ortszeit sechs Stunden addiert werden müssen um die Weltzeit zu erhalten. (W3Schools, 2007; Wolf und Wicksteed, 1997; Microsoft, 2000)

```
1 <!-- Verwendung des Datentyps Date -->
2 <xs:element name="start" type="xs:date" />
3 <start>1805-12-01</start>
4
5 <!-- Verwendung des Datentyps Time -->
6 <xs:element name="start" type="xs:time" />
7 <start>12:59:11</start>
8
```

2. Universal Time Coordinate, Abkürzung für die koordinierte universale Weltzeit

```
9 <!-- Verwendung des Datentyps DateTime -->
10 <xs:element name="startdate" type="xs:dateTime"/>
11 <startdate>1951-04-28T09:30:40</startdate>
12
13 <!-- Verwendung der Zeitzone mit Z -->
14 <startdate>2002-05-30T09:30:10Z</startdate>
15
16 <!-- Verwendung der Zeitzone mit (-) -->
17 <startdate>2002-05-30T09:30:10-06:00</startdate>
18
19 <!-- Verwendung der Zeitzone mit (+) -->
20 <startdate>2002-05-30T09:30:10+06:00</startdate>
```

**Listing 7.1:** XSD Datentypen „Date“, „Time“, „DateTime“ und die Verwendung der Zeitzone (Quelle: W3Schools, 2007)

Wenn man einen dieser Datentypen bei Query Abfragen verwendet, dann müssen die Angaben hier präzise gemacht werden. Oft kommt es aber vor, dass man das Datum oder auch die Zeit nicht so exakt angeben kann, wie es bei diesen Datentypen gefordert wird. Dies kann unterschiedliche Gründe haben, wie z.B. dass das genaue Datum bzw. die genaue Zeit nicht bekannt ist. Oft weiß man aber auch nur einen Zeitintervall oder einen Datumsbereich. Dies kann unter anderem dann vorkommen, wenn z.B. ein bestimmtes Dokument gesucht wird, welches in den letzten drei Tagen editiert worden ist. In solch einem Fall sind die zuvor beschriebenen Datentypen nicht brauchbar. Diese Problemstellung war der Anlass dafür, eine Erweiterung von Query Abfragen für Datums- und Zeitwerte zu implementieren. Durch diese Erweiterung ist es dann möglich, dass Datums- und Zeitwerte ungenau angegeben werden können. Es soll hierbei möglich sein, dass man den Datums- und Zeitwerte als Intervall angeben kann. Des Weiteren soll die Möglichkeit gegeben werden, dass einzelne Fragmente eines Datums- oder Zeitwertes ausgelesen werden können. Solche Abfragen sind sehr nützlich wenn man z.B. eine Abfrage nur auf den Jahresbereich eingrenzen will.

## 7.2 Anforderungsanalyse

In RDF Dokumenten werden Datumswerte sowie auch Zeitwerte durch die Verwendung des Datentyps „xsd:DateTime“ dargestellt. Mit der Darstellung von unpräzisen Datumswerten sowie von unpräzisen Zeitwerten befassen sich auch andere Initiativen wie z.B. „Cyc“ bzw. „OpenCyc“. Dabei erforschen diese Initiativen auch das Modellieren von semantischen Relationen und Nähe.

„Cyc“ ist eine Wissensdatenbank, welche bereits seit 1984 laufend weiterentwickelt wird. Die Inhalte werden als logische Aussagen in der Ontologiesprache „CycL“ formuliert. Herausgegeben wird „Cyc“ von Cycorp Inc., die Open-Source Variante von „Cyc“ ist „OpenCyc“. (Cycorp, Inc., 2007)

Die Bibliotheken von „Jena+ARQ“ sollen so erweitert werden, dass eine Query Abfrage von unpräzisen bzw. unscharfen Datums- und Zeitwerten möglich ist. Diese „neu“ entwickelte Bibliothek soll dann anschließend im „SemanticLIFE“ Projekt eingebunden und verwendet werden. Dadurch, dass diese Erweiterung als eine „zusätzliche“ Bibliothek implementiert wird, ist es leicht, diese Bibliothek auch in andere Projekt einzubinden und zu verwenden.

### 7.2.1 Anwendungsfälle

Im Gegensatz zu dem präzisen Datentyp „xsd:DateTime“ soll die neue Erweiterung Query Abfragen erlauben, welche ungenauer formuliert werden können. Anwendungsfälle hierzu können sowohl Datumswerte, als auch Zeitwerte, ungenau abfragen. Nachfolgend werden einige mögliche Anwendungsfälle beschrieben, für welche die Angabe von ungenauen Datumswerten benötigt werden.

- Es sollen alle Dokumente ausgegeben werden, welche immer an jedem 3. Montag erstellt worden sind. Bei dieser Abfrage soll das Jahr sowie auch das Monat und die Zeitwerte nicht beachtet werden. Dies kann sein, wenn man z.B. weiß, dass man immer an jedem 3. Montag im Monat ein Meeting hat, bei welchem ein Protokoll erstellt wird.
- Alle Dokumente sollen angezeigt werden, welche immer am 21. eines Monats bearbeitet wurden, und dies in einer Zeitspanne von März bis Juni. Das Jahr sowie die Zeitwerte sollen bei dieser Abfrage keinen Einfluß haben.
- Es soll geprüft werden, ob ein bestimmter Tag ein Mittwoch war. Bei dieser Abfrage soll das Jahr, das Monat und auch die Zeitwerte nicht beachtet werden.
- Alle Dokumente sollen ausgegeben werden, welche zwischen dem Jahr 2001 und 2002 erstellt worden sind. Dabei sollen das Monat und der Tag sowie die Zeitangaben nicht berücksichtigt werden. Solch eine Abfrage kann z.B. durchgeführt werden, wenn man weiß, dass man einen Kurs in diesem Zeitraum besucht hat. Analog hierzu soll eine ähnliche Abfrage auch für den Zeitraum zwischen zwei Monaten möglich sein. Des Weiteren soll eine ähnlich Abfrage durchführbar sein, bei welcher der Zeitbereich mit zwei Wochentage angegeben wird.
- Bei einem Dokument soll geprüft werden ob dieses an einem Dienstag erstellt worden ist. Dabei soll das Jahr, das Monat und der konkrete Tag sowie die Zeitwerte keinen Einfluß haben. Eine solche Abfrage ist sinnvoll, wenn man

wissen möchte, ob ein bestimmtes Dokument an einem Donnerstag geändert worden ist.

- Von einem aktuellen Objekt sollen nur die Werte für das Datum ausgelesen werden. Dabei sollen die Zeitwerte unberücksichtigt bleiben. Dies ist von Vorteil, wenn man alle Dokumente eines bestimmten Tages haben möchte.
- Von einem Datum soll nur der Wert des Jahres ausgelesen werden, unabhängig vom Monat und vom Tag sowie von den Zeitwerten. Dies ist von Vorteil, wenn man mit dem Jahr weitere Vergleiche durchführen möchte. Analog hierzu soll eine solche Abfrage auch für den Wert des Monats und des Tages durchführbar sein.

Die nachfolgenden Anwendungsfälle beschreiben einige mögliche Abfragen, bei welchen ungenaue Zeiangaben benötigt werden.

- Alle Dokumente sollen ausgegeben werden, welche zwischen 14 Uhr und 17 Uhr erstellt worden sind. Die Minuten und die Sekunden sowie die Zeitzone und die Datumswerte sollen bei dieser Abfrage keinen Einfluss haben. Es ist sinnvoll, wenn bei einer solchen Abfrage das Stundenformat „HH12“ oder „HH24“ angegeben werden kann. Wenn das Stundenformat „HH12“ verwendet wird, dann muss auch „AM“<sup>3</sup> bzw. „PM“<sup>4</sup> angegeben werden.
- Es sollen alle Dokumente ausgegeben werden, welche zwischen zwei Minutenwerten geändert wurden. Dabei soll der Stundenwert und der Minutenwert, sowie die Zeitzone und die Datumswerte, nicht berücksichtigt werden. Analog hierzu soll eine solche Abfrage auch für die Sekundenwerte durchführbar sein.
- Von einem aktuellen Objekt sollen nur die Zeitwerte ohne die Zeitzone ausgelesen werden. Dabei sollen die Datumswerte unberücksichtigt bleiben. Dies ist von Vorteil, wenn man alle Dokumente von einem bestimmten Zeitpunkt haben möchte.
- Von einem Zeitwert soll nur der Stundenwert ausgelesen werden, unabhängig von den Minuten und Sekunden sowie von der Zeitzone und vom Datumswert. Dies ist von Vorteil, wenn man mit dem Stundenwert weitere Verleiche durchführen möchte. Analog hier soll eine solche Abfrage auch für den Wert der Minuten und der Sekunden durchführbar sein.
- Bei einem Dokument soll geprüft werden, ob dieses eine bestimmte Zeitzone besitzt. Dabei sollen die Stunden, die Minuten und die Sekunden sowie die Datumswerte keinen Einfluß haben. Eine solche Abfrage ist sinnvoll, wenn man wissen möchte, ob ein bestimmtes Dokument in der Zeitzone „Z“ geändert wurde.
- Es soll von einem Objekt nur die Zeitzone ausgegeben werden. Dies soll unabhängig von den anderen Zeitwerten und vom Datumswert geschehen.

---

3. ante meridiem, im deut. Vormittag

4. post meridiem, im deut. Nachmittag

Durch die Kombination der verschiedenen Anwendungsfälle ist ein weites Spektrum von möglichen Abfragen abgedeckt.

### 7.3 Konzeption & Software Design

In den nachfolgenden Abschnitten wird beschrieben, wie die Anforderungen aus dem Abschnitt 7.2 implementiert werden. Dabei wird auch auf die Design Entscheidungen eingegangen.

#### 7.3.1 Erweiterung von ARQ

In ARQ gibt es verschiedene Möglichkeiten für Erweiterungen. Eine Möglichkeit sind die „Filter Funktionen“. Die weiteren Möglichkeiten um ARQ zu erweitern wird bei Hewlett-Packard Development Company (2007a) beschrieben. Durch Filter Funktionen sollen die jeweiligen Argumente evaluiert werden. Eine Filter Funktion soll nicht auf einen Graphen zugreifen, und es sollen keine unterschiedlichen Werte für das gleiche Argument zurückgeliefert werden. Für solch eine Funktionalität eignen sich „Property Funktionen“ besser. (Hewlett-Packard Development Company, 2007a,c)

Applikationen können SPARQL Filter Funktionen zur Query Engine hinzufügen, indem das Interface `com.hp.hpl.jena.sparql.function.Function` in der jeweiligen Klasse implementiert wird. Durch diese Klasse wird die Funktionalität der „neu“ entwickelten Filter Funktion zur Verfügung gestellt. Es ist aber komfortabler und vor allem leichter mit der abstrakten Klasse für die Spezifizierung von Argumenten zu arbeiten. `com.hp.hpl.jena.sparql.function.FunctionBase1` ist ein Beispiel für eine solche abstrakte Klasse. Diese abstrakte Klasse wird von Filter Funktionen implementiert, welche ein Argument haben. Filter Funktionen haben keine fixe Anzahl von Argumenten. Wenn während der Ausführung einer Filter Funktion ein Fehler auftritt, dann wird dieser auf der Konsole ausgegeben. Aufgrund der Definition der Interfaces für Filter Funktionen kann keine Exception<sup>5</sup> weiter geleitet werden. (Hewlett-Packard Development Company, 2007a)

Um eine entwickelte Filter Funktion verwenden zu können, muss sie entweder registriert werden, oder sie wird dynamisch geladen. Um eine Funktion zu registrieren wird sie von der Applikation in der „Function Registry“ installiert. Die „Function Registry“ ist eine Abbildung von einer URI zu einer „Factory Klasse“ für eine Funktion. Wenn eine Filter Funktion von einer Query ausgeführt wird, dann wird eine neue Instanz erzeugt. Eine weitere allgemeinere Möglichkeit ist, dass eine spezifische Klasse für eine Filter Funktion registriert wird. Filter Funktionen können aber auch dynamisch geladen werden.

---

5. Fehler während der Ausführung eines Programms

Dieser Mechanismus wird auch von der ARQ Bibliothek verwendet. Der Namespace für die ARQ Bibliothek ist `<http://jena.hpl.hp.com/function#>.`, inklusive dem `(.)` am Ende. Um eine Filter Funktion dynamisch zu laden wird das URI Schema `java:` verwendet. Durch dieses URI Schema wird der Klassenname der Filter Funktion angegeben. Die URI `<java:test.myFilterFunction.>` ist hierfür ein Beispiel. Auch hier darf der `(.)` am Ende nicht vergessen werden. Durch diese URI wird angegeben, das sich die Filter Funktion im Package `test.myFilterFunction` befindet. (Hewlett-Packard Development Company, 2007a,c)

### 7.3.2 Aufbau der Implementierung

Für die Implementierung der Filter Funktionen werden die zwei Packages benötigt. Ein Package `com.eder.datetime.filter.functions` wird für die Implementierung der Filter Funktionen verwendet. Für die Implementierung der Testfälle wird das Package `com.eder.datetime.filter.unittests` benötigt. Abbildung 7.1 zeigt alle Packages welche für die Implementierung benötigt werden.



**Abbildung 7.1:** Struktur der Packages

Im Package `com.eder.datetime.filter.functions` werden alle Klassen für die Filter Funktionen implementiert. Hierfür werden auch noch Hilfsklassen benötigt, welche dann im Package `com.eder.datetime.filter.functions.util` implementiert werden. Diese Klassen werden an unterschiedlichen Codestellen im Package der Filter Funktionen verwendet, daher ist es sinnvoll diese Klasse in einem separaten Package zu kapseln. Dies hat den Vorteil, dass bei Änderungen nur eine zentrale Code-stelle editiert werden muss.

Das Package `com.eder.datetime.filter.unittests` beinhaltet alle Klassen, welche für das Testen der Funktionalität der Filter Funktionen benötigt werden. Für einen besseren Überblick über die Testklassen existieren in diesem Package noch die folgenden Packages:

- `com.eder.datetime.filter.unittests.util`
- `com.eder.datetime.filter.unittests.test.functions`
- `com.eder.datetime.filter.unittests.test.utils`

Für die Durchführung der Tests werden auch Hilfsklassen benötigt, welche im Package `com.eder.datetime.filter.unittests.util` implementiert werden. Alle Testklassen, welche die Funktionalität der Filter Funktionen testen, befinden sich im Package `com.eder.datetime.filter.unittests.test.functions`. Im Package `com.eder.datetime.filter.unittests.test.utils` befinden sich alle anderen Testklassen.

Durch diese Struktur erfolgt eine Kapselung der Klassen nach ihrer Funktionalität. Im Kapitel 8 werden die Klassen dieser Packages mit ihrer Funktionalität beschrieben.

Es werden keine zusätzlichen Bibliotheken verwendet außer jene, welche bereits von ARQ verwendet werden. Für das Logging von Informations-, Warnungs- oder Fehlermeldungen wird innerhalb von ARQ das Framework „log4j“<sup>6</sup> verwendet. Dieses Framework stellt Logging Levels zur Verfügung, d.h. je nachdem mit welchem Logging Level eine Applikation läuft, werden unterschiedliche Nachrichten geloggt. Die wichtigsten Logging Levels sind folgende:

- **ERROR:** Bei diesen Level werden alle Fehlermeldungen ausgegeben.
- **INFO:** Wenn dieses Level eingestellt ist, dann werden allgemeine Informationen ausgegeben. Zusätzlich werden auch die Nachrichten vom „ERROR“ Level ausgegeben.
- **DEBUG:** Durch die Angabe von diesem Level werden Debugging Nachrichten ausgegeben. Zusätzlich werden auch die Nachrichten vom „ERROR“ und vom „INFO“ Level ausgegeben.

Abbildung 7.1 zeigt auch die Datei „log4j.properties“. Dies ist eine Properties Datei, über welche das Logging Level steuerbar ist.

Zum Kompilieren des Quellcodes sowie für die Erstellung der Jar Datei wird ANT<sup>7</sup> verwendet. ANT wird über eine sogenannte „Build Datei“ gesteuert, wobei diese standardmäßig „build.xml“ heisst. In dieser Datei werden die benötigten Kommandos für das Kompilieren des Quellcodes sowie für die Erzeugung der Jar Datei geschrieben. Abhängig von der definierten Reihenfolge in der „Build Datei“ werden die einzelnen Aufgaben ausgeführt. Durch ANT ist es möglich, viele verschiedene Aufgaben automatisiert ablaufen zu lassen.

### 7.3.3 Anwendung

Nach der Entwicklungs- und Testphase wird die praktische Anwendung der implementierten Filter Funktionen getestet. Die Funktionalität dieser Funktionen wird als Jar Bibliothek zur Verfügung gestellt. Dadurch soll die Einbindung dieser Bibliothek in eine

---

6. Siehe dazu auch <http://logging.apache.org/log4j/index.html> - Zugriffsdatum: 16.12.2007

7. Another Neat Tool, siehe dazu auch <http://ant.apache.org/manual/index.html> - Zugriffsdatum: 16.12.2007

Applikation erleichtert werden. Für diese Tests wird ein Testtool mit einer graphischen Benutzeroberfläche implementiert. Über dieses Tool können die Filter Funktionen innerhalb einer Applikation getestet werden. Dieses Testtool stellt noch weitere Funktionen zur Verfügung sowie das Laden von einem RDF Model aus einer Datei. Abschließend wird die entwickelte Bibliothek in eine bestehende Applikation eingebunden.

# Kapitel 8

## Praxis: Implementierung & Evaluierung

Der erste Abschnitt dieses Kapitels befasst sich mit den entwickelten Filter Funktionen. Dabei wird die Funktionalität sowie das Verhalten der Funktionen im Fehlerfall beschrieben. Bei den Beschreibungen wird jeweils ein Beispiel dafür angegeben, wie die jeweilige Filter Funktion verwendet werden kann. Des Weiteren werden auch die Hilfsklassen beschrieben, auf welche die implementierten Filter Funktionen zugreifen.

Im darauffolgenden Abschnitt wird die Evaluierung der Implementierung beschrieben. Dazu zählt eine Beschreibung der Testmethode sowie des Testverlaufs. Die Verwendung der Filter Funktionen in einem anderem Projekt ist auch getestet worden. Auf die Verwendung der Filter Funktionen in diesem Projekt, wird im abschließendem Abschnitt eingegangen.

### 8.1 Implementierung

Die Abschnitte dieses Kapitels befassen sich mit der Beschreibung der implementierten Filter Funktionen und der dafür benötigten Hilfsklassen. Für die Implementierung wurde Java<sup>1</sup> in der Version 1.5 verwendet. Als Entwicklungsumgebung wird „Eclipse“<sup>2</sup> verwendet

#### 8.1.1 Beschreibung der Filter Funktionen

Dieser Abschnitt beschreibt die Funktionalität der implementierten Filter Funktionen, welche im Package `com.eder.datetime.filter.functions` implementiert sind. Während dieser Beschreibungen wird angegeben, welche Parameterwerte für welche Filter Funktion gültig sind. Wenn eine Filter Funktion mit einem ungültigen Parameterwert aufgerufen wird, dann wird eine Fehlermeldung ausgegeben.

---

1. <http://java.sun.com/> - Zugriffsdatum: 17.12.2007

2. <http://www.eclipse.org/> - Zugriffsdatum: 17.12.2007

```

1 PREFIX afn:<java:com.eder.datetime.filter.functions.>
2 PREFIX dc: <http://purl.org/dc/elements/1.1/>
3
4 SELECT ?x ?date ?title ?description ?identifier
5 WHERE
6   { ?x   dc:date           ?date ;
7         dc:description    ?description ;
8         dc:title          ?title ;
9         dc:identifier     ?identifier .
10      FILTER ( ?identifier < afn:getYearValue(?date) )
11   }

```

**Listing 8.1:** Beispiel für eine Query mit einer Filter Funktion

Listing 8.1 zeigt eine Query, bei welcher eine der implementierten Filter Funktionen verwendet wird. Diese Query gibt von einem Modell alle Ressourcen aus, welche die Eigenschaft `dc:date`, `dc:title`, `dc:description` und `dc:identifiziert` haben. Das Ergebnis dieser Query wird noch weiter eingeschränkt durch die Codezeile `FILTER (?identifier<afn:getYearValue(?date))`. Dadurch werden nur Ressourcen ausgegeben, bei denen der Wert von `?identifier` kleiner dem Jahreswert `dc:date` ist. Der Pfad zu den implementierten Filter Funktionen wird durch den Präfix `afn:<java:com.eder.datetime.filter.functions.>` angegeben. Nachfolgend wird bei den Beschreibungen nur mehr die Codezeile für den Aufruf der Filter Funktion angegeben. Durch die Kombination von unterschiedlichen Filter Funktionen kann das Ergebnis einer Query noch weiter eingeschränkt werden.

```

1 PREFIX afn: <java:com.eder.datetime.filter.functions.>
2 PREFIX dc: <http://purl.org/dc/elements/1.1/>
3
4 SELECT ?x ?date ?title ?description ?identifier
5 WHERE
6   { ?x   dc:date           ?date ;
7         dc:description    ?description ;
8         dc:title          ?title ;
9         dc:identifier     ?identifier .
10      FILTER ( ( ?date = afn:rangeYear(?date, 2007, 2008)) &&
11              ( ?date = afn:rangeMonth(?date, "oct", "nov")))
12   }

```

**Listing 8.2:** Beispiel für eine Query mit zwei Filter Funktion

Es besteht auch die Möglichkeit, dass die unterschiedlichen Filter Funktionen miteinander kombiniert werden können. Das Listing 8.2 zeigt ein Beispiel für eine Kombination von zwei Filter Funktionen. Hierbei wird zuerst geprüft, ob das „xsd:DateTime“ Objekt zwischen den Jahren „2007“ und „2008“ ist. Anschließend wird noch geprüft ob dieses Objekt zwischen den Monaten „Oktober“ und „November“ liegt. Da die zwei Filter Funktionen durch && miteinander verknüpft sind, müssen beide Bedingungen zutreffen.

Bei den Beschreibungen wird des Öfteren erwähnt, dass das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert wird. Definiert wird dieser Wert wird in der Hilfsklasse `FilterFunctionsConstants.java`. Dieses Datum wurde deshalb so gewählt, da dies ein unrealistisches Datum ist. Des Weiteren werden bei den Beschreibungen auch die Abkürzungen für Datums- und Zeitwerte verwendet, welche im Abschnitt 7.1 bereits verwendet wurden.

#### 8.1.1.1 `getYearValue(xsd:DateTime obj)`

Diese Filter Funktion liefert von einem „xsd:DateTime“ Objekt in einem Modell den Wert des Feldes „YYYY“. Mit dem erhaltenen Wert des Jahres kann anschließend weiter gearbeitet werden. Diese Filter Funktion liefert den Wert als „xsd:Integer“ Objekt zurück. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des `<where>` Elements einer Query sieht wie folgt aus:

```
FILTER (2010 < afn:getYearValue(?date))
```

#### 8.1.1.2 `getMonthValue(xsd:DateTime obj)`

Als Eingabewert erhält diese Filter Funktion ein „xsd:DateTime“ Objekt eines Modells. Von diesem Objekt wird dann der Wert des Feldes „MM“ zurückgegeben. Dieser Wert wird als „xsd:Integer“ Objekt zurückgeliefert. Mit diesem Wert des Monats kann anschließend weiter gearbeitet werden. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des `<where>` Elements einer Query sieht wie folgt aus:

```
FILTER (6 > afn:getMonthValue(?date))
```

#### 8.1.1.3 `getDayValue(xsd:DateTime obj)`

Mit dieser Filter Funktion wird von einem „xsd:DateTime“ Objekt in einem Modell der Wert des Feldes „DD“ zurückgeliefert. Mit dem erhaltenen Wert des Tages kann anschließend weiter gearbeitet werden. Das Ergebnis wird als „xsd:Integer“ Objekt zurück

geliefert. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (15 < afn:getDayValue(?date))
```

#### 8.1.1.4 getDateValue(xsd:DateTime obj)

Ausgehend von einem „xsd:DateTime“ Objekt eines Modells liefert diese Filter Funktion die Werte der Felder „YYYY“, „MM“ und „DD“. Die Werte werden mit dem Zeichen (-) getrennt. Das Resultat wird als „xsd:String“ Objekt zurückgeliefert. Mit diesem Datumswert kann anschließend weiter gearbeitet werden. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER ("2009-1-1" = afn:getDateValue(?date))
```

Beim zurückgelieferten Datumswert diese Filter Funktion wird der Tag sowie das Monat ohne führende Null angegeben. Dies muss beim Aufruf in der Query beachtet werden.

#### 8.1.1.5 getHoursValue(xsd:DateTime obj)

Diese Filter Funktion liefert von einem „xsd:DateTime“ Objekt in einem Modell den Wert des Feldes „hh“. Mit dem erhaltenen Wert des Jahres kann anschließend weiter gearbeitet werden. Diese Filter Funktion liefert den Wert als „xsd:Integer“ Objekt zurück. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (13 > afn:getHoursValue(?date))
```

#### 8.1.1.6 getMinutesValue(xsd:DateTime obj)

Als Eingabewert erhält diese Filter Funktion ein „xsd:DateTime“ Objekt eines Modells. Von diesem Objekt wird dann der Wert des Feldes „mm“ zurückgegeben. Dieser Wert wird als „xsd:Integer“ Objekt zurückgeliefert. Mit diesem Wert des Monats kann anschließend weiter gearbeitet werden. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (45 < afn:getMinutesValue(?date))
```

#### 8.1.1.7 *getSecondsValue(xsd:DateTime obj)*

Mit dieser Filter Funktion wird von einem „xsd:DateTime“ Objekt in einem Modell der Wert des Feldes „ss“ zurückgeliefert. Mit dem erhaltenen Wert des Tages kann anschließend weiter gearbeitet werden. Das Ergebnis wird als „xsd:Integer“ Objekt zurückgeliefert. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?identifizier < afn:getSecondsValue(?date))
```

#### 8.1.1.8 *getTimeValue(xsd:DateTime obj)*

Ausgehend von einem „xsd:DateTime“ Objekt eines Modells liefert diese Filter Funktion die Werte der Felder „hh“, „mm“ und „ss“. Die Werte werden mit dem Zeichen (:) getrennt. Das Resultat wird als „xsd:String“ Objekt zurückgeliefert. Mit diesem Zeitwert kann anschließend weiter gearbeitet werden. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER ("14:8:11" = afn:getTimeValue(?date))
```

Beim zurückgelieferten Zeitwert dieser Filter Funktion wird der Wert der Stunde, des Monats und der Sekunden ohne führende Null angegeben. Dies muss beim Aufruf in der Query beachtet werden.

#### 8.1.1.9 *getTimezoneValue(xsd:DateTime obj)*

Diese Filter Funktion liefert von einem „xsd:DateTime“ Objekt in einem Modell den Wert der Zeitzone. Mit dem erhaltenen Wert des Jahres kann anschließend weiter gearbeitet werden. Diese Filter Funktion liefert den Wert als „xsd:String“ Objekt zurück. Ein gültiger Parameterwert muss vom Typ „xsd:DateTime“ sein. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER ("+06:00" = afn:getTimezoneValue(?date))
```

#### 8.1.1.10 *filterEqualTimezone(xsd:DateTime obj, String timezone)*

Als Eingabewert erhält diese Filter Funktion ein „xsd:DateTime“ Objekt eines Modells und einen String für die Zeitzone. Anschließend wird geprüft, ob dieses Objekt die gesuchte Zeitzone enthält. Wenn das „xsd:DateTime“ Objekt diese Zeitzone enthält,

dann wird dieses Objekt auch wieder zurückgegeben. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Dieser Wert wird als „xsd:DateTime“ Objekt zurückgeliefert. Mit diesem Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Gültige Parameterwerte für diese Filter Funktion sind ein Objekt vom Typ „xsd:DateTime“ und ein String für die gesuchte Zeitzone. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:filterEqualTimezone(?date, "-06:00"))
```

Bei der Angabe der Zeitzone muss diese den Wert „Z“ haben, oder sie muss mit dem Zeichen (+) bzw. (-) beginnen. Des Weiteren darf die Zeitzone bei „Z“ nicht mehr als eine Stelle haben. Wenn die Zeitzone mit (+) oder mit (-) beginnt, dann darf die Zeitzone nicht länger als sechs Stellen sein.

#### 8.1.1.11 *filterUnequalTimezone(xsd:DateTime obj, String timezone)*

Mit dieser Filter Funktion wird geprüft, ob ein „xsd:DateTime“ Objekt in einem Modell die gesuchte Zeitzone nicht enthält. Wenn das „xsd:DateTime“ Objekt diese Zeitzone nicht enthält, dann wird dieses Objekt auch wieder zurückgegeben. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Mit dem erhaltenen Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Das Ergebnis wird als „xsd:DateTime“ Objekt zurückgeliefert. Ein Objekt vom Typ „xsd:DateTime“ und ein String für die Zeitzone sind hier gültige Parameterwerte. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:filterUnequalTimezone(?date, "-06:00"))
```

Bei der Angabe der Zeitzone muss diese den Wert „Z“ haben, oder sie muss mit dem Zeichen (+) bzw. (-) beginnen. Des Weiteren darf die Zeitzone bei „Z“ nicht mehr als eine Stelle haben. Wenn die Zeitzone mit (+) oder mit (-) beginnt, dann darf die Zeitzone nicht länger als sechs Stellen sein.

#### 8.1.1.12 *filterEqualWeekday(xsd:DateTime obj, String weekday)*

Ausgehend von einem „xsd:DateTime“ Objekt eines Modells prüft diese Filter Funktion ob dieses Objekt an einem bestimmten Wochentag ist. Wenn das „xsd:DateTime“ Objekt an dem gesuchten Wochentag ist, dann wird dieses wieder zurückgeliefert. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Das Resultat wird als „xsd:DateTime“ Objekt zurückgeliefert. Mit diesem Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Gültige Parameter für diese Filter Funktion

sind ein Objekt vom Typ „xsd:DateTime“ und ein String für den gesuchten Wochentag. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:filterEqualWeekday(?date, "mon"))
```

Der Wochentag muss in englischer Sprache sein, und es müssen mindestens die ersten drei Buchstaben des Wochentages angegeben werden. Die Angabe des Wochentages ist nicht „case sensitiv“. Wenn ein anderer Wert als ein Wochentag angegeben wird, dann wird eine Fehlermeldung ausgegeben.

#### 8.1.1.13 filterUnequalWeekday(xsd:DateTime obj, String weekday)

Diese Filter Funktion prüft, ob „xsd:datetime“ Objekt in einem Modell nicht an einem bestimmten Wochentag ist. Wenn das „xsd:DateTime“ Objekt nicht an dem gesuchten Wochentag ist, dann wird dieses wieder zurückgeliefert. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Mit dem erhaltenen Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Diese Filter Funktion liefert ein „xsd:DateTime“ Objekt zurück. Gültige Parameter für diese Filter Funktion sind ein Objekt vom Typ „xsd:DateTime“ und ein String für den gesuchten Wochentag. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?date=afn:filterUnequalWeekday(?date, "weDnesD"))
```

Der Wochentag muss in englischer Sprache sein und es müssen mindestens die ersten drei Buchstaben des Wochentages angegeben werden. Die Angabe des Wochentages ist nicht „case sensitiv“. Wenn ein anderer Wert als ein Wochentag angegeben wird, dann wird eine Fehlermeldung ausgegeben.

#### 8.1.1.14 rangeYear(xsd:DateTime obj, Integer year, Integer year)

Als Eingabewert erhält diese Filter Funktion ein „xsd:datetime“ Objekt eines Modells und zwei Integer Werte als Suchbereich. Anschließend wird geprüft, ob der Wert des Jahres dieses Objekt innerhalb des Suchbereichs liegt. Wenn das „xsd:datetime“ Objekt innerhalb des Suchbereichs ist, dann wird dieses Objekt auch wieder zurückgegeben. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Dieser Wert wird als „xsd:datetime“ Objekt zurückgeliefert. Mit diesem Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Gültige Parameterwerte für diese Filter Funktion sind ein Objekt vom Typ „xsd:datetime“ und zwei Integer Werte für

den gesuchten Jahresbereich. Der Aufruf dieser Filter Funktion innerhalb des `<where>` Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:rangeYear(?date, 2006, 2007))
```

Bei der Angabe des Suchbereichs für die Jahreszahl wird geprüft, ob diese gültige Werte besitzen. Die Untergrenze darf nicht kleiner sein als das Jahr „1000“ und die Obergrenze darf nicht höher sein als das Jahr „9999“. Diese Unter- und Obergrenze wurde deshalb so gewählt, damit sichergestellt ist, dass der Jahreswert genau vier Stellen lang ist.

#### 8.1.1.15 *rangeMonth(xsd:DateTime obj, String month, String month)*

Mit dieser Filter Funktion wird geprüft, ob ein „xsd:datetime“ Objekt in einem Modell innerhalb von zwei Monaten liegt. Wenn das „xsd:datetime“ Objekt innerhalb der zwei Monate ist, dann wird dieses Objekt auch wieder zurückgegeben. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Mit dem erhaltenen Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Das Ergebnis wird als „xsd:datetime“ Objekt zurückgeliefert. Ein Objekt vom Typ „xsd:datetime“ und zwei Strings für die Angabe der Monate sind hier gültige Parameterwerte. Der Aufruf dieser Filter Funktion innerhalb des `<where>` Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:rangeMonth(?date, "Oct", "Dec"))
```

Das Monat muss in englischer Sprache sein, und es müssen mindestens die ersten drei Buchstaben des Monats angegeben werden. Die Angabe des Monats ist nicht „case sensitiv“. Wenn ein anderer Wert als ein Monat angegeben wird, dann wird eine Fehlermeldung ausgegeben.

#### 8.1.1.16 *rangeDay(xsd:DateTime obj, String weekday, String weekday)*

Ausgehend von einem „xsd:DateTime“ Objekt eines Modells prüft diese Filter Funktion, ob dieses Objekt innerhalb von zwei Wochentagen ist. Wenn das „xsd:DateTime“ Objekt innerhalb der angegebenen Wochentage ist, dann wird dieses wieder zurückgeliefert. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Das Resultat wird als „xsd:DateTime“ Objekt zurückgeliefert. Mit diesem Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Gültige Parameter für diese Filter Funktion sind ein Objekt vom Typ „xsd:DateTime“ und zwei Strings für den Suchbereich der Wochentage. Der Aufruf dieser Filter Funktion innerhalb des `<where>` Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:rangeDay(?date, "mon", "wed"))
```

Der Wochentag muss in englischer Sprache sein und es müssen mindestens die ersten drei Buchstaben des Wochentages angegeben werden. Die Angabe des Wochentages ist nicht „case sensitiv“. Wenn ein anderer Wert als ein Wochentag angegeben wird, dann wird eine Fehlermeldung ausgegeben.

#### 8.1.1.17 rangeHours(xsd:DateTime obj, String hour, String hour, String format)

Diese Filter Funktion prüft ob „xsd:datetime“ Objekt in einem Modell innerhalb von zwei Werten von Stunden ist. Wenn das „xsd:DateTime“ Objekt innerhalb dieses Suchbereichs ist, dann wird dieses wieder zurückgeliefert. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Mit dem erhaltenen Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Es wird ein „xsd:DateTime“ Objekt zurückgeliefert. Mit dem letzten Parameter wird entweder das Stundenformat „HH24“ oder „HH12“ angegeben. Wenn das Format „HH12“ verwendet wird, dann muss zusätzlich zum Stundenwert die Angabe von „AM“ bzw. „PM“ gemacht werden. Gültige Parameter für diese Filter Funktion sind ein Objekt vom Typ „xsd:DateTime“, zwei String Werte für den Suchbereich der Stunden und ein String für das Zeitformat. Aufrufe für diese Filter Funktion innerhalb des <where> Elements einer Query sehen wie folgt aus:

```
FILTER (?date=afn:rangeHours(?date, 08, 14, "hh24"))
```

```
FILTER (?date=afn:rangeHours(?date, "9 am", "11 AM", "hh12"))
```

```
FILTER (?date=afn:rangeHours(?date, "11 am", "2 PM", "hh12"))
```

Bei der Angabe der Stundenwerte wird geprüft, ob diese keine negativen Werte enthalten. Wenn das Format „HH24“ angegeben worden ist, dann darf der Stundenwert nicht größer als „24“ sein. Beim „HH12“ Format darf die maximale Stundenanzahl von „12“ nicht überschritten werden. Des Weiteren wird geprüft, ob beim Stundenwert „AM“ bzw. „PM“ angegeben ist. Wenn dies nicht der Fall ist, dann wird eine Fehlermeldung ausgegeben.

#### 8.1.1.18 rangeMinutes(xsd:DateTime obj, Integer minute, Integer minute)

Als Eingabewert erhält diese Filter Funktion ein „xsd:datetime“ Objekt eines Modells und zwei Integer Werte als Suchbereich. Anschließend wird geprüft, ob der Wert der

Minuten dieses Objekt innerhalb dieses Bereichs ist. Wenn das „xsd:datetime“ Objekt innerhalb des Suchbereichs ist, dann wird dieses Objekt auch wieder zurückgegeben. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Dieser Wert wird als „xsd:datetime“ Objekt zurückgeliefert. Mit diesem Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Gültige Parameterwerte für diese Filter Funktion sind ein Objekt vom Typ „xsd:datetime“ und zwei Integer Werte für den gesuchten Minutenbereich. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:rangeMinutes(?date, 10, 30))
```

Bei der Angabe des Suchbereichs für die Minuten wird geprüft, ob diese gültige Werte besitzen. Die Untergrenze darf nicht kleiner sein als „0“ und die Obergrenze darf nicht größer sein als „60“.

#### 8.1.1.19 rangeSeconds(xsd:DateTime obj, Integer second, Integer second)

Mit dieser Filter Funktion wird geprüft, ob ein „xsd:datetime“ Objekt in einem Modell innerhalb eines Suchbereichs ist. Wenn der Wert der Sekunden des „xsd:datetime“ Objekts innerhalb des Suchbereichs ist, dann wird dieses Objekt wieder zurückgegeben. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Mit dem erhaltenen Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Das Ergebnis wird als „xsd:datetime“ Objekt zurück geliefert. Ein Objekt vom Typ „xsd:datetime“ und zwei Integer Werte für den Suchbereich sind hier gültige Parameterwerte. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:rangeSeconds(?date, 25, 50))
```

Bei der Angabe des Suchbereichs für die Sekunden wird geprüft, ob diese gültige Werte besitzen. Die Untergrenze darf nicht kleiner sein als „0“ und die Obergrenze darf nicht größer sein als „60“.

#### 8.1.1.20 rangeDayInWeeks(xsd:DateTime obj, Integer occurrence, String weekday)

Ausgehend von einem „xsd:DateTime“ Objekt eines Modells prüft diese Filter Funktion ob dieses Objekt an einem bestimmten Wochentag ist. Es wird z.B. geprüft, ob das „xsd:DateTime“ Objekt an jedem vierten Sonntag ist. Wenn das „xsd:DateTime“ Objekt an dem gesuchten Tag ist, dann wird dieses „xsd:DateTime“ Objekt wieder zurückgeliefert. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert.

Mit diesem Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Gültige Parameter für diese Filter Funktion sind ein Objekt vom Typ „xsd:DateTime“, ein Integer Wert für das wievielte Vorkommen und ein String für den gesuchten Wochentag. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?date = afn:rangeDayInWeeks(?date, 4, "sun"))
```

Der Wochentag muss in Englischer Sprache sein und es müssen mindestens die ersten drei Buchstaben des Wochentages angegeben werden. Die Angabe des Wochentages ist nicht „case sensitiv“. Wenn ein anderer Wert als ein Wochentag angegeben wird, dann wird eine Fehlermeldung ausgegeben. Es wird auch der Wert für das Vorkommen des Wochentages geprüft. Dieser darf nicht kleiner sein als „1“ und nicht größer als „5“.

#### 8.1.1.21 *rangeDayInMonths(xsd:DateTime obj, Integer day, String month, String month)*

Diese Filter Funktion prüft ob „xsd:datetime“ Objekt in einem Modell an einem bestimmten Tag ist, und ob dieser Tag innerhalb eines Monatsbereichs ist. Wenn das „xsd:DateTime“ Objekt an dem gesuchten Tag ist und innerhalb des Suchbereichs, dann wird dieses wieder zurück geliefert. Andernfalls wird das Standard Datum „0001-01-01T00:00:00Z“ zurückgeliefert. Mit dem erhaltenen Datums- und Zeitwert kann anschließend weiter gearbeitet werden. Diese Filter Funktion liefert ein „xsd:DateTime“ Objekt zurück. Gültige Parameter für diese Filter Funktion sind ein Objekt vom Typ „xsd:DateTime“, ein Integer Wert für den Tag und zwei Strings für den Suchbereich der Monate. Der Aufruf dieser Filter Funktion innerhalb des <where> Elements einer Query sieht wie folgt aus:

```
FILTER (?date=afn:rangeDayInMonths(?date,20,"Oct" "Dec"))
```

Der Monatswert muss in englisch formuliert werden und es müssen mindestens die ersten drei Buchstaben des Monats angegeben werden. Die Angabe des Monats ist nicht „case sensitiv“. Wenn ein anderer Wert als ein Monat angegeben wird, dann wird eine Fehlermeldung ausgegeben. Der Wert für den Tag muss zwischen 1 und 31 liegen, sonst wird ebenfalls eine Fehlermeldung ausgegeben.

### 8.1.2 Beschreibung der Hilfsklassen

In diesem Abschnitt wird die Funktionalität der Hilfsklassen beschrieben. Diese Klassen sind alle im Package `com.eder.datetime.filter.functions.util` implementiert. Die Funktionen wurden jeweils als eine eigene Klasse implementiert, da die

Funktionen an unterschiedlichen Codestellen verwendet werden. Bei einer Änderung muss diese dann nur in einer Klasse durchgeführt werden, was für die Wartung oder Weiterentwicklung von Vorteil ist.

### 8.1.2.1 Klasse für Datums- & Zeitumwandlungen

Für die Implementierung der Filter Funktionen werden verschiedene Datums- und Zeitumwandlungen benötigt. Diese werden öfters in unterschiedlichen Klassen benötigt. Damit diese Funktionalität zentral gekapselt ist, werden alle diese Funktionen in der Klasse `DateConverter.java` implementiert. Diese Klasse implementiert die folgenden Funktionalitäten:

- *int convertDayOfWeek(String dateTime)*: Ausgehend von einem Datums- und Zeitwert liefert diese Funktion den Integer Wert des Wochentags zurück.
- *int convertDayOfWeekInMonth(String dateTime)*: Diese Funktion retourniert von einen Datums- und Zeitwert den Integer Wert des Monats.
- *int convertHoursHH12(String dateTime)*: Ausgehend von einem Datums- und Zeitwert liefert diese Funktion den Stunden Wert im „HH12“ Format.
- *int convertHoursHH24(String dateTime)*: Diese Funktion retourniert von einem Datums- und Zeitwert den Stunden Wert im „HH24“ Format.
- *convertMonth(String month)*: Ausgehend von einem Monat als String konvertiert diese Funktion den String Wert in den Integer Wert des Monats. Der Integer Wert des Monats wird anschließend retourniert.
- *Calendar convertToCalendar(String dateTime)*: Durch diese Funktion wird ein Datums- und Zeitwert in einem Kalender gesetzt. Anschließend wird dieses Kalender Objekt zurück geliefert.
- *convertWeekday(String weekday)*: Ausgehend von einem Wochentag als String konvertiert diese Funktion den String Wert in den Integer Wert des Wochentags. Der Integer Wert des Wochentags wird anschließend retourniert.
- *int convertCommon(String dateString, String format, int result)*: Diese Funktion konvertiert einen String mit dem angegebenen Format in das angegebene Kalenderformat.

Der Eingabe String der Funktionen „`convertDayOfWeek(String dateTime)`“, „`convertDayOfWeekInMonth(String dateTime)`“, „`convertHoursHH12(String dateTime)`“, „`convertHoursHH24(String dateTime)`“ und „`convertToCalendar(String dateTime)`“ muss das Format „`yyyy-MM-dd'T'hh:mm:ss`“ besitzen. Alle hier beschriebenen Funktionen werfen im Fehlerfall eine „`ParseException`“. Diese Exception wird geworfen, wenn eine Formatangabe nicht korrekt ist.

### 8.1.2.2 Klasse für Konstanten

In den Klassen für die Filter Funktionen werden immer wieder konstante Werte benötigt. Diese werden zentral in der Klasse `FilterFunctionsConstants.java` gesetzt. Dies hat den Vorteil, dass wenn die Werte geändert werden sollen, dann muss dies nur zentral an einem Punkt gemacht werden. In dieser Klasse werden folgende Werte gesetzt:

- Werte für Datumsoperationen (Standard Datumswert, Trennzeichen für Datumswerte, Grenzen für Datumsabfragen)
- Wert für Zeitoperationen (Zeitformate für „HH24“ und „HH12“, Trennzeichen für Zeitwerte, Grenzen für Zeitabfragen)
- Werte für Zeitzonenoperationen (Erlaubte Zeichen für Zeitzonen, maximale Länge von Zeitzonen)
- Werte für Datums- und Zeitoperationen (Trennzeichen zwischen einem Datums- und einem Zeitwert)
- Wert für Kalenderoperationen (Standardeinstellung für den Kalender, Standardformate für Datum und Zeit)
- Werte für Fehlermeldungen und Warnungen

### 8.1.2.3 Klasse für die Zeitzone

Um mit der Zeitzone aus einem Datums- und Zeit String effektiv arbeiten zu können, wird die Hilfsklasse `Timezone.java` benötigt. Die Klasse stellt die Funktionalität bereit um aus einem Datums- und Zeit String die Zeitzone heraus zu lesen und zu speichern. Des Weiteren kann über diese Klasse die Zeitzone wieder ausgelesen werden.

### 8.1.2.4 Klasse für Validierungen

Für die Implementierung der Klassen für die Filter werden auch verschiedene Validierungsabfragen benötigt. Diese Validierungen sind in der Klasse `Validate.java` gekapselt. Dadurch sind die verschiedenen Validierungsfunktionen in einer Klasse zentral implementiert. Wenn eine der Validierungsbedingungen nicht erfüllt wird, dann wird eine „Validate Exception“ geworfen. Implementiert ist diese Exception in der Klasse `ValidateException.java`. Folgende Validierungen sind in dieser Klasse implementiert:

- `validateTimezone(String timezone)`: Durch diese Funktion wird geprüft, ob die eingegebene Zeitzone korrekt ist. Eine Zeitzone muss den Wert „Z“ haben, oder sie muss mit dem Zeichen (+) bzw. (-) beginnen. Wenn die Zeitzone den Wert „Z“ hat, darf sie nicht länger als ein Zeichen sein. Sollte die Zeitzone aber mit dem Zeichen (+) oder (-) beginnen, dann darf die maximale Länge von sechs Stellen nicht überschritten werden.

- *validateYear(int year)*: Über diese Funktion wird geprüft, ob der Wert des Jahres korrekt ist. Der Wert des Jahres muss zwischen „1000“ und „9999“ liegen. Diese Werte wurden deshalb so gewählt, da es unrealistisch ist, wenn der Wert für das Jahr außerhalb dieses Bereichs liegt.
- *validateDayValue(int dayValue)*: Diese Funktion prüft, ob der Wert für einen Tag korrekt ist. Dieser Wert muss zwischen „1“ und „31“ sein, da ein Monat nicht weniger als einen Tag und nicht mehr als 31 Tage haben kann.
- *validateHoursHH12(String hoursStr)*: Durch diese Funktion wird geprüft, ob der eingegebene String konform mit dem Zeitformat „HH12“ ist. Der String muss mit „AM“ oder mit „PM“ enden. Des Weiteren darf der Wert der Stunde nicht negativ sein und auch nicht größer „12“ sein.
- *validateHoursHH24(String hoursStr)*: Diese Funktion prüft, ob der Stundenwert nicht negativ ist und auch nicht größer als „24“ ist.
- *validateOccurrence(int occurrence)*: Durch diese Funktion wird geprüft, ob der Wert für das Vorkommen eines Wochentages korrekt ist. Diese Funktion wird verwendet, wenn z.B. geprüft wird, ob ein Objekt genau am 3. Montag vorkommt. Dabei muss dieser Wert zwischen „1“ und „5“ liegen, da ein Monat nicht mehr Wochen haben kann.
- *validateSecondsMinutes(int secMin)*: Diese Funktion prüft den Sekunden- bzw. den Minutenwert. Dabei muss der Sekunden- bzw. Minutenwert zwischen „0“ und „60“ liegen.

## 8.2 Evaluierung

Dieser Abschnitt beschreibt die Evaluierung durch das Testen der Funktionalität der implementierten Filter Funktionen. Des Weiteren wird noch die praktische Verwendung dieser Filter Funktionen beschrieben.

### 8.2.1 Funktionstest

Für das Testen der implementierten Filter Funktionen wird das Framework „JUnit“<sup>3</sup> verwendet. Alle Klassen, welche für die Testphase verwendet werden, befinden sich im Package `com.eder.datetime.filter.unittests`. Der Name einer Testklasse setzt sich aus dem Namen der zu testenden Klasse und dem Wort „Test“ zusammen.

Im Package `com.eder.datetime.filter.unittests.util` sind die Hilfsklassen implementiert, welche während des Verlaufs der Testphase benötigt werden. Da diese Klassen an verschiedenen Codestellen benötigt werden, ist es von Vorteil, wenn

---

3. Siehe dazu auch <http://www.junit.org/> - Zugriffsdatum: 16.12.2007

diese zentral gekapselt sind. Bei einer Änderung muss diese dann nur an einer Code-stelle durchgeführt werden. Diese Hilfsklassen sind folgende:

- *TestConstants.java*: In dieser Klasse befinden sich konstante Werte.
- *TestModel.java*: Durch diese Klasse können verschiedene Models generiert werden.
- *TestQueryExecution.java*: In dieser Klasse wird die Ausführung einer Query sowie die Anzeige des Resultats der Query implementiert.
- *TestResourceProperties.java*: Diese Klasse wird als Parameter für die Generierung eines Models benötigt. Hier werden die Ressourcen und die Eigenschaften für das zu generierende Model abgespeichert.
- *TestUtil.java*: In dieser Klasse werden verschiedene Hilfsfunktionen implementiert, wie z.B. die Ausgabe von Texten.

Für jede Filter Funktion aus Abschnitt 8.1.1 existiert eine Testklasse, welche sich im Package `com.eder.datetime.filter.unittests.test.functions` befindet. In den Testklassen wird das Verhalten der jeweiligen Filter Funktion im fehlerfreien sowie auch im fehlerhaften Fall geprüft.

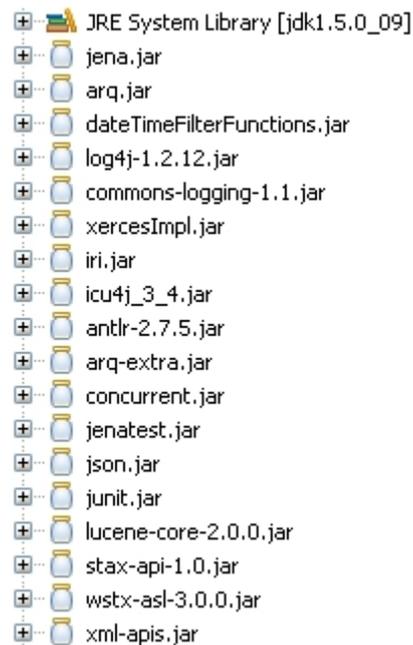
Die Testklassen für die Hilfsklassen aus Abschnitt 8.1.2 befinden sich im Package `com.eder.datetime.filter.unittests.test.utils`. Hier sind auch die Testklassen für das Package `com.eder.datetime.filter.unittests.util` implementiert.

## 8.2.2 Test in einer Applikation

Nach dem Funktionstest werden die entwickelten Filter Funktionen innerhalb einer Applikationen getestet. Um dies durchzuführen, müssen einige Voraussetzungen erfüllt sein. Anschließend wird ein graphisches Tools implementiert um die Filter Funktionen zu testen. Durch diesen Test soll sichergestellt werden, dass die entwickelte Bibliothek auch innerhalb einer kleinen Anwendung funktioniert.

### 8.2.2.1 Voraussetzungen

Grundsätzlich ist diese Integraton bei allen Applikationen in der folgenden Art und Weise durchführbar. Für die Verwendung der entwickelten Jar Bibliothek für Abfragen von „xsd:DateTime“ Objekten muss mindestens Java in der Version 1.5 verwendet werden. Es muss auch mindestens die Version 2.5.4 des Jena Frameworks verwendet werden. Alle Tests sind mit den hier genannten Versionen durchgeführt worden. Damit diese Bibliothek lauffähig ist, sind die Bibliotheken des Jena Frameworks notwendig. In Abbildung 8.1 sind die benötigten Bibliotheken abgebildet.



**Abbildung 8.1:** Benötigte Bibliotheken für die Abfragen

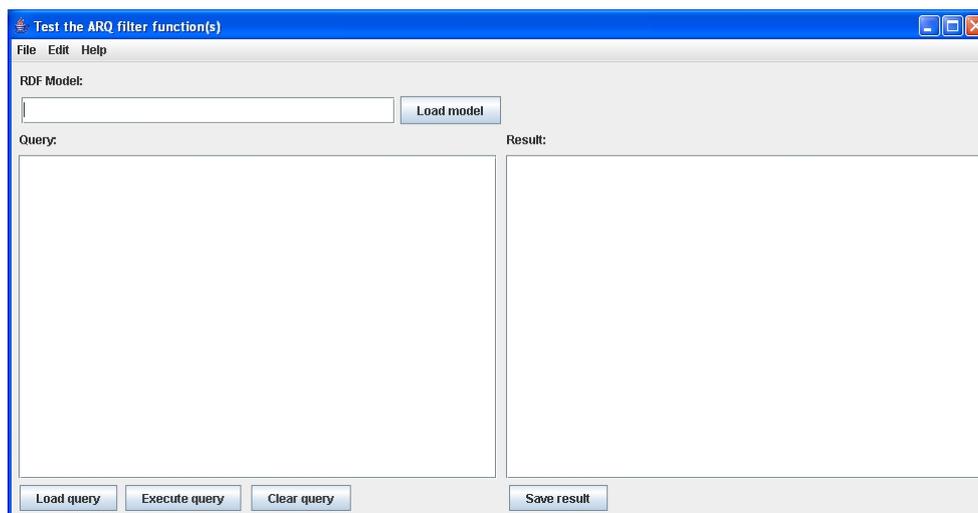
Um die erzeugte Jar Bibliothek in eine Anwendung einbinden zu können, müssen einige Vorbedingungen erfüllt sein. Die Jar Datei für die Abfragen von „xsd:DateTime“ Objekten kann wie eine Java Bibliothek verwendet werden. Dazu ist es notwendig, dass diese Bibliothek zur gewünschten Applikation hinzugefügt wird. Wenn bei der Applikation ein Ordner für Bibliotheken existiert, dann muss diese Jar Datei dort eingefügt werden. Üblicherweise existiert ein Ordner „lib“ für verwendete Bibliotheken. Anschließend muss die neu adaptierte Bibliothek zum „Classpath“ hinzugefügt werden.

Anschließend können die Filter Funktionen der Jar Bibliothek für die Abfragen von „xsd:DateTime“ Objekten in einer Applikation verwendet werden. Hierfür müssen die Filter Funktionen so angewendet werden, wie sie im Abschnitt 8.1.1 beschrieben sind.

#### 8.2.2.2 Testtool

Die implementierten Filter Funktionen werden durch ein graphisches Testtool getestet, welches in Abbildung 8.2 dargestellt ist. Über dieses Testtool werden alle Testfälle, welche auch mit JUnit durchgeführt wurden, wiederholt. Dazu werden die Abfragen in einer Datei abgespeichert um diese dann im Tool zu öffnen.

Bei diesem Testtool kann ein gespeichertes RDF Model über den Button „Load model“ geladen werden. Der Pfad zu diesem Model wird anschließend im zugehörigen Textfeld



**Abbildung 8.2:** Start Ansicht des Testtools.

angezeigt. Durch die Verwendung des Buttons „Load query“ kann eine gespeicherte Abfrage geladen werden. Diese wird anschließend im Textfeld angezeigt. Es besteht aber auch die Möglichkeit direkt eine Abfrage in das Textfeld zu schreiben. Für den Testverlauf wurde für jede Filter Funktion mindestens ein Testfall erstellt. Dieser Testfall besteht aus einer Abfrage, welche in einer Datei abgespeichert wird. Dadurch wird der Testverlauf erleichtert, da eine Abfrage nicht immer neu eingegeben werden muss. Eine Abfrage kann dann durch die Verwendung des Buttons „Execute query“ ausgeführt werden. Das Ergebnis dieser Abfrage wird im Textfeld angezeigt. Wenn eine Abfrage kein Ergebnis liefert, dann wird als Information „There is no result“ im Textfeld ausgegeben. Dieses Ergebnis kann durch die Verwendung des Button „Save result“ abgespeichert werden. Wenn der Button „Clear query“ verwendet wird, dann werden alle Felder mit ihren initialen Werten geladen. Abbildung 8.3 zeigt diese Applikation nach dem eine Abfrage auf ein Model ausgeführt wurde. Dabei wird auch das Resultat der Abfrage angezeigt.

In Abbildung 8.4 sind die Einträge des „File“ Menüs dargestellt. Über dieses Menü kann eine Abfrage oder ein RDF Model von einer Datei geladen werden. Eine Abfrage oder ein Ergebnis einer Abfrage kann über dieses Menü in einer Datei abgespeichert werden. Beim Start des Testtools kann ein RDF Model aus einer Datei geladen werden, dabei wird aber nur der Pfad zu diesem Model angezeigt. Durch den Menüeintrag „Show model“ kann dieses Model angezeigt werden. Dieses Model wird in einem separaten Fenster dargestellt, welches in Abbildung 8.5 dargestellt ist. Über den „Close“ Button kann das Fenster für die Ansicht des Models geschlossen werden. Über den letzten Menüeintrag „Exit“ kann das Testtool geschlossen werden.

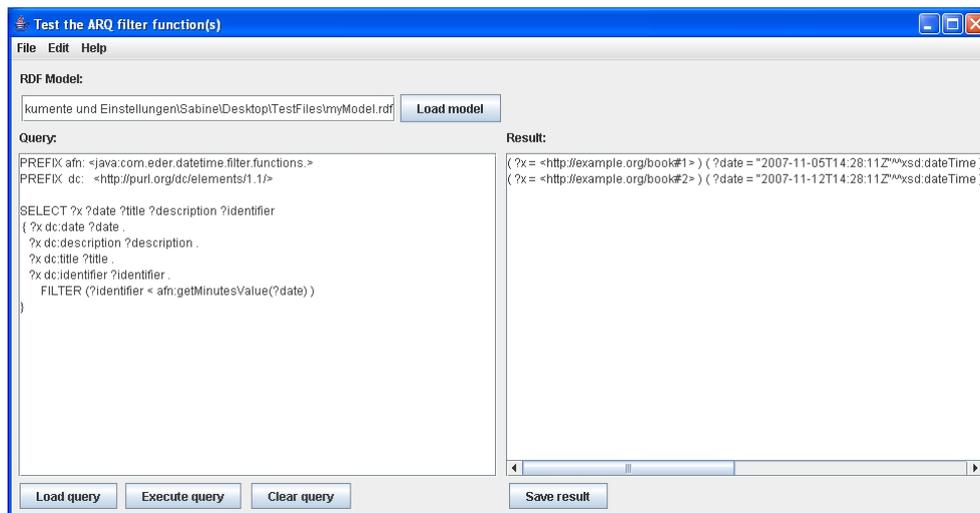


Abbildung 8.3: Ansicht nach dem Ausführen einer Anfrage

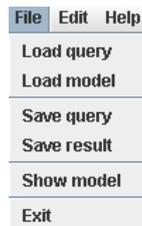


Abbildung 8.4: Einträge des Menü „File“

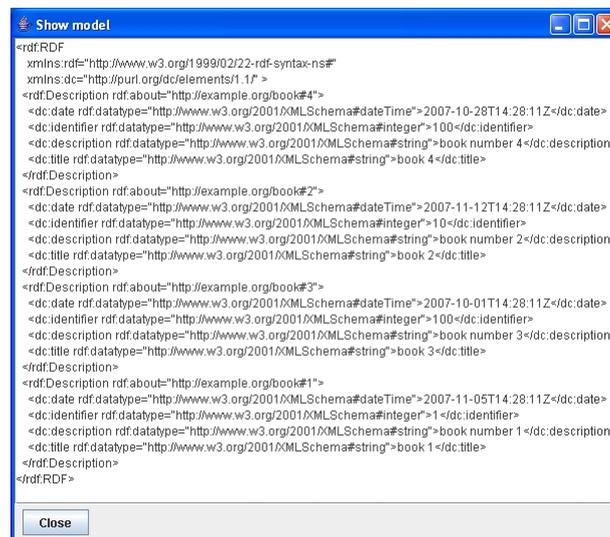
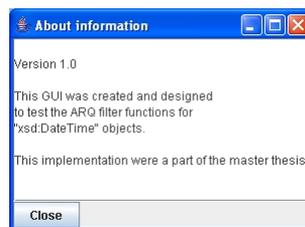
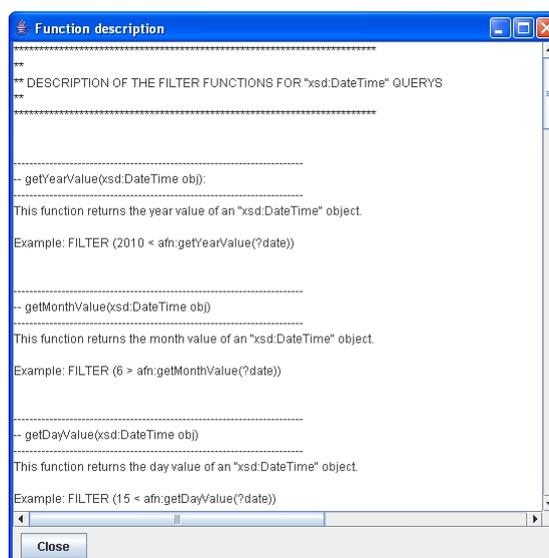


Abbildung 8.5: Ansicht zum Anzeigen eines Modells

Durch das Menü „Edit“ stehen die Befehle „Cut“, „Copy“ und „Paste“ zur Verfügung. Diese Befehle können für das Schreiben der Abfrage genutzt werden. Das Menü „Help“ besteht aus den zwei Einträgen „About“ und „Functions description“. Über den Eintrag „About“ werden verschiedene Informationen zu diesem Testtool angezeigt. Dafür öffnet sich ein separates Fenster, welches in Abbildung 8.6 abgebildet ist. Durch den Menüeintrag „Functions description“ wird eine Beschreibung der verfügbaren Filter Funktionen für „xsd:DateTime“ Objekte angezeigt. Hierzu wird ebenfalls ein separates Fenster geöffnet, welches in Abbildung 8.7 abgebildet ist.



**Abbildung 8.6:** Ansicht zum Anzeigen von Informationen



**Abbildung 8.7:** Ansicht zum Anzeigen der Beschreibung der Filter Funktionen.

Für das Speichern in eine Datei oder das Laden aus einer Datei wird ein Dateibrowser verwendet. Dieser ist in Abbildung 8.8 dargestellt.

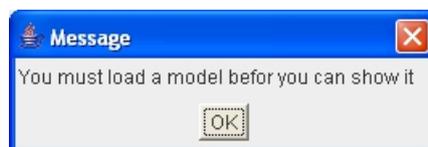
Während der Benutzung des Testtools werden Dialoge verwendet, um dem bzw. der BenutzerIn verschiedene Informationen mitzuteilen. Ein Beispiel für einen solchen Dialog



**Abbildung 8.8:** Dateibrowser zum Laden oder Speichern einer Datei

ist in Abbildung 8.9 dargestellt. Diese Dialoge werden in den folgenden Situationen verwendet:

- Das Textfeld für die Anzeige des Resultats einer Abfrage ist leer. Der bzw. die BenutzerIn versucht das Ergebnis in eine Datei zu speichern. Dann wird dem bzw. der BenutzerIn über ein Dialogfenster mitgeteilt, dass noch nichts zum Speichern vorhanden ist.
- Das Textfeld für die Eingabe einer Abfrage ist leer. Der bzw. die BenutzerIn versucht aber eine Abfrage in eine Datei zu speichern. Dann wird dem bzw. der BenutzerIn über ein Dialogfenster mitgeteilt, dass noch nichts zum Speichern vorhanden ist.
- Es ist kein RDF Model geladen worden. Der bzw. die BenutzerIn möchte aber das RDF Model anzeigen. Dann wird dem bzw. der BenutzerIn über ein Dialogfenster mitgeteilt, dass zuvor ein RDF Model geladen werden muss.
- Der bzw. die BenutzerIn möchte eine Abfrage über den Button „Execute Query“ ausführen. Das Textfeld für die Abfrage ist aber leer. Dann wird dem bzw. der BenutzerIn über ein Dialogfenster mitgeteilt, dass zuvor eine Abfrage eingegeben werden muss.
- Der bzw. die BenutzerIn möchte eine Abfrage über den Button „Execute Query“ ausführen. Es wurde aber kein RDF Model geladen. Dann wird dem bzw. der BenutzerIn über ein Dialogfenster mitgeteilt, dass zuvor ein RDF Model geladen werden muss.



**Abbildung 8.9:** Beispiel eines Dialoges

Generell wurde versucht das Testtool modular aufzubauen, sodass einzelne Module auch für andere Applikationen verwendet werden können. Das Testtool gliedert sich in die Klassen `DateTimeGUI.java`, `Dialoge.java`, `FileBrowser.java` sowie `GUIConstants.java` und `LoadModelFromFile.java`.

Die Klasse `DateTimeGUI.java` enthält die Hauptfunktionen des Testtools. Hier werden die GUI<sup>4</sup> Elemente erzeugt und auch positioniert. In dieser Klasse sind auch die benötigten Action Listener implementiert. Ein Action Listener wird immer dann benötigt, wenn auf eine Benutzerinteraktion mit dem GUI eine Aktion folgen soll. Ein Beispiel hierfür ist, wenn der bzw. die BenutzerIn auf den Button „Execute Query“ klickt, dann sollen die Funktionen für das Ausführen einer Abfrage gestartet werden. Die Steuerung der einzelnen Views geschieht ebenfalls über diese Klasse. Des Weiteren werden noch verschiedene Initialisierungen durchgeführt, sowie das Setzen des Layouts.

Durch die Klasse `Dialoge.java` werden die Funktionen für die Erstellung von Dialog Boxen bereit gestellt. Es stehen Funktionen zur Verfügung, um einen „OK“ Dialog, einen „YES/NO“ Dialog oder einen „YES/NO/CANCEL“ Dialog zu generieren. Des Weiteren werden in dieser Klasse die benötigten Action Listener implementiert.

Um eine Datei im lokalen Dateisystem zu speichern, oder um eine Datei aus dem lokalen Dateisystem zu laden, wird die Klasse `FileBrowser.java` verwendet. Diese Klasse stellt die Funktionen für einen Dateibrowser zur Verfügung. Dazu gehört die graphische Erzeugung des Dateibrowsers sowie die dafür benötigten Funktionen. Als Resultat wird hier der ausgewählte Dateipfad zurück geliefert. Die eigentliche Speicherung der Datei erfolgt in der Hauptklasse `DateTimeGUI.java`.

Konstante Werte, welche mehrmals im Programmcode benötigt werden, wie z.B. der Text für die Dialoge, wird in der Klasse `GUIConstants.java` gespeichert. Bei einer Änderung muss diese nur zentral in einer Datei durchgeführt werden und nicht mehreren Dateien. Die Klasse `LoadModelFromFile.java` liest ein RDF Model aus einer Datei ein.

### 8.3 Einsatz

Nach der Evaluierungsphase wird die entwickelte Bibliothek in einer Applikation angewendet. Durch das Einbinden der Bibliothek in eine andere Java Applikation ist die Verwendbarkeit gut ersichtlich.

---

4. Graphical User Interface

### 8.3.1 PIB - Personal Information Box

Die entwickelte Bibliothek wird in der Applikation „PIB - Personal Information Box“<sup>5</sup> verwendet. PIB wurde im Zuge der Diplomarbeit von Feurle (2007) entwickelt. Das Hauptziel dieser Applikation ist es, dass persönliche Daten mit Hilfe von Metadaten effizient wieder gefunden werden. Hierbei können verschiedene Arten von Dateien von der Applikation verwaltet werden. Einige dieser Dateitypen sind HTML Dokumente, E-Mail Nachrichten, Microsoft Word und Excel Dokumente sowie OpenOffice Dokumente und PDF Dokumente. Bei dieser Suche werden auch die Metadaten der Dateien durchsucht. Dadurch hat der bzw. die BenutzerIn den Vorteil, dass mehr Informationen durchsucht werden können. Nachdem die verschiedenen Dateien durchsucht wurden, werden die extrahierten Metadaten sowie der zugehörige Text in einem RDF Triple Store gespeichert. Um die Informationen aus den unterschiedlichen Dateien zu extrahieren, verwendet diese Applikation das Semantic Web Framework „Aperture“<sup>6</sup>. Um die Informationen aus dem Volltext Inhalt und den Metadaten zu extrahieren, wird vom Framework eine Ontologie verwendet. (Feurle, 2007)

In der Regel ist es der Fall, dass ein bzw. eine BenutzerIn eine große Datenmenge zu verwalten hat. Daher stellt die Applikation PIB unterschiedliche Suchanfragen zur Verfügung. Eine davon ist die Volltextsuche. Bei dieser Suchanfrage werden die Dateien sowie die Metadaten der Dateien nach dem gesuchten Text durchsucht. Des Weiteren besteht die Möglichkeit das nur bestimmte Dateitypen angezeigt werden. Eine weitere Möglichkeit ist, dass die Suche auf einen Datums- oder einen Zeitwert eingeschränkt wird. Datums- und Zeitwerte werden mit dem Datentyp „xsd:DateTime“ dargestellt. Für die Einschränkungen dieser Suche wird die entwickelte Bibliothek für Datums- und Zeitabfragen verwendet. Die Einbindung der Bibliothek für die Datums- und Zeitabfragen erfolgte nach der Beschreibung aus Abschnitt 8.2.2.1. Dabei sind auch keine weiteren Probleme aufgetreten. (Feurle, 2007)

Abbildung 8.10 zeigt die graphische Benutzeroberfläche von PIB. Auf dieser Abbildung sind die Masken für die Sucheinschränkung nach einem Datums- oder Zeitwerte dargestellt. (Feurle, 2007)

In Abbildung 8.11 wird die Eingabe der Datumswerte dargestellt. Die Auswahl des Datums erfolgt über ein Kalenderobjekt. Dadurch ist auch sichergestellt, dass keine ungültigen Eingabewerte möglich sind. Wenn der bzw. die BenutzerIn ein Datum ausgewählt hat, so wird dieses in jeweiligem Datumsfeld angezeigt. Für die Suchabfrage ob Dateien innerhalb eines Datumsbereichs sind, wurden die Filter Funktionen „*rangeDay(xsd:DateTime obj, String weekday, String weekday)*“ und „*rangeMonth(xsd:DateTime obj, String month, String month)*“ sowie „*rangeYear(xsd:DateTime obj, Integer year,*

5. Im Folgenden weiter PIB genannt

6. <http://aperture.sourceforge.net/> - Zugriffsdatum: 17.12.2007

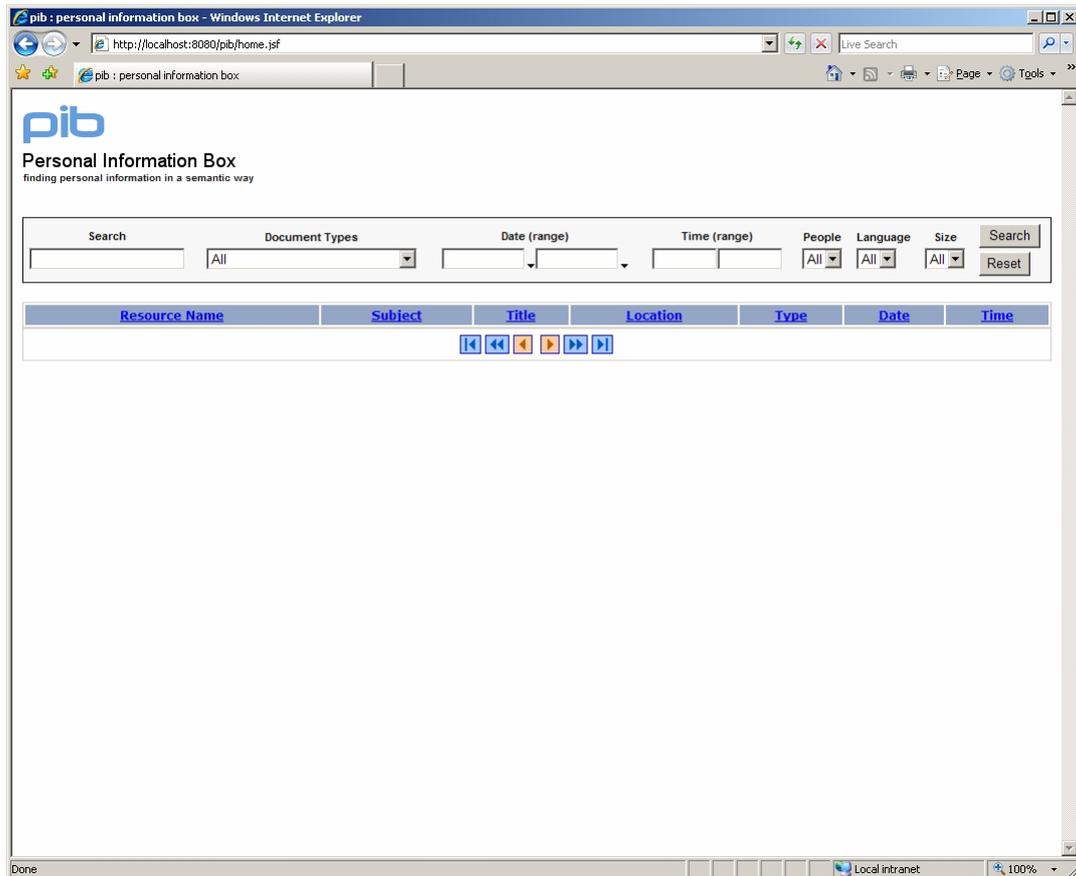
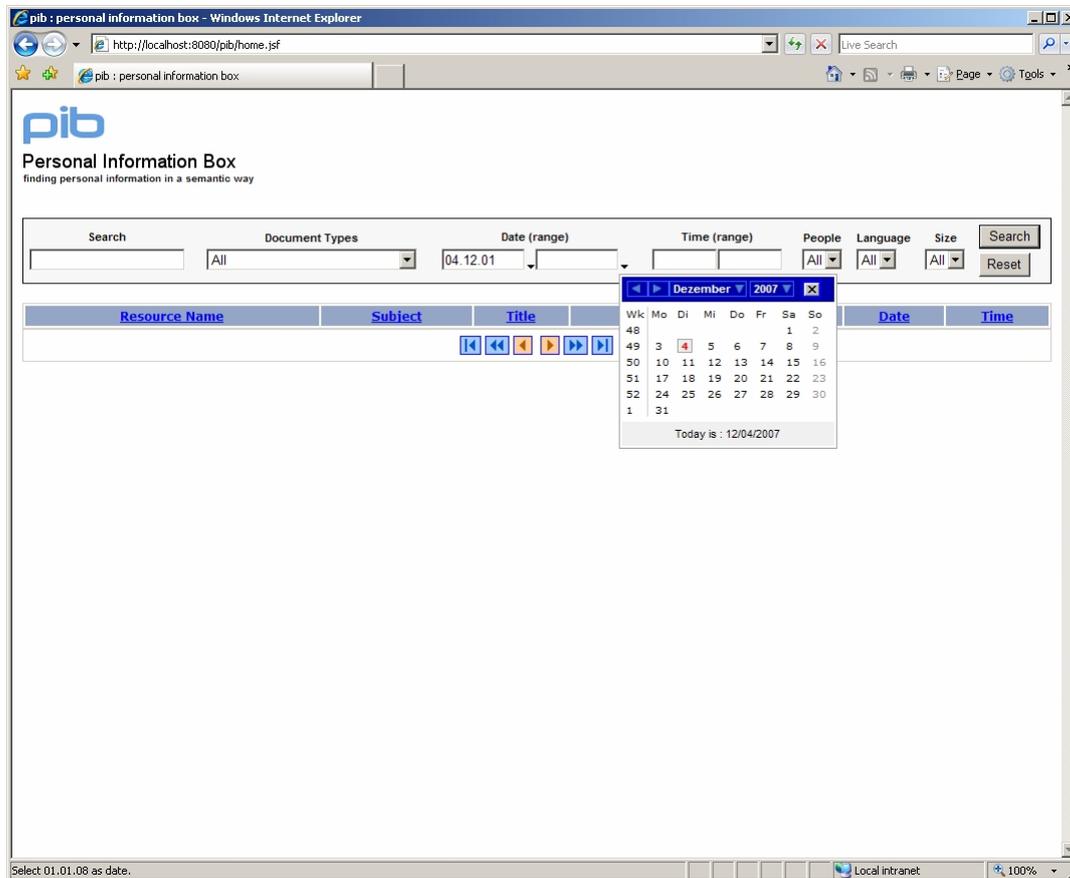


Abbildung 8.10: PIB - Nach dem Start (Quelle: Feurle, 2007)

*Integer year*)“ verwendet. Innerhalb der Query für die Suchabfrage werden diese Filter Funktionen durch ein logischen AND mit einander verknüpft. (Feurle, 2007)



**Abbildung 8.11:** PIB - Eingabe für die Suche nach einem Datumswert (Quelle: Feurle, 2007)

Die Suchabfrage wird über den Button „Search“ gestartet, nachdem die Werte für den Datumsbereich ausgewählt wurden. Abbildung 8.12 zeigt das Ergebnis einer Suchabfrage. Dabei wurde nach allen Dokumente gesucht, bei welchen das Datum zwischen dem 04.12.2001 und dem 04.12.2007 liegt.

Es kann auch nach Dateien gesucht werden, welche innerhalb einer bestimmten Zeitspanne liegen. Für diese Suchabfrage werden die Eingabefelder „Time (range)“ verwendet. Diese Suchabfrage ist ähnlich zu der Suche innerhalb eines Datumsbereich. Für diese Einschränkung werden die Filter Funktionen „*rangeHours(xsd:DateTime obj, String hour, String hour, String format)*“ und „*rangeMinutes(xsd:DateTime obj, Integer minute, Integer minute)*“ sowie „*rangeSeconds(xsd:DateTime obj, Integer second, Integer*

**Personal Information Box**  
finding personal information in a semantic way

Search:  Document Types: All Date (range): 04.12.01 - 04.12.07 Time (range):  -  People: All Language: All Size: All Search  Reset

Resource Name	Subject	Title	Location	Type	Date	Time
file:/C:/testdocs/DA.pdf	email - versuche des verstehens	Diplomarbeit	file:/C:/testdocs/	PDF Document	Do, 18. Okt 2007	05:32
file:/C:/testdocs/Test-de.doc			file:/C:/testdocs/	Microsoft Word Document	Mi, 26. Sep 2007	14:45
file:/C:/testdocs/aperture-architecture.ppt		PowerPoint Presentation	file:/C:/testdocs/	Microsoft Powerpoint Document	Mo, 1. Jan 1601	00:00
file:/C:/testdocs/component-architecture.odp			file:/C:/testdocs/	OpenDocument Presentation	Fr, 24. Feb 2006	11:55
file:/C:/testdocs/eBooks/business/IT-KV2007.pdf		Kollektivvertrag	file:/C:/testdocs/eBooks/business/	PDF Document	Di, 14. Nov 2006	09:26
file:/C:/testdocs/eBooks/manuals/JabRef-UserManual_de.pdf	Einführung in die Benutzung von JabRef	JabRef 2.3 (beta) Benutzerhandbuch	file:/C:/testdocs/eBooks/manuals/	PDF Document	Mi, 4. Jul 2007	09:00
file:/C:/testdocs/eBooks/manuals/LBuch.pdf	Latex	Praktisches Latex - eine Einfuehrung	file:/C:/testdocs/eBooks/manuals/	PDF Document	Di, 10. Jul 2001	07:29
file:/C:/testdocs/eBooks/manuals/latexkurzreferenz.pdf		LaTeX-Kurzreferenz.doc	file:/C:/testdocs/eBooks/manuals/	PDF Document	Fr, 12. Mai 2006	19:06
file:/C:/testdocs/eBooks/manuals/lshort.pdf			file:/C:/testdocs/eBooks/manuals/	PDF Document	So, 4. Apr 2004	21:27
file:/C:/testdocs/eBooks/manuals/scrguide.pdf	Die Anleitung	KOMA-Script	file:/C:/testdocs/eBooks/manuals/	PDF Document	Mo, 1. Okt 2007	08:05

Navigation: << < > >> 1 2 >>>

Abbildung 8.12: PIB - Resultat einer Suche nach einem Datumswert (Quelle: Feurle, 2007)

*second*)“ verwendet. Auch diese Filter Funktionen werden innerhalb der Query durch ein logischen AND miteinander verknüpft. (Feurle, 2007)

## Kapitel 9

### Abschluss

In diesem Kapitel wird die Arbeit noch einmal betrachtet. Dazu gehört auch eine abschließende Zusammenfassung der Arbeit. Im zweiten Abschnitt werden die Schlußfolgerungen beschrieben, welche aus dieser Arbeit gezogen werden. Des Weiteren wird ein Ausblick auf mögliche Erweiterungen gegeben.

#### 9.1 Zusammenfassung

Das Semantic Web ist ein aktuelles und aktives Forschungsgebiet. Daher werden viele verschiedene Applikationen in diesem Bereich entwickelt. Im Unterschied zu anderen Bereichen findet man hier keine „große“ Applikation, durch welche alle Anforderungen abgedeckt sind. Ganz im Gegenteil, man findet in diesem Forschungsbereich viele „kleine“ Applikationen für die verschiedenen Anforderungen. Da dieser Forschungsbereich eine sehr große Interessensgemeinschaft besitzt, werden laufend unterschiedliche Ansätze erforscht. Das „SemanticLIFE Project“ ist ein aktuelles Forschungsprojekt auf diesem Gebiet. Im wirtschaftlichen Bereich bekommt das Semantic Web auch eine immer größere Bedeutung. Dies wird dadurch deutlich, dass „große“ Software Firmen wie z.B. Oracle<sup>1</sup> Semantic Web Technologien einsetzen.

Eine Zielsetzung des Semantic Web ist es, dass Informationen auch von Maschinen gelesen und verarbeitet werden können. Für diesen Zweck wird unter anderem RDF verwendet. RDF ist XML von der Syntax her sehr ähnlich, aber mit dem Unterschied, dass RDF auf einem höheren Abstraktionsniveau arbeitet. Aus diesem Grund werden Query Sprachen benötigt. Über diese Query Sprachen ist es möglich, dass Informationen anhand bestimmter Kriterien gefunden werden können. SPARQL und RDQL sind nur zwei Vertreter solcher Query Sprachen für RDF.

---

1. Software Company - siehe dazu auch <http://www.oracle.com/> - Zugriffsdatum: 16.12.2007

In RDF werden Datums- und Zeitwerte durch den XSD Datentyp „xsd:DateTime“ dargestellt. Dieser Datentyp ist exakt definiert und läßt daher auch keine Spielraum zu. Dies wiederum kann zu Einschränkungen bei der Datenabfrage führen. Wenn dieser Datentyp in Verwendung ist, dann ist es nicht möglich, dass z.B. nur der Zeitwert abgefragt wird. Auf diese Problemstellung wird vertiefend in dieser Arbeit eingegangen. Mit den implementierten Erweiterungen ist eine solche ungenaue Abfrage auf den „xsd:DateTime“ Datentyp möglich. Die Erweiterungen für unpräzise Datums- und Zeitabfragen sind als Java Bibliothek für das Semantic Web Framework „Jena“ implementiert. Das Framework Jena unterstützt auch SPARQL Queries. Für die konkrete Realisierung der ungenauen Suchabfragen wurden Filter Funktionen verwendet. Die Filter Funktionen sind so entworfen, dass sehr viele möglichen Abfragen für den „xsd:DateTime“ Datentyp möglich sind.

Die Funktionen sind so entworfen, dass die Funktionalität auf mehrere Funktionen aufgeteilt ist. Diese Filter Funktionen können innerhalb einer SPARQL Query verwendet werden. Hierzu muss die Programmiersprache Java verwendet werden und ein Semantic Web Framework, welches SPARQL Queries unterstützt. Alle entwickelten Filter Funktionen können innerhalb einer Query miteinander kombiniert werden. Die Queries werden dabei durch ein logisches OR oder durch ein logisches AND mit einander verknüpft. Durch die Kombinationsmöglichkeiten wird ein weites Spektrum für variable Datums- und Zeitabfragen abgedeckt.

Bei der Entwicklung der flexiblen Datums- und Zeitabfragen war die Portabilität ein wichtiger Aspekt. Die Erweiterungen können ohne einen großen Aufwand in bestehende Applikationen eingebunden werden. Nach der Entwicklung wurden unterschiedliche Tests durchgeführt, bei welchen auch die Portabilität getestet wurde. Die Java Bibliothek für die unpräzisen Datums- und Zeitabfragen ist in der Applikation „Personal Information Box“ eingebunden und verwendet worden.

## 9.2 Schlussfolgerung

Das Semantic Web ist ein sehr dynamisches und auch aktives Forschungsgebiet. Dies erkennt man unter anderem daran, dass auf diesem Gebiet eine große Anzahl von wissenschaftlichen Arbeiten und Software Applikationen vorhanden sind. Ein aktuelles Forschungsprojekt auf diesem Gebiet ist das „SemanticLIFE Project“. Das Semantic Web ist eine Erweiterung des World Wide Web (WWW). Der Unterschied besteht darin, dass im World Wide Web (WWW) Informationen nur dargestellt werden. Eine Zielsetzung des Semantic Web ist, dass Informationen auch von Maschinen gelesen und verarbeitet werden können. Hierfür wird unter anderem RDF verwendet. Durch RDF werden Informationen auf eine Art und Weise dargestellt, dass sie auch von Maschinen interpretiert

werden können. Um in RDF Datums- und Zeitwerte darzustellen wird der Datentyp „xsd:DateTime“ verwendet.

Dieser Datentyp ist auf den ersten Blick sehr komfortabel, da Datums- und Zeitwerte gemeinsam abgebildet werden. Wenn es aber darum geht, Informationen von diesem Datentyp abzufragen, stößt man schnell auf die Schwächen des Datentyps. Es müssen immer alle Werte des Datums- und Zeitwerts bekannt sein. Da es oft vorkommt, dass nicht alle diese Werte bekannt sind, kann sich dies als ein Problem bei Queries erweisen. Um diesen Datentyp abzufragen wird eine Query Sprache benötigt. Eine solche Query Sprache ist SPARQL. Der Schwerpunkt dieser Arbeit ist eine flexible Suchabfrage für Datums- und Zeitwerte. Hierfür ist das Framework Jena verwendet worden. Jena stellt eine Vielzahl von unterschiedlichen Funktionen zur Verfügung und wird auch von vielen Applikationen verwendet. Eine Engine für SPARQL wird ebenfalls unterstützt. Die variable Suchabfrage für Datums- und Zeitwerte wird durch Filter Funktionen realisiert.

Eine solche Filter Funktion wird innerhalb einer SPARQL Query angewendet. Durch eine Kombination der unterschiedlichen Filter Funktionen ist ein weites Spektrum von möglichen variablen Suchabfragen abgedeckt. Mögliche Suchabfragen wären z.B. folgende:

- Ausgabe von allen Objekten, bei welchen das Jahr zwischen 1990 und 2000 liegt.
- Es sollen alle Objekte ausgegeben werden, welche jeweils einen Montag als Wochentag haben.
- Anzeigen von jenen Objekten, die innerhalb der Zeispanne von 15:00 Uhr bis 20:00 Uhr liegen.

Die Funktionalität dieser Filter Funktionen wird als Java Bibliothek zur Verfügung gestellt. Dadurch ist es möglich, dass diese Funktionalität mit geringem Aufwand in bestehende Projekte eingebunden werden kann.

### 9.3 Ausblick

Es existieren noch weitere Datentypen, bei welchen auf Grund ihrer Definition keine ungenauen Suchabfragen möglich sind. Hier sind Erweiterungen für ungenaue Suchabfragen von Vorteil.

Zwei solcher Datentypen sind „xsd:Time“ und „xsd:Date“. Diese Datentypen sind dem „xsd:DateTime“ Datentyp sehr ähnlich. Mit dem Datentyp „xsd:Time“ wird eine Zeitangabe dargestellt. Es müssen die Werte für Stunden, Minuten und Sekunden immer angegeben werden. Die Zeitzone kann optional angegeben werden, dabei wird diese im UTC Format dargestellt. Datumsangaben ohne einen Zeitwert können durch den Datentyp „xsd:Time“ dargestellt werden. Benötigt werden die Werte für den Tag und das

Monat sowie das Jahr. Auch hier kann der Fall vorkommen, dass nicht alle benötigten Werte bekannt sind. Es stellt sich hier eine ähnliche Problematik wie bei der Suchabfrage vom „xsd:DateTime“ Datentyp. Eine unpräzise Suchabfrage würde bei diesen Datentypen ebenfalls sinnvoll sein. (W3Schools, 2007)

## Anhang A

### RDF - Klassen & Eigenschaften

Einen Überblick über die Klassen und Eigenschaften von RDF geben die Tabellen A.2 und A.1. In beiden Tabellen existieren Klassen und Eigenschaften, welche an dieser Stelle das erste Mal erwähnt werden. Dies soll zu einem besseren Überblick beitragen. Wobei die Tabelle A.2 so zu lesen ist:

„Die Eigenschaft `rdf:type` erlaubt als Domäne die Klassen `rdfs:Resource` und als Wertebereich die Klassen `rdfs:Class`.“

Name	Beschreibung
<code>rdfs:Resource</code>	Klasse für Ressourcen, alle anderen Klassen sind Subklassen.
<code>rdfs:Literal</code>	Klasse für Literale Werte, z.B. textuelle Strings und Zahlen.
<code>rdf:XMLLiteral</code>	Klasse für XML Werte.
<code>rdfs:Class</code>	Klasse für Klassen.
<code>rdf:Property</code>	Klasse für RDF Eigenschaften.
<code>rdfs:Datatype</code>	Klasse für RDF Datentypen.
<code>rdf:Statement</code>	Klasse für RDF Aussagen.
<code>rdf:Bag</code>	Klasse für ungeordnete Containers.
<code>rdf:Seq</code>	Klasse für geordnete Containers.
<code>rdf:Alt</code>	Klasse für Containers von Alternativen.
<code>rdfs:Container</code>	Klasse für RDF Containers.

**Tabelle A.1:** RDF Klassen - Überblick (Quelle: W3C u. a., 2004).

– Fortsetzung auf der nächsten Seite –

*Fortsetzung:*

Name	Beschreibung
rdfs:ContainerMembershipProperty	Klasse für Eigenschaften von Containermitgliedern, rdf:_1, rdf:_2, usw.
rdf:List	Klasse für RDF Listen

**Tabelle A.1:** RDF Klassen - Überblick (Quelle: W3C u. a., 2004)

Name	Beschreibung	domain	range
rdf:type	Das Subjekt ist eine Instanz einer Klasse	rdfs:Resource	rdfs:Class
rdfs:subClassOf	Das Subjekt ist eine Subklasse einer Klasse	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	Das Subjekt ist eine Subeigenschaft einer Klasse	rdf:Property	rdf:Property
rdfs:domain	Eine Domäne der Eigenschaft Subjekt	rdf:Property	rdfs:Class
rdfs:range	Eine Abgrenzung der Eigenschaft Subjekt	rdf:Property	rdfs:Class
rdfs:label	Ein menschenlesbarer Name für ein Subjekt	rdfs:Resource	rdfs:Literal
rdfs:comment	Eine Beschreibung für die Subjekt Ressource	rdfs:Resource	rdfs:Literal

**Tabelle A.2:** RDF Eigenschaften - Überblick (Quelle: W3C u. a., 2004)

– Fortsetzung auf der nächsten Seite –

*Fortsetzung:*

<b>Name</b>	<b>Beschreibung</b>	<b>domain</b>	<b>range</b>
rdfs:member	Ein Mitglied der Subjekt Ressource	rdfs:Resource	rdfs:Resource
rdf:first	Das erste Element in der Subjekt RDF Liste	rdf:List	rdfs:Resource
rdf:rest	Der Rest der Subjekt RDF Liste nach dem erstem Element	rdf:List	rdf:List
rdfs:seeAlso	Weitere Informationen über die Subjekt Ressource	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	Die Definition der Subjekt Ressource	rdfs:Resource	rdfs:Resource
rdf:value	Idiomische Eigenschaft, wird für strukturierte Werte verwendet	rdfs:Resource	rdfs:Resource
rdf:subject	Das Subjekt einer RDF Aussage	rdf:Statement	rdfs:Resource
rdf:predicate	Das Prädikat einer RDF Aussage	rdf:Statement	rdfs:Resource
rdf:object	Das Objekt einer RDF Aussage	rdf:Statement	rdfs:Resource

**Tabelle A.2:** RDF Eigenschaften - Überblick (Quelle: W3C u. a., 2004)

## Anhang B

### Dublin Core - Metadata Element Set

Die Tabelle B.1 soll einen Überblick über das „Metadaten Element Set“ des Dublin Core geben.

<b>Element</b>	<b>Beschreibung</b>
Title	Gibt den Namen der Ressource an.
Creator	Gibt den Hauptverantwortlichen für eine Ressource
Subject	Gibt an zu welchem Thema der Ressourceninhalt gehört.
Description	Gibt einen verbalen Überblick über die Ressource an.
Publisher	Gibt an wer für die Veröffentlichung der Ressource verantwortlich ist.
Contributor	Gibt an wer noch einen Beitrag zur Ressource leistet.
Date	Gibt den Zeitpunkt eines Ereignisses einer Ressource an.
Type	Gibt das Genre des Inhaltes an.
Format	Gibt die Beschreibung der physikalischen oder digitalen Form der Ressource an.
Identifizier	Gibt eine eindeutige Identifizierung innerhalb eines Kontextes an.

**Tabelle B.1:** DCMI Metadata Element Set - Überblick (Quelle: Eckstein und Eckstein, 2003; DCMI, 2007)

– Fortsetzung auf der nächsten Seite –

*Fortsetzung:*

<b>Element</b>	<b>Beschreibung</b>
Source	Gibt die Quelle an, von welcher eine Ressource abgeleitet ist.
Language	Gibt die Sprache an in welcher eine Ressource verfasst ist.
Relation	Gibt Verwandte Ressourcen an.
Coverage	Gibt die Ausdehnung uder den Inhaltsbereich einer Ressource an.
Rights	Gibt das Copyright, Eigentumsrechte oder ähnliches zu einer Ressource an.

**Table B.1:** DCMI Metadata Element Set - Überblick (Quelle: Eckstein und Eckstein, 2003; DCMI, 2007)

## **Danksagung**

An dieser Stelle möchte ich mich bei meinem Umfeld für die Unterstützung bedanken. Zuerst möchte ich meiner Betreuerin **Monika Lanzenberger** dafür danken, dass sie mir immer sehr viel Freiraum gelassen hat, und dass sie immer mit Rat und Tat zur Seite stand. Meiner Familie möchte ich an dieser Stelle auch danken für die Unterstützung in den letzten Jahren. Bei **Thorsten Guggenberger** möchte ich mich für die konstruktive Zusammenarbeit und für seine Unterstützung während dieser Arbeit und des gesamten Studiums bedanken. Konstruktiv und hilfreich war auch die Zusammenarbeit mit **Daniel Feurle**. Ein Dank gilt auch **Gerd Reichmann** für das laufende Korrektur lesen dieser Arbeit. Für die Latex Vorlage zur Diplomarbeit möchte ich mich bei **Markus Rester** bedanken.

## Abbildungsverzeichnis

3.1	Architektur des Semantic Web . . . . .	15
3.2	Weiterentwickelte Architektur des Semantic Web . . . . .	21
4.1	Einfache RDF Aussage . . . . .	27
4.2	RDF Aussage mit strukturierten Werten zur Beschreibung . . . . .	28
4.3	RDF Schema - Eigenschaften und Klassen . . . . .	36
5.1	Aufbau des Inferenzsystems . . . . .	47
5.2	Einfacher RDF Graph . . . . .	48
5.3	Zwei Modelle bevor die Operation „Union“ ausgeführt wird . . . . .	52
5.4	Modell nach dem die Operation „Union“ ausgeführt wurde . . . . .	53
6.1	Architektur des „SemanticLIFE“ Systems . . . . .	59
6.2	Query Prozess im „SemanticLIFE“ System . . . . .	63
6.3	Virtuelles Query System (VQS) im „SemanticLIFE“ System . . . . .	65
6.4	Workflow des Virtuellen Query System (VQS) im „SemanticLIFE“ System . . . . .	68
7.1	Struktur der Packages . . . . .	82
8.1	Benötigte Bibliotheken . . . . .	100
8.2	Start Ansicht des Testtools . . . . .	101
8.3	Ansicht nach dem Ausführen einer Anfrage . . . . .	102
8.4	Einträge des Menü „File“ . . . . .	102
8.5	Ansicht zum Anzeigen eines Models . . . . .	102
8.6	Ansicht zum Anzeigen von Informationen . . . . .	103
8.7	Ansicht zum Anzeigen der Beschreibung der Filter Funktionen . . . . .	103
8.8	Dateibrowser zum Laden oder Speichern einer Datei . . . . .	104
8.9	Beispiel eines Dialoges . . . . .	104
8.10	PIB - Nach dem Start . . . . .	107
8.11	PIB - Eingabe für die Suche nach einem Datumswert . . . . .	108
8.12	PIB - Resultat einer Suche nach einem Datumswert . . . . .	109

## Tabellenverzeichnis

A.1	RDF Klassen - Überblick . . . . .	115
A.1	RDF Klassen - Überblick . . . . .	116
A.2	RDF Eigenschaften - Überblick . . . . .	116
A.2	RDF Eigenschaften - Überblick . . . . .	117
B.1	DCMI Metadata Element Set - Überblick . . . . .	118
B.1	DCMI Metadata Element Set - Überblick . . . . .	119

## Listings

4.1	RDF Basis Syntax (Quelle: Harmelen und Antoniou, 2004, S. 70) . . .	29
4.2	Verwendung der Attribute „rdf:about“ und „rdf:ID“ (Quelle: Harmelen und Antoniou, 2004, S. 73) . . . . .	29
4.3	Verwendung der Eigenschafts-Elemente (Quelle: Harmelen und Antoniou, 2004, S. 73) . . . . .	30
4.4	Verwendung des Type-Elementes (Quelle: Harmelen und Antoniou, 2004, S. 74) . . . . .	31
4.5	Verwendung von Container-Elementen (Quelle: Harmelen und Antoniou, 2004, S. 77) . . . . .	32
5.1	Erzeugung eines Graphen in Jena (Quelle: McBride, 2006) . . . . .	48
5.2	Auflisten von Aussagen (Quelle: McBride, 2006) . . . . .	49
5.3	Wert einer Eigenschaft abfragen (getObject) (Quelle: McBride, 2006) .	50
5.4	Wert einer Eigenschaft abfragen (getResource) (Quelle: McBride, 2006)	51
5.5	Auflisten der Eigenschaften (Quelle: McBride, 2006) . . . . .	51
6.1	Beispiel einer XML basierenden VQL Query vom Typ „data“ (Quelle: Hanh und Tjoa, 2006, S. 5) . . . . .	70
6.2	Beispiel einer XML basierenden VQL Query vom Typ „schema“ (Quelle: Hanh und Tjoa, 2006, S. 5) . . . . .	72
6.3	Beispiel einer XML basierenden VQL Query vom Typ „iTQL type 2“ (Quelle: Hanh und Tjoa, 2006, S. 7) . . . . .	73
7.1	XSD Datentypen „Date“, „Time“, „DateTime“ und die Verwendung der Zeitzone (Quelle: W3Schools, 2007) . . . . .	77
8.1	Beispiel für eine Query mit einer Filter Funktion . . . . .	86
8.2	Beispiel für eine Query mit zwei Filter Funktion . . . . .	86

## Literaturverzeichnis

- [Ahmed u. a. 2004] AHMED, Mansoor ; HOANG, Huu H. ; KARIM, Shuaib ; KHUSRO, Shah ; LANZENBERGER, Monika ; LATIF, Khalid ; MICHLMAYR, Elke ; MUSTOFA, Khabib ; THO, Manh ; RAUBER, Andreas ; SCHATTEN, Alexander ; TJOA, A M.: 'SemanticLIFE' - A Framework for Managing Information of a Human Lifetime. In: *Proceedings of the 6th International Conference on Information Integration and Web-based Applications and Services (iiWAS)* (2004), September. – URL <http://storm.ifs.tuwien.ac.at/publications/iiwas2004.pdf>. – Zugriffsdatum: 04.05.2007
- [Badertscher 2006] BADERTSCHER, Guido: *Generelle Queryverarbeitung für das Semantic Web - Effiziente Suche in strukturierten Daten*. Dudweiler Landstr. 125 a, 66123 Saarbrücken : VDM Verlag Dr. Müller e. K., September 2006. – ISBN 978-3-86550-904-8
- [Banks u. a. 2002] BANKS, Dave ; CAYZER, Steve ; DICKINSON, Ian ; REYNOLDS, Dave: *The ePerson Snippet Manager: a Semantic Web Application*. Internetseite. November 2002. – URL [http://www.hpl.hp.com/techreports/2002/HPL-2002-328.pdf?jumpid=reg\\_R1002\\_USEN](http://www.hpl.hp.com/techreports/2002/HPL-2002-328.pdf?jumpid=reg_R1002_USEN). – Zugriffsdatum: 26.09.2007
- [Beckett und Broekstra 2007] BECKETT, Dave ; BROEKSTRA, Jeen: *SPARQL Query Results XML Format*. Internetseite. September 2007. – URL <http://www.w3.org/TR/rdf-sparql-XMLres/>. – Zugriffsdatum: 05.11.2007
- [Berners-Lee 2000] BERNERS-LEE, Tim: *Semantic Web on XML*. Internetseite. 2000. – URL <http://www.w3.org/2000/Talks/1206-xml2k-tbl/>. – Zugriffsdatum: 14.03.2007
- [Berners-Lee 2005] BERNERS-LEE, Tim: *RFC3986*. Internetseite. Januar 2005. – URL <http://rfc.net/rfc3986.html>. – Zugriffsdatum: 15.12.2007
- [Berners-Lee 2006] BERNERS-LEE, Tim: *Notation 3*. Internetseite. März 2006. – URL <http://www.w3.org/DesignIssues/Notation3.html>. – Zugriffsdatum: 02.06.2007
- [Birkenbihl 2006] BIRKENBIHL, Klaus: Standards für das Semantic Web. In: (Pellegrini und Blumauer, 2006), S. 73–86. – ISBN 978-3-540-29324-8
- [Bratt 2006] BRATT, Steve: *Emerging Web Technologies to Watch*. Internetseite. 2006. – URL <http://www.w3.org/2006/Talks/>

- 1023-sb-W3CTechSemWeb/Overview.html#(19). – Zugriffsdatum: 15.12.2007
- [Brockhaus 1990] BROCKHAUS (Hrsg.): *DUDEN - Fremdwörterbuch*. 5. Auflage. Dudenstraße 6, 68167 Mannheim : Bibliographisches Institut & F. A. Brockhaus AG, 1990. – ISBN 978-3-411-20915-6
- [Brockhaus 2002] BROCKHAUS (Hrsg.): *Der Brockhaus Computer und Informationstechnologie. Hardware, Software, Multimedia, Internet, Telekommunikation*. 1. Auflage. Dudenstraße 6, 68167 Mannheim : Bibliographisches Institut & F. A. Brockhaus AG, August 2002 (Brockhaus Fachlexikon). – ISBN 978-3-765-30251-0
- [Bundesministerium für Finanzen 2007] BUNDESMINISTERIUM FÜR FINANZEN: *Lissabon-Strategie*. Internetseite. 2007. – URL [http://www.bmf.gv.at/Wirtschaftspolitik/Wirtschaftspolitik/510/EuropischeWirtschaft/730/LissabonStrategie/727/\\_start.htm](http://www.bmf.gv.at/Wirtschaftspolitik/Wirtschaftspolitik/510/EuropischeWirtschaft/730/LissabonStrategie/727/_start.htm). – Zugriffsdatum: 13.03.2007
- [Bundesministerium für Verkehr, Innovation und Technologie 2007] BUNDESMINISTERIUM FÜR VERKEHR, INNOVATION UND TECHNOLOGIE: *FIT-IT*. Internetseite. 2007. – URL <http://www.fit-it.at/>. – Zugriffsdatum: 13.03.2007
- [Bush 1945] BUSH, Vannevar: *As We May Think*. Internetseite. 1945. – URL <http://www.ps.uni-sb.de/~duchier/pub/vbush/vbush.shtml>. – Zugriffsdatum: 24.09.2007. – This article was originally published in the July 1945 issue of *The Atlantic Monthly*. It is reproduced here with their permission.
- [Cycorp, Inc. 2007] CYCORP, INC.: *Cycorp, Inc.* Internetseite. 2007. – URL <http://www.cyc.com/>. – Zugriffsdatum: 22.09.2007
- [DATACOM 2007] DATACOM: *Unicode*. Internetseite - DATACOM Buchverlag GmbH. 2007. – URL <http://www.itwissen.info/definition/lexikon/unicode/unicode.html>. – Zugriffsdatum: 14.03.2007
- [DCMI 2007] DCMI: *Dublin Core Metadata Initiative - Making it easier to find information*. Internetseite. 2007. – URL <http://dublincore.org/>. – Zugriffsdatum: 22.04.2007
- [Dittrich und Gatzju 2000] DITTRICH, Klaus R. ; GATZJU, Stella: *Aktive Datenbanksysteme - Konzepte und Mechanismen*. 2. Auflage. Ringstrasse 19 B, 69115 Heidelberg : Dpunkt.Verlag GmbH, 2000. – ISBN 3-932588-19-3
- [Eckstein und Eckstein 2003] ECKSTEIN, Rainer ; ECKSTEIN, Silke: *XML und Datenmodellierung. XML Schema und RDF zur Modellierung von Daten und Metadaten einsetzen*. Ringstrasse 19 B, 69115 Heidelberg : Dpunkt.Verlag GmbH, November 2003 (xml.bibliothek). – ISBN 978-3-89864-222-4
- [Erlenkötter 2003] ERLENKÖTTER, Helmut: *XML. Extensible Markup Language von Anfang an*. Hamburger Straße 17, 21465 Reinbek : Rowohlt Taschenbuch Verlag GmbH, September 2003 (Grundkurs Computerpraxis). – ISBN 978-3-499-61209-1
- [Feurle 2007] FEURLE, Daniel: *Personal Information Box - ein Tool zur einheitli-*

- chen Verwaltung heterogener Daten mittels Semantic Web Technologien*, Technische Universität Wien, Diplomarbeit, Dezember 2007
- [Fikes u. a. 2005] FIKES, Richard ; MCCOOL, Rob ; MCGUINNESS, Deborah: *OWL-QL Project for the Stanford Knowledge Systems Laboratory*. Internetseite. 2005. – URL <http://www.ksl.stanford.edu/projects/owl-ql/>. – Zugriffsdatum: 21.04.2007
- [Freeman 1997] FREEMAN, Eric T.: *The Lifestreams Software Architecture*, Yale University, Ph.D. Dissertation, Mai 1997. – URL <http://www.cs.yale.edu/homes/freeman/dissertation/etf.pdf>. – Zugriffsdatum: 18.12.2007
- [Gemmell u. a. 2002] GEMMELL, Jim ; BELL, Gordon ; LUEDER, Roger ; DRUCKER, Steven ; WONG, Curtis: *MyLifeBits: Fulfilling the Memex Vision*. In: *Proceedings of ACM Multimedia'02* (2002), Dezember, S. 235–238. – URL <http://research.microsoft.com/~jgemmell/pubs/MyLifeBitsMM02.pdf>. – Zugriffsdatum: 23.09.2007
- [Grant und Beckett 2004] GRANT, Jan ; BECKETT, Dave: *RDF Test Cases*. Internetseite. Februar 2004. – URL <http://www.w3.org/TR/rdf-testcases/>. – Zugriffsdatum: 02.06.2007
- [Gruber 1993] GRUBER, Thomas R.: *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. In: *Formal Ontology in Conceptual Analysis and Knowledge Representation* (1993), August. – URL <http://ksl.stanford.edu/knowledge-sharing/papers/>. – Zugriffsdatum: 07.04.2007
- [Guarino und Welty 2000] GUARINO, Nicola ; WELTY, Christopher: *Ontological Analysis of Taxonomic Relationships*. In: *Proceedings of ER-2000: The International Conference on Conceptual Modeling* (2000), Oktober. – URL <http://www.cs.vassar.edu/faculty/welty/papers/er2000/LADSEB05-2000.pdf>. – Zugriffsdatum: 07.12.2007
- [Guggenberger 2008] GUGGENBERGER, Thorsten: *Combining ontology alignment techniques - Mapping techniques relating ontologies*, Technische Universität Wien, Master Thesis, 2008
- [Hanh und Tjoa 2006] HANH, Hoang H. ; TJOA, A M.: *The Virtual Query Language for Information Retrieval in the SemanticLIFE Framework*. In: *Proceedings of International Workshop on Web Information Systems Modeling (CAiSE)* (2006). – URL [http://storm.ifs.tuwien.ac.at/publications/WISM06\\_hanh\\_tjoa\\_cr\\_final.pdf](http://storm.ifs.tuwien.ac.at/publications/WISM06_hanh_tjoa_cr_final.pdf). – Zugriffsdatum: 05.05.2007
- [Harmelen und Antoniou 2004] HARMELEN, Frank V. ; ANTONIOU, Grigoris: *A Semantic Web Primer*. 55 Hayward Street, Cambridge, MA 02142-1493, USA : MIT Press, Juli 2004 (Cooperative Information Systems). – ISBN 978-0-26201-210-2
- [Herman 1994] HERMAN, Ivan: *Semantic Web*. Internetseite. 1994. – URL <http://www.w3.org/2001/sw/>. – Zugriffsdatum: 08.04.2007
- [Hewlett-Packard Development Company 2007a] HEWLETT-PACKARD DEVELOP-

- MENT COMPANY, LP: *Extensions in ARQ*. Internetseite. 2007. – URL <http://jena.sourceforge.net/ARQ/extension.html>. – Zugriffsdatum: 07.11.2007
- [Hewlett-Packard Development Company 2007b] HEWLETT-PACKARD DEVELOPMENT COMPANY, LP: *Jena - A Semantic Web Framework for Java*. Internetseite. 2007. – URL <http://jena.sourceforge.net/index.html>. – Zugriffsdatum: 28.04.2007
- [Hewlett-Packard Development Company 2007c] HEWLETT-PACKARD DEVELOPMENT COMPANY, LP: *Writing Filter Functions for ARQ*. Internetseite. 2007. – URL [http://jena.sourceforge.net/ARQ/writing\\_functions.html](http://jena.sourceforge.net/ARQ/writing_functions.html). – Zugriffsdatum: 08.11.2007
- [Hoang u. a. 2006] HOANG, Hanh H. ; TJOA, A M. ; NGUYEN, Tho M.: *Ontology-based Virtual Query System for the SemanticLIFE Digital Memory Project: Concepts, Designs and Implementation*. In: *Proceedings of 4th IEEE International Conference on Computer Sciences* (2006), Februar. – URL [http://storm.ifs.tuwien.ac.at/publications/RIVF06\\_hanh\\_tjoa\\_tho\\_cr.pdf](http://storm.ifs.tuwien.ac.at/publications/RIVF06_hanh_tjoa_tho_cr.pdf). – Zugriffsdatum: 18.09.2007
- [Huynh u. a. 2002] HUYNH, David ; KARGER, David ; QUAN, Dennis: *Haystack: A Platform for Creating, Organizing and Visualizing Information Using RDF*. (2002). – URL <http://semanticweb2002.aifb.uni-karlsruhe.de/proceedings/Research/huynh.pdf>. – Zugriffsdatum: 23.09.2007
- [Jacobs 2004] JACOBS, Ian: *Architecture of the World Wide Web, Volume One*. Internetseite. Dezember 2004. – URL <http://www.w3.org/TR/webarch/>. – Zugriffsdatum: 16.12.2007
- [John und Drescher 2006] JOHN, Michael ; DRESCHER, Jörg: *Semantische Technologien im Informations- und Wissensmanagement*. In: (Pellegrini und Blumauer, 2006), S. 247–253. – ISBN 978-3-540-29324-8
- [Kargl u. a. 1997] KARGL, Maria ; WETSCHANOW, Karin ; WODAK, Ruth: *Kreatives Formulieren - Anleitung zu geschlechtergerechtem Sprachgebrauch*. Band 13. Ballhausplatz 1, 1014 Wien : Bundesministerium für Frauen Angelegenheiten und Verbraucherschutz, Juli 1997. – ISBN 3-9011-92227-8
- [Klyne u. a. 2004] KLYNE, Graham ; REYNOLDS, Franklin ; WOODROW, Chris ; OHTO, Hidetaka ; HJELM, Johan ; TRAN, Mark H. Butlerand L.: *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0*. Internetseite. Januar 2004. – URL <http://www.w3.org/TR/CCPP-struct-vocab/>. – Zugriffsdatum: 23.04.2007
- [Kommission der Europäischen Gemeinschaften 2003] KOMMISSION DER EUROPÄISCHEN GEMEINSCHAFTEN: *Mitteilung der Kommission an den Rat, das europäische Parlament, den Wirtschafts- und Sozialausschuss der europäischen Gemeinschaft und den Ausschuss der Regionen - Innovationspolitik: An-*

- passung des Ansatzes der Union im Rahmen der Lissabon-Strategie*. Internetseite. März 2003. – URL [http://ec.europa.eu/enterprise/innovation/communication/doc/innovation\\_comm\\_de.pdf](http://ec.europa.eu/enterprise/innovation/communication/doc/innovation_comm_de.pdf). – Zugriffsdatum: 13.03.2007
- [Lanzenberger u. a. 2007] LANZENBERGER, Monika ; SAMPSON, Jennifer ; RESTER, Markus ; NAUDET, Yannick ; LATOUR, Thibaud: *Visual ontology alignment for knowledge sharing and reuse*. In: *Journal of Knowledge Management* (2007). – ISSN 1367-3270
- [McBride 2006] MCBRIDE, Brian: *An Introduction to RDF and the Jena RDF API*. Internetseite. Dezember 2006. – URL [http://jena.sourceforge.net/tutorial/RDF\\_API/index.html](http://jena.sourceforge.net/tutorial/RDF_API/index.html). – Zugriffsdatum: 25.04.2007
- [McCarthy 2005] MCCARTHY, Philip: *Search RDF data with SPARQL - SPARQL and the Jena Toolkit open up the semantic Web*. Internetseite. Mai 2005. – URL <http://www-128.ibm.com/developerworks/xml/library/j-sparql/>. – Zugriffsdatum: 28.04.2007
- [Microsoft 2000] MICROSOFT (Hrsg.): *Computer-Fachlexikon mit Fachwörterbuch (deutsch-englisch/englisch-deutsch)*. Ausgabe 2000. Edisonstr. 1, 85716 Unterschleißheim : Microsoft Press Deutschland, 2000. – ISBN 3-86063-822-X
- [Moats 1997] MOATS, Ryan: *RFC2141*. Internetseite. Mai 1997. – URL <http://rfc.net/rfc2141.html>. – Zugriffsdatum: 16.12.2007
- [Pellegrini und Blumauer 2006] PELLEGRINI, Tassilo (Hrsg.) ; BLUMAUER, Andreas (Hrsg.): *Semantic Web. Wege zur vernetzten Wissensgesellschaft*. Springer-Verlag Berlin Heidelberg New York, 2006 (X.media.press). – ISBN 978-3-540-29324-8
- [Prud'hommeaux und Seaborne 2007] PRUD'HOMMEAUX, Eric ; SEABORNE, Andy: *SPARQL Query Language for RDF*. Internetseite. März 2007. – URL <http://www.w3.org/TR/rdf-sparql-query/>. – Zugriffsdatum: 22.04.2007
- [Reynolds 2007] REYNOLDS, Dave: *Overview of inference support*. Internetseite. Januar 2007. – URL <http://jena.sourceforge.net/inference/index.html>. – Zugriffsdatum: 28.04.2007
- [Seaborne 2004] SEABORNE, Andy: *RDQL - A Query Language for RDF*. Internetseite. Januar 2004. – URL <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>. – Zugriffsdatum: 17.12.2007
- [Signaturgesetz, § 2 2000] SIGNATURGESETZ, § 2: *Signaturgesetz, § 2. Rechtssystem - RIS*. Januar 2000. – URL <http://www.ris.bka.gv.at/bundesrecht/>. – Zugriffsdatum: 14.03.2007
- [Smith u. a. 2004] SMITH, Michael K. ; WELTY, Chris ; MCGUINNESS, Deborah L.: *OWL Web Ontology Language Guide*. Internetseite. 2004. – URL <http://www.w3.org/TR/owl-guide/>. – Zugriffsdatum: 09.04.2007
- [Tochtermann und Maurer 2006] TOCHTERMANN, Klaus ; MAURER, Hermann: *Se-*

- mantic Web - Geschichte und Ausblick einer Vision. In: (Pellegrini und Blumauer, 2006), S. 1–6. – ISBN 978-3-540-29324-8
- [W3C u. a. 2004] W3C ; BRICKLEY, Dan ; GUHA, Ramanathan V.: *RDF Vocabulary Description Language 1.0: RDF Schema*. Internetseite. Februar 2004. – URL <http://www.w3.org/TR/rdf-schema/>. – Zugriffsdatum: 30.03.2007
- [W3C und Quin 2003] W3C ; QUIN, Liam: *Extensible Markup Language (XML)*. Internetseite. 2003. – URL <http://www.w3.org/XML/>. – Zugriffsdatum: 16.12.2007
- [W3Schools 2007] W3SCHOOLS: *XSD Date and Time Data Types*. Internetseite. 2007. – URL [http://www.w3schools.com/schema/schema\\_dtypes\\_date.asp](http://www.w3schools.com/schema/schema_dtypes_date.asp). – Zugriffsdatum: 24.08.2007
- [Wichmann 2007] WICHMANN, Gabriele: *Entwurf Semantic Web - Entwicklung, Werkzeuge, Sprachen*. Dudweiler Landstr. 125 a, 66123 Saarbrücken : VDM Verlag Dr. Müller e. K., Februar 2007. – ISBN 978-3-83640-398-6
- [Wohlkinger und Pellegrini 2006] WOHLKINGER, Bern ; PELLEGRINI, Tassilo: *Semantic Systems Technologiepolitik in der Europäischen Union*. In: (Pellegrini und Blumauer, 2006), S. 115–131. – ISBN 978-3-540-29324-8
- [Wolf und Wicksteed 1997] WOLF, Misha ; WICKSTEED, Charles: *Date and Time Formats*. Internetseite. 1997. – URL <http://www.w3.org/TR/NOTE-datetime>. – Zugriffsdatum: 09.11.2007