Semantisches Web und Kontext –
Speicherung von und Anfragen auf RDF-Daten unter Berücksichtigung des Kontextes

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe – Universität
in Frankfurt am Main

von
Karsten Tolle
aus Korbach

Frankfurt 2006
(D 30)

vom Fachbereich Informatik und Mathematik der
Johann Wolfgang Goethe – Universität als Dissertation angenommen.

Dekan:	Prof. Dr.-Ing. Detlef Krömker
Gutachter:	Prof. Dott.-Ing. Roberto V. Zicari
	Prof. Dr. Oswald Drobnik
Datum der Disputation:	31. Mai 2006

Vorwort

Das *Semantische Web* (Semantic Web) stellt eine Weiterentwicklung des Internets dar, sodass Metadaten von Maschinen und Programmen *verstanden* werden können. Dies wird vielleicht in naher Zukunft unser alltägliches Leben und unser Arbeitsumfeld ähnlich stark beeinflussen, wie dies durch das Internet geschehen ist.

In dieser Arbeit befasse ich mich mit dem Speichern von und der Anfrage auf Daten des Semantischen Webs. Dabei wird sich auf das *Resource Description Framework (RDF)* als Fundament des Semantischen Webs konzentriert. Es werden das Speichersystem *RDF-Source related Storage System (RDF-S3)* und die Anfragesprache *easy RDF Query Language (eRQL)* vorgestellt, die von mir entwickelt wurden. Zielsetzung war dabei eine Lösung zu schaffen, die in einem Informationsportal eingesetzt und dessen Besuchern genutzt werden kann. Dabei standen die Intuitivität der Anfragesprache sowie die Glaubwürdigkeit und das Verständnis von Anfrageergebnissen im Vordergrund. Für die Glaubwürdigkeit und das Verständnis der Anfrageergebnisse spielen Herkunftsinformationen und der Kontext eine entscheidende Rolle. RDF-S3 und eRQL sind für deren Speicherung und Nutzung innerhalb von Anfragen ausgelegt.

Die Systeme RDF-S3 und eRQL sind frei als Open-Source verfügbar. Sie können zusammen mit weiterführender Literatur und Werkzeugen von mir aber auch von Studenten, die ich betreut habe, unter folgender Webseite[1] bezogen werden:

http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/

Wer immer mir Kommentare, Anregungen oder Kritik an der Arbeit übermitteln möchte, ist herzlich eingeladen, mir eine E-Mail an folgende Adresse zu schicken:

tolle@dbis.informatik.uni-frankfurt.de

Die vorliegende Arbeit wurde während meiner Tätigkeit an der Professur für Datenbanken und Informationssysteme (DBIS) der Johann Wolfgang Goethe-Universität Frankfurt am Main erstellt. Ohne die zahlreiche Unterstützung, die mir wiederfahren ist, hätte ich dieses Werk so nicht erstellen können.

Mein besonderer Dank gilt Herrn Prof. Dott.-Ing. Roberto V. Zicari, der die Arbeit betreut und konstruktiv-kritisch gefördert hat. Auch möchte ich mich bei Prof. Dr. Vassilis Christophides, Prof. Dr. Dimitris Plexousakis, Sofia Alexaki und Gregory Karvounarakis vom ICS-FORTH bedanken. Durch meine damalige Diplomarbeit am ICS-FORTH und die spätere Zusammenarbeit haben sie den fachlichen Grundstein dieser Arbeit gelegt.

Mein Dank gilt weiter all denjenigen, die mich in den letzten Jahren und vor allem in der Endphase der Arbeit unterstützt haben. Insbesondere sind dies: Prof. Dr. Christoph Schommer, Peter Werner, Jochen Sorell, Sascha (Alexander!-) Kaufmann, Natascha Höbel und Markus Michalek.

Ich danke allen Studenten, die ich in Seminaren und Diplomarbeiten mit dem Thema des Semantischen Webs genervt habe und die sehr gute Leistungen erbracht haben. Insbesondere möchte ich von diesen Fabian Wleklinski besonders danken, der mich selbst nach seiner hervorragenden Diplomarbeit weiter unterstützt hat und stets sehr hilfreiche Anmerkungen hatte.

[1] Die Gültigkeit aller in dieser Arbeit referenzierten Webseiten wurde am 03.12.2005 überprüft.

Zu guter Letzt möchte ich meiner Familie danken. Ein besonderer Dank dabei an meine Frau Astrid, die sich trotz fehlender Vorkenntnisse tapfer durch manche Texte von mir gekämpft hat. Ihr und meiner Tochter Julia-Grete möchte ich diese Arbeit widmen, hatte ich durch die Arbeit doch oftmals viel zu wenig Zeit für sie.

Inhalt

1 Einführung ... 1
1.1 Fokus der Arbeit ... 2
1.2 Abgrenzung der Arbeit und historische Entwicklung ... 4
1.3 Aufbau der Arbeit ... 7

2 Grundlagen und Hintergrundinformationen ... 9
2.1 Das Semantische Web ... 9
2.2 Einführung in das Resource Description Framework (RDF) ... 13
2.2.1 Ausführlicheres Beispiel – Kultur-Portal-Katalog ... 17
2.2.2 Weitere Konzepte in RDF ... 19
2.3 Probleme und Anmerkungen zur Entwicklung von RDF ... 20
2.4 Ein Blick in die Zukunft ... 26

3 Kontext im Semantischen Web ... 29
3.1 Kontext Verständnis ... 29
3.2 Kontext für das Semantische Web ... 31
3.2.1 Anwendungsfälle für den Gebrauch von Kontext in RDF ... 31
3.2.2 Aufteilung des Kontextes nach seiner Herkunft ... 32
3.2.3 Existierende Ansätze Kontext zu modellieren ... 34
3.2.4 Vergleich der Ansätze ... 36
3.3 Zusammenfassung ... 37

4 Speicherung von RDF-Daten plus Kontext ... 39
4.1 Datenmodelle zur Speicherung von RDF-Daten ... 39
4.2 Kombinierte Repräsentation (ComRepr) ... 41
4.2.1 Aufbau der kombinierten Repräsentation ... 42
4.2.2 Zusammenfassung der kombinierten Repräsentation ... 46
4.2.3 Nutzung der ComRepr für andere Kontext-Darstellungsarten ... 48
4.3 RDF-Source related Storage System (RDF-S3) ... 49
4.3.1 Funktionalitäten und Benutzungsoberflächen ... 50
4.3.2 Verbindung von VRP und RDF-S3 ... 53
4.3.3 Implementierung von RDF-S3 ... 54
4.3.4 Leistungsfähigkeit und Skalierbarkeit von RDF-S3 ... 62
4.4 Zusammenfassung ... 70

5 easy RQL – Anfragen mit Kontextunterstützung ... 73
5.1 Die Anfragesprachen RDQL und RQL ... 74
5.1.1 RDF Data Query Language (RDQL) ... 74

5.1.2 RDF Query Language (RQL) ... 75
5.2 easy RQL (eRQL) ... 76
5.2.1 eRQL Eigenschaften, Syntax und Nutzung ... 77
5.2.2 Implementierung, Ergebnistypen und Interpretation des Anfragebaums ... 86
5.2.3 Anfragegeschwindigkeit ... 97
5.3 Zusammenfassung ... 99

6 Zukünftige Arbeiten ... 101
6.1 Erweiterungsmöglichkeiten für RDF-S3 ... 101
6.1.1 Weitere Möglichkeiten der Aktualisierung ... 101
6.1.2 Behandlung von Schemaänderungen ... 102
6.2 Erweiterungsmöglichkeiten für eRQL ... 106
6.2.1 Erweiterungen auf funktioneller Ebene ... 106
6.2.2 Erweiterungen zur Verbesserung der Anfragegeschwindigkeit ... 107

7 Zusammenfassung ... 109

8 Anlagen ... 111
8.1 Syntaxbeschreibung von eRQL ... 111
8.2 Schnellübersicht der Klassenarchitektur für RDF-S3 ... 116
8.3 Schnellübersicht der Klassenarchitektur für eRQL ... 118
8.4 Installation und Start von RDF-S3 und eRQL ... 119
8.5 RDF-S3-Namensraum ... 120

9 Literaturverzeichnis ... 124

10 Lebenslauf ... 131

1 Einführung

Das *Semantische Web (Semantic Web)* wird als nächster Evolutionsschritt des Internets propagiert [Ber98, BHL01, BM02]. Hierbei werden zusätzlich zu den für Menschen gedachten Bildern, Musikdateien und Texten auch sogenannte *Metadaten* in einer standardisierten Form veröffentlicht. Diese Metadaten können von Maschinen gelesen und „*verstanden*“[2] werden. Durch das Verständnis der Bedeutung, der Zugriffsfähigkeit oder der Qualität der Daten können bessere und neue Dienste im Internet angeboten werden.

Ein Beispiel für den Nutzen des Semantischen Webs liefert die Internetsuche. Wird heutzutage mit einer Internetsuchmaschine nach dem Begriff „Jaguar“ gesucht, erhält man Informationen zum Jaguar als Katze, Auto, Flugzeug oder Fahrrad. Es gibt keine Möglichkeit für den Anfragesteller der Internetsuchmaschine von vorneherein und unmissverständlich mitzuteilen, nach welcher der verschiedenen Bedeutungen von „Jaguar“ sie suchen soll. Abgesehen davon wäre die Suchmaschine auch gar nicht in der Lage, die verschiedenen Bedeutungen von „Jaguar“ in den durchsuchten Texten zu erkennen und zu unterscheiden. Im Semantischen Web ist genau dies möglich: Suchmaschinen erkennen die verschiedenen Bedeutungen des Wortes „Jaguar“, und Anfragesteller können bei der Anfrage definieren, an welcher der Bedeutungen sie interessiert sind.

Ein komplexeres Beispiel stellt die Suche nach einer Frau mit den Nachnamen „Tolle“ dar, von der lediglich bekannt ist, dass ihr Mann an der Universität arbeitet. Mit der Stichwortsuche heutiger Suchmaschinen wäre ein Finden der gesuchten Information reines Glück – und das, obwohl es vermutlich nur sehr wenige Personen auf der Welt gibt, auf die diese vier Kriterien zutreffen: „Person ist weiblich“, „Person heißt *Tolle* mit Nachnamen“, „Person ist verheiratet“, „Gatte arbeitet an der Universität“. Selbst wenn die gesuchte Information im Ergebnis der Suchmaschine erscheinen sollte, wird sie von vielen anderen Treffern überlagert, „tolle Universitäten“ gibt es schließlich mehrere[3]. Mit dem Semantischen Web werden Antworten auf derlei Fragen möglich. Dies gilt insbesondere, da die einzelnen Daten im Semantischen Web miteinander vernetzt sein können. Ist die gesuchte Information nicht in einem Dokument zu finden, helfen heutige Suchmaschinen überhaupt nicht. Im Semantischen Web können Objekte über eine URI eindeutig identifiziert werden. Auch wenn die Informationen über ein Objekt auf verschiedene Dokumente verteilt sein sollte, ist es nun möglich diese Informationen wieder zusammenzufügen. In unserem Beispiel könnten also die einzelnen Kriterien auf verschiedene Dokumente verteilt vorliegen und trotzdem könnte die Frage beantwortet werden.

Kurzum: Das Semantische Web ermöglicht Maschinen das „verstehen“ von Webseiten und dadurch automatisch auch, Informationen verschiedener Webseiten miteinander zu verknüpfen.

Eine zentrale Rolle für das Semantische Web und für diese Arbeit spielt das *Resource Description Framework (RDF).* Es wurde unter Leitung des *World Wide Web Consortium (W3C)* entwickelt und legt fest, wie die Metadaten erstellt und für den Datenaustausch in XML kodiert werden müssen. Das zugrunde liegende RDF-Modell basiert auf einem gerichteten und beschrifteten Grafen. Die Knoten des Graphen werden Ressourcen und die Kanten Eigenschaften genannt. Zwei miteinander durch eine Eigenschaft verbundene Ressourcen werden als eine *RDF-Aussage* bezeichnet. Sie können als einfacher Satz mit Subjekt, Prädikat und Objekt verstanden werden. Durch das Zusammenfügen vieler kleiner Sätze können so komplizierte Sachverhalte beschrieben werden.

[2] Unter „*verstehen*“ wird in diesem technischen Zusammenhang das machinelle Auswerten anhand eines definierten Regelwerkes verstanden.

[3] Die Suchanfrage mit den beiden Begriffen *Tolle* und *Universität* lieferte bei *Google* 143.000 Ergebnisse (06. Juni 2005).

Neben dem RDF-Modell existiert die Beschreibungssprache *RDF-Schema (RDFS)*, welche ihrerseits wiederum auf dem RDF-Modell aufbaut. Durch RDFS ist es möglich eigene Eigenschaften, Klassen oder weitere RDF-Konstrukte zu beschreiben. Es wird dadurch jedem ermöglicht, sich ein eigenes Vokabular zu definieren, zu nutzen und zu veröffentlichen.

1.1 Fokus der Arbeit

Im Fokus der Arbeit steht eine Lösung zu finden, die für eine Internetsuchmaschine eingesetzt werden kann. Sollen über diese Anfragen an das Semantische Web ad hoc beantwortet werden, so ist dies nur möglich, wenn die Daten in einer zentralen Datenbank abgelegt sind. Mit dieser Technik arbeiten auch alle derzeit gebräuchlichen Internetsuchmaschinen. Aus den Erfahrungen dieser Suchmaschinen und deren Kampf gegen *Suchmaschinen-Spamming*[4] ist davon auszugehen, dass auch im Semantischen Web der Zukunft falsche und irreführende Informationen absichtlich veröffentlicht werden.

Wenn im Semantischen Web Maschinen das Filtern von Informationen für den Menschen übernehmen und beginnen, selbstständig Informationen miteinander zu verknüpfen, dann erhält die Glaubwürdigkeit dieser Informationen eine besondere Brisanz. Ist heutzutage eine angezeigte Internetseite für den Menschen (eventuell) noch als „Fälschung“ oder „Suchmaschinen-Spamming“ zu erkennen, so ist dies im Semantischen Web ohne weiteres nicht mehr der Fall. Einer einzelnen Aussage „[Subjekt Prädikat Objekt]“ lässt sich ihr Wahrheitsgehalt unmöglich ansehen.

Für die Nutzer des Semantischen Webs spielt daher die Glaubwürdigkeit von Anfrageergebnissen eine entscheidende Rolle. Ich halte daher die folgenden beiden Punkte für besonders wichtig:

- Damit entschieden werden kann, ob ein Ergebnis vertrauenswürdig ist oder nicht, muss die Herkunft der Daten angezeigt werden können.
- Um die Aktualität der Daten überprüfen zu können, muss die Speicher- oder Aktualisierungszeit zugänglich sein.

Für die Speicherung derartiger Metainformationen gibt es in RDF verschiedene Ansätze. Diese unterscheiden sich auch in ihrer Zuverlässigkeit. Um sicherzustellen, dass ein System nicht unterlaufen werden kann, halte ich eine klare Trennung der RDF-Daten von den Metadaten über diese für nötig.

Eine weitere entscheidende Rolle spielt die Anfragesprache. Um genau wie eine Internetsuchmaschine auch für unerfahrene Anwender nützlich zu sein, sind folgende Kriterien wichtig:

- Die Anfragesprache muss eine einfache und intuitive Bedienung ermöglichen.
- Die Anfrageergebnisse müssen verständlich sein. Hierfür ist es wichtig die Ergebnisse in ihrem Zusammenhang, also mit Kontextinformationen, zu präsentieren.
- Es sollte die Möglichkeit vorhanden sein die Anfrage auf bestimmte vertrauenswürdige Quellen zu begrenzen oder umgekehrt gezielt unglaubwürdige Quellen auszugrenzen.
- Die kombination von Anfragen auf Daten- und Schemainformationen sollte unterstützt werden.

[4] Suchmaschinen-Spamming, online unter: http://de.wikipedia.org/wiki/Suchmaschinen-Spamming

Existierende Speicher- und Anfragesysteme für RDF im Speziellen oder das Semantische Web im Allgemeinen werden diesen Anforderungen nur teilweise gerecht. Ich habe daher ein neues Speicher- und Anfragesystem entwickelt, welches die genannten Punkte abdeckt. Da noch eine hohe Unvertrautheit und Skepsis gegenüber dem Semantischen Web existiert, war es mir wichtig ein System zu erstellen, das einfach verstanden wird und leicht auszuprobieren ist. Um dies zu erreichen, habe ich entschieden auf gängige relationale Datenbankmanagementsysteme (RDBMS) als Speichersystem aufzubauen. Ich habe nur Eigenschaften verwendet, die von den meisten RDBMS unterstützt werden und die Portierung auf ein anderes RDBMS nicht erschweren[5]. Um die Portabilität weiter zu erleichtern, habe ich mich für die Implementierung in Java entschieden. Das Ergebnis ist das *RDF-Source related Storage System (RDF-S3)* zum Speichern von RDF-Dokumenten und die Anfragesprache *easy RDF Query Language (eRQL)*, die auf RDF-S3 aufsetzt. Beide sind frei als Open-Source auf der Webseite von RDF-S3[6] erhältlich.

Mit RDF-S3 und eRQL werden folgende Funktionalitäten realisiert:

- In RDF-S3 werden die einzelnen RDF-Aussagen zusammen mit ihren Herkunftsinformationen (Quelle und Speicherzeit) gespeichert.
- RDF-S3 bietet eine gute Wartbarkeit der gespeicherten Daten. Dies wird durch die Möglichkeit gezielt die Aussagen einzelner Quellen wieder zu entfernen oder zu aktualisieren realisiert.
- Durch die Kombination verschiedener Speichermodelle werden sowohl Daten- als auch Schemaanfragen durch RDF-S3 optimal unterstützt.
- eRQL bietet die Möglichkeit von kombinierten Anfragen auf Daten- und Schemainformationen.
- eRQL ist durch die Anlehnung an Anfragesprachen existierender Internetsuchmaschinen intuitiv nutzbar.
- In den Ergebnissen von eRQL-Anfragen können die Herkunftsinformationen der Daten beinhaltet werden. Hierdurch wird das Vertrauen in die Ergebnisse gesteigert und eine einfachere Weiterverwendung dieser ermöglicht.
- eRQL ermöglicht es, gezielt Anfragen auf eine Gruppe von Quellen zu begrenzen.
- Mit eRQL ist es möglich, eine Gruppe von Quellen für eine Anfrage auszugrenzen.
- Durch eine mögliche Integration von umgebenden Aussagen zu den Ergebnissen können diese leichter verstanden werden.

Die Implementierungen von RDF-S3 und eRQL umfasst zusammen fast 800 KB Quellcode und über 90 Klassen. In dieser Arbeit wird entsprechend auch auf die Implementierung eingegangen und Aufschluss über die internen Abläufe in den Systemen gegeben. Auch wird auf die Leistungsfähigkeit und Skalierbarkeit von RDF-S3 und eRQL eingegangen. Ein Überblick über die entwickelten Klassen mit Referenzen auf die jeweiligen Abschnitte dieser Arbeit, sind im Anhang unter 8.2 und 8.3 zu finden. Weitere Informationen, wie auch die Java-Dokumentation, sind unter der Webseite von RDF-S3 zu finden.

[5] Die Lauffähigkeit des entwickelten Speicher- und Anfragesystems wurde mit DB2 UDB v8.1 von IBM und mit Version 4.1 von MySQL erfolgreich getestet.

[6] RDF-S3 und eRQL sind online erhältlich unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/RDFS3/.

1.2 Abgrenzung der Arbeit und historische Entwicklung

Historische Entwicklung von RDF

RDF bildet die unterste Ebene des Semantischen Webs. Es stellt damit dessen Fundament dar. Dieses sollte entsprechend stabil sein. Es gibt jedoch noch offene Punkte. Ich konzentriere mich in diese Arbeit auf RDF. Höhere Ebenen, wie der *Web Ontology Language (OWL)*, betrachte ich nur am Rande.

Das erste *Arbeitspapier (Working Draft)* zur *RDF Model und Syntax Specification* wurde 1997 veröffentlicht. Anfang 1999 wurde diese Spezifikation als W3C-Vorschlag verabschiedet. Ergänzend gab es nur die *RDF Schema Specification 1.0*, die jedoch noch nicht den Status eines W3C-Vorschlages[7] *(W3C Recommendation)* erreicht hatte und später auf die Ebene eines Arbeitspapiers zurückgestuft wurde. Im Laufe der Zeit wurden vier weitere Spezifikationen in die RDF-Standard-Familie aufgenommen. In Abbildung 1 ist die zeitliche Entwicklung für die RDF-Standards visualisiert.

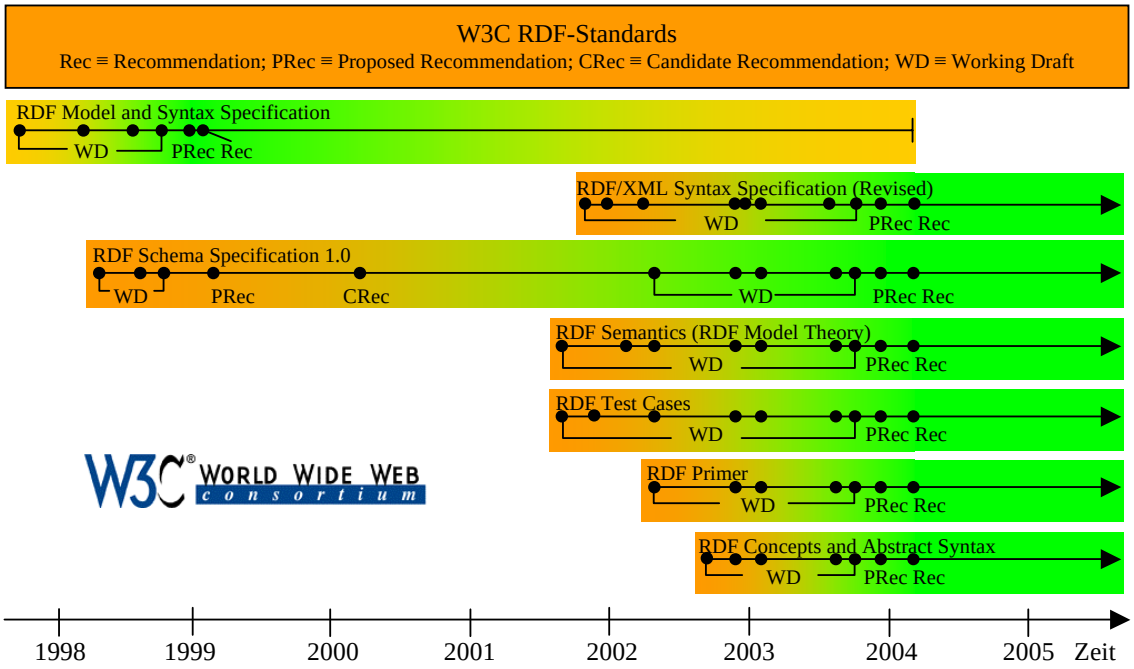


Abbildung 1 Übersicht über die zeitliche Entwicklung der RDF-Standards. Der Farbwechsel nach Grün soll dabei eine höhere Reife und Gültigkeit anzeigen.

Durch diese Darstellung möchte ich auf die durchaus turbulente Vergangenheit der RDF-Standards hinweisen. Der Umgang mit sich ständig ändernden Spezifikationen ist für die Entwicklung neuer Systeme und die Wartung Existierender durchaus ein Problem. Insbesondere da ich einige Änderungen für problematisch halte. Hierauf wird in 2.3 weiter eingegangen. Auch folgt aus den häufigen Änderungen, dass nur wenige Beschreibungen von RDF existieren, die wirklich den heutigen Spezifikationen entsprechen. Für Neueinsteiger, die lieber eine kurze Beschreibung lesen als den inzwischen über 80-seitigen *RDF Primer* [W3C Primer 04], führt dies schnell zur Verwirrung.

Abgrenzung von RDF zu anderen Standards

Die Darstellung von Informationen in einer Weise, die von Maschinen verstanden wird, ist nicht nur im Semantischen Web, sondern auch in vielen anderen Bereichen Ziel einiger

[7] Ein W3C-Vorschlag ist mit einem Standard gleichzusetzen.

Bestrebungen. Hilfreiche Verweise zu Vergleichen mit *UML/XMI* und *Topic Maps* sind unter der RDF-Startseite des W3C[8] zu finden. Da die mir am Häufigsten gestellte Frage dabei nach dem Unterschied von RDF gegenüber XML ist, möchte ich kurz die wichtigsten Unterschiede diesbezüglich erläutern.

Insbesondere im Bereich des E-Commerce (vorwiegend im B2B-Bereich) sind XML-Standards verfügbar, die Einigungen einzelner Branchenteilnehmer auf ein einheitliches Vokabular darstellen (auch *XML-Branchenstandards* genannt). Als eines von vielen Beispielen sei hier *FundsXML*[9] als Standard des *Bundesverband Investment und Asset Management e.v.* (BVI) genannt. Bleibt man in seiner Tätigkeit auf die entsprechenden Teilnehmer begrenzt, so ist sicherlich die Einigung auf einen Standard eine Lösung. Jedoch stellt dies eine Insellösung dar. Eine Transformation in einen verwandten Standard muss jeweils aufwendig erstellt werden. Aufwendig ist dies insbesondere, da in einem XML-Schema oder einer DTD nicht das OO-Modell beschrieben wird, sondern in welcher Reihenfolge einzelne Elemente in einer XML-Datei erscheinen dürfen. Das gleiche OO-Modell kann daher zu unterschiedlichen XML-Schemata oder DTDs führen. Die Arbeit von einem XML-Schema oder einer DTD auf das OO-Modell zu schließen ist sehr aufwendig – so genanntes *Reverse Engengering*. Dies führt auch bei späteren Anpassungen immer wieder zu zusätzlichem Aufwand. In RDF hingegen ist die Reihenfolge nebensächlich und ein RDF-Schema beschreibt bereits das OO-Modell, wodurch Zeit und Geld gespart werden kann [Dec+00]. Zwar können in RDF weiterhin unterschiedliche OO-Modelle von gleichen Sachverhalten erstellt werden, jedoch werden diese nicht durch Reihenfolgenzwänge verschleiert. Dadurch sind RDF-Modelle für den Menschen leichter zu verstehen und Anwendungen können entwickelt werden, die versuchen können Bedeutungen für ihnen unbekannte Vokabularien aus bekannten abzuleiten. Dies stellt sicherlich einen großen Vorteil von RDF dar.

Ein weiterer Vorteil ist die in [W3C Syntax 04] fest definierte Transformation von RDF/XML in eine andere Repräsentationsart. Diese kann daher immer durchgeführt werden, unabhängig ob die verwendeten RDF-Schemata zugänglich sind oder nicht. Daraus folgt, dass auch auf Teilinformationen von RDF/XML Dokumenten zugegriffen werden kann. In XML ist dies nicht möglich [Ber98].

Andere Anwendungen zur Speicherung und Anfrage von RDF-Daten

Bereits 1999 startete ich mit der Entwicklung des *Validating RDF Parsers (VRP)* am ICS-FORTH[10] und begann mich mit RDF und dem Semantischen Web zu beschäftigen. Damals war gerade die *RDF Model and Syntax Specification* als W3C-Vorschlag verabschiedet worden. Anfang 2000 wurde die erste Version von VRP veröffentlicht [Tol00, TC00]. Anschließend wechselte ich an die Johann Wolfgang Goethe-Universität in Frankfurt am Main. Meine Tätigkeit als wissenschaftlicher Mitarbeiter bot mir die Freiheit VRP weiter zu verbessern und am Aufbau der ICS-FORTH RDFSuite [Ale+01, Kar+04] mitzuwirken. Sie stellt ein Speicher- und Anfragesystem dar, die aus VRP, dem *RDF Schema Specific DataBase (RSSDB) Loader* und der Anfragesprache *RDF Query Language (RQL)* besteht. Die RDFSuite nutzt die Datenbank *PostgreSQL*[11] und deren objektrelationale Eigenschaften. Entwickelt wurde RDFSuite vorwiegend in Java und wurde auf der RDF-Startseite des W3C als Verweis für Java-Entwickler aufgenommen. Neben der RDFSuite kann dort auch das System *Jena* [McB02] von Hewlett-Packard gefunden werden. Jena unterstützt eine eigene Anfragesprache namens *RDF*

[8] RDF-Startseite des W3C, online unter: http://www.w3.org/RDF/

[9] FundsXML, online unter: www.funds-xml.org

[10] Institute of Computer Science-Foundation of Research and Technology - Hellas (ICS-FORTH)

[11] PostgresSQL, Open-Source Datenbank, online unter: http://www.postgresql.org

Data Query Language (RDQL). Später kam *Sesame* von ADUNA[12] hinzu. Sesame bietet sowohl eine Schnittstelle für RQL- als auch für RDQL-Anfragen[13]. Die drei Systeme RDFSuite, Jena und Sesame mit den Anfragesprachen RQL und RDQL sind sicherlich für die Entwicklung in Java, die am weitest verbreiteten. Aus diesem Grund konzentriere ich mich in dieser Arbeit auf die Vergleiche von RDF-S3 und eRQL mit diesen drei. Auf die Beschreibung von weiteren Systemen und deren Vergleich habe ich in dieser Arbeit verzichtet. Hierfür möchte ich auf den Bericht meiner Kollegen des ICS-FORTH unter [Mar+02] verweisen. Ein aktuellerer Vergleich ist unter [HBEV04] zu finden.

Die ersten Versionen von RDF-S3 und eRQL wurden von mir im Herbst 2003 veröffentlicht. Die damalige Version von eRQL beruhte auf der Arbeit von Fabian Wleklinski [Wle03][14]. In der Folgezeit wurde eRQL von mir erweitert und setzt seit Mitte 2004 auf RDF-S3 auf. Eine Übersicht über die zeitliche Entwicklung der für diese Arbeit wichtigsten Anwendungen bietet Abbildung 2.

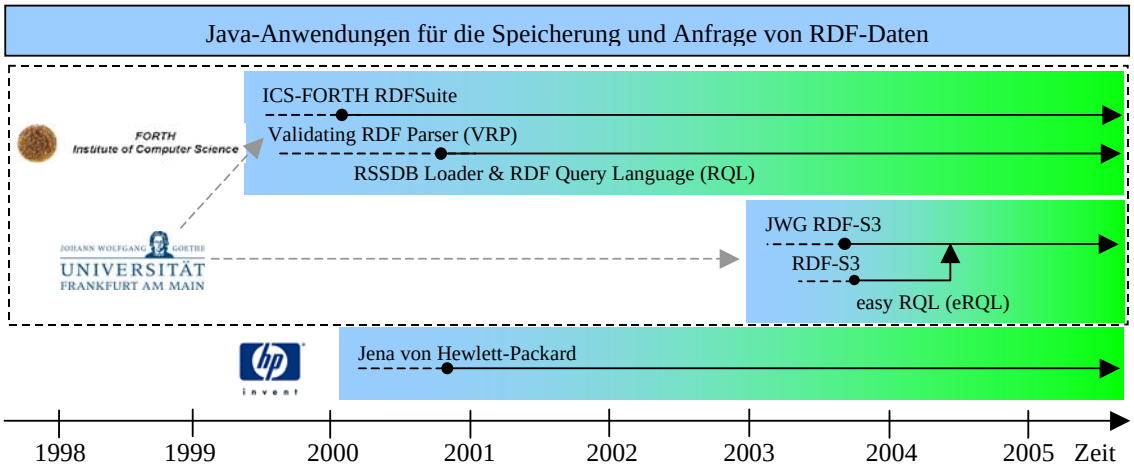


Abbildung 2 Übersicht über die zeitliche Entwicklung der für diese Arbeit vorwiegend relevanten Werkzeuge. Der Farbwechsel nach Grün soll dabei die wachsende Reife der Anwendungen verdeutlichen.

Durch den gegebenen Scherpunkt als Anwendungsbeispiel für eine Internetsuchmaschine liegt das Hauptaugenmerk in dieser Arbeit auf den Funktionalitäten, die für eine Internetsuchmaschine benötigt werden. Für eine intensive formale und allgemeine Betrachtung von RDF-Anfragen sei auf weiterführende Literatur [FHVB04, End04] verwiesen.

Neben den Internetsuchmaschinen existieren sogenannte *Metasuchmaschinen*. Metasuchmaschinen verfolgen einen verteilten Ansatz, indem sie die Anfrage an verschiedene Suchmaschinen weiterleiten und deren Einzelergebnisse in einem Gesamtergebnis präsentieren. Ein Beispiel hierfür stellt MetaGer[15] dar, über verschiedene deutschsprachige Suchmaschinen agiert. Im Bereich des Semantischen Webs ist QUEST (QUErying Specialized collecTions on the Web) [HMD00, Heß02] zu nennen. Für die Metasuchmaschinen stellt sich dabei vorwiegend die Frage, wie eine gegebene Anfrage auf die verschiedenen Suchmaschinen abgebildet werden muss und wie die einzelnen Ergebnisse zu einem Gesamtergebnis kombiniert

[12] Sesame ist online unter: http://openrdf.org zu finden. Es wird von der Firma ADUNA betreut (früher Aidministrator Nederland), online unter: http://aduna.biz/index.html.

[13] Mittlerweile wurde das *Kowari Metastore* System auf der RDF-Startseite des W3Cs hinzugefügt. Die erste Veröffentlichung dieses Speichersystems erfolgte etwa zeitgleich mit RDF-S3 und eRQL im Herbst 2003.

[14] Diese erste Version von eRQL beruhte nicht auf einer Datenbank, sondern die Anfragen wurden direkt an einzelne RDF-Dateien gestellt. Der hierfür entwickelte Prototyp ist ebenfalls frei erhältlich unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/eRQL/.

[15] MetaGer, die Metasuchmaschine über deutschsprachige Suchmaschinen, online unter: http://meta.rrzn.uni-hannover.de/

werden können. In dieser Arbeit werden diese Fragen nicht behandelt. Die intuitive Nutzbarkeit der Anfragesprache ist jedoch auch für Metasuchmaschinen relevant, sodass die Ansätze von eRQL auch für diese interessant sind.

1.3 Aufbau der Arbeit

In **Kapitel 2** wird genauer auf das Semantische Web eingegangen und dessen Ziele und Möglichkeiten aufgezeigt. Es wird aber auch erläutert, warum viele Menschen dem Semantischen Web sehr skeptisch gegenüberstehen. Anschließend wird im Abschnitt 2.2 RDF genauer beschrieben, welches das Fundament des Semantischen Webs darstellt. Es werden die Basiskonstrukte und Darstellungsarten von RDF an kleineren Beispielen erläutert. Ein größeres Beispiel fasst anschließend die wichtigsten Eigenschaften von RDF zusammen. In 2.3 werden Probleme und Anmerkungen zu Änderungen in den RDF-Standards aufgeführt. Diese wurden teilweise von mir in den RDF-Foren diskutiert und als noch zu klärende Punkte in [W3C RDF Tracking] aufgenommen.

In **Kapitel 3** wird darauf eingegangen, was unter *Kontext* zu verstehen ist. Es wird aufgezeigt, dass nur mithilfe des Kontextes aus einfachen Daten Informationen oder gar Wissen entstehen können. Dabei wird verdeutlicht, dass der Umgang mit Kontext schwierig ist. Eine Problematik ist die Abgrenzung des Kontextes. In 3.2 wird auf die Kontextnutzung speziell für RDF eingegangen. Durch die Möglichkeit Metainformationen in RDF darstellen zu können, werden diese oft mit gegebenen RDF-Daten vermengt. Dies hat jedoch Auswirkungen auf den Schutz gegenüber absichtlich falschen Aussagen, wodurch die Glaubwürdigkeit der Daten nicht mehr gegeben ist. Es wird eine Auftrennung des Kontextes in *internen* und *externen Kontext* vorgeschlagen, die nach der Herkunft der Daten die RDF-Daten von ihren Metadaten trennt. Anschließend werden die vorhandenen Darstellungsarten für Kontext in RDF vorgestellt. Dabei wird insbesondere die Möglichkeit der sicheren Trennung des internen und externen Kontextes betrachtet.

In **Kapitel 4** werden die beiden bekannten Datenmodelle zur Repräsentation von RDF in Datenbanken beschrieben – *generische Repräsentation (GenRepr)* und *schemaspezifische Repräsentation (SpecRepr)*. Diese Repräsentationsformen werden von Jena (GenRepr) bzw. der RDFSuite (SpecRepr) verwendet. In Abschnitt 4.2 wird ein neues Datenmodell präsentiert, welches beide Ansätze kombiniert und daher von mir *kombinierte Repräsentation* genannt wird. Das Datenmodell speichert zusätzlich für jede RDF-Aussage deren externe Kontextinformationen, wobei diese von den RDF-Daten klar getrennt werden, um das Glaubwürdigkeitsproblem zu lösen.

Anschließend wird im Abschnitt 4.3 RDF-S3 beschrieben, welches auf der kombinierten Repräsentation aufbaut. Es wird auf die einzelnen Funktionalitäten von RDF-S3 eingegangen und die internen Abläufe und Einzelheiten der Implementierung beschrieben. Insbesondere wird auf die Leistungsfähigkeit und Skalierbarkeit von RDF-S3 untersucht und die unterschiedlichen Maßnahmen vorgestellt, wie diese verbessert werden konnten.

Kapitel 5 befasst sich mit der eigenentwickelten Anfragesprache eRQL (*easy RDF Query Language*). Hier werden die Anforderungen an eine Anfragesprache für das Semantische Web aus Sicht der Benutzer beschrieben und die existierenden RDF-Anfragesprachen RQL und RDQL vorgestellt. Beide werden den beschriebenen Anforderungen jedoch nicht voll gerecht. In Abschnitt 5.2 werden daher die entwickelten Funktionalitäten und Nutzungsmöglichkeiten von eRQL präsentiert. Es wird ebenfalls auf die Implementierung und die genaue interne Interpretation von eRQL-Anfragen eingegangen. Auch werden einige Testergebnisse zur Leistungsfähigkeit gezeigt.

In **Kapitel 6** werden mögliche Erweiterungen und Ergänzungen genannt und vorskizziert. Für RDF-S3 wird dabei auf eine intelligentere Aktualisierungsstrategie eingegangen. Auch wird diskutiert, wie sich Änderungen von RDF-Schemata auswirken können, wenn man diese auf die

Daten propagieren möchte. Auch für eRQL werden Möglichkeiten zur Verbesserung und funktionellen Erweiterung gegeben.

Abschließend werden in **Kapitel 7** die Ergebnisse der Arbeit zusammengefasst. Hierbei werden diese in Bezug zu aktuellen Entwicklungen des W3C gesetzt.

2 Grundlagen und Hintergrundinformationen

Dieses Kapitel führt in die Grundlagen des Semantischen Webs im Allgemeinen und von RDF im Speziellen ein. Das *Resource Description Framework (RDF)* darf als Fundament des Semantischen Webs verstanden werden, und muss daher entsprechend stabil sein. Müssen Änderungen am Fundament vorgenommen werden, kann dies schwerwiegende Änderungen in darauf aufbauenden Standards nach sich ziehen. Der Schwerpunkt dieser Arbeit konzentriert sich daher auf RDF.

Zur Motivation werden in Abschnitt 2.1 zunächst kritisch die Vision und der Aufbau des Semantischen Webs beschrieben. Im anschließenden Abschnitt 2.2 wird in RDF und seine besonderen Merkmale eingeführt. Der Abschnitt 2.3 befasst sich mit Problemen, die in RDF existieren. Dabei wird auch auf Änderungen in den RDF-Standards und deren Effekt eingegangen. Im abschließenden Abschnitt 2.4 wird eine zusammenfassende Prognose für die weiteren Entwicklungen im Bereich des Semantischen Webs und RDF gegeben.

2.1 Das Semantische Web

Das *Semantische Web (Semantic Web)* wurde erstmals Ende der neunziger Jahre unter anderem von Tim Berners-Lee als Vision propagiert. Die Idee basiert darauf neben den Informationen, die man zur Kommunikation mit dem Menschen im Internet anbietet, zusätzliche Daten bereitzustellen, die von Maschinen und Programmen nicht nur gelesen, sondern auch „verstanden“ werden können. Unter „verstehen“ wird in diesem technischen Zusammenhang das maschinelle Auswerten anhand eines definierten Regelwerkes verstanden.

Die Vision des Semantischen Webs wird unter anderem in [BHL01, Fra01, Zie02] durch Anwendungsszenarien mit Leben gefüllt. Die Szenarios reichen von erweiterten Suchmöglichkeiten bis hin zu komplexen Aktionsketten. Ein Beispiel ist die selbstständige Reiseplanung und Buchung im Falle einer Registrierung zu einer Konferenz. Ein weiteres Beispiel ist eine automatische Terminabsprache mehrerer Personen zu einem Geschäftsessen durch den Abgleich elektronischer Terminkalender. Hierbei könnten persönliche Vorlieben und Wünsche in der Lösung berücksichtigt werden.

Als ultimatives Ziel des Semantischen Webs finden wir in [Fra01]:

> *„... the ultimate goal of the Semantic Web is to give users near omniscience over the vast resources of the Internet, turning the millions of existing database islands into a single gigantic database Pangea[16].“*

Von diesem ultimativen Ziel sind wir zurzeit noch weit entfernt und ob es überhaupt erreichbar ist, bleibt vorerst fraglich [Doc01]. Die weitere Entwicklung des Semantischen Webs ist kaum vorhersagbar. Sie bedingt immer mehr eine Kooperation mit benachbarten Forschungsgebieten wie Web Services, Peer-to-Peer, Ubiquitous Computing und Grid. Momentan scheint es so, dass viele Menschen das Semantische Web noch nicht ernst nehmen oder ihm abwartend gegenüberstehen. So sagte Mark Hapner[17] gegenüber ZDNet:

> *„RDF beziehungsweise das Semantische Web ist seiner Zeit gewissermaßen voraus: Aktuell geht es darum, alles aus XML herauszuholen, was den Unternehmen nutzen könnte. Dann muss RDF erstmal den Eintritt schaffen. Dazu braucht es ein oder zwei Killerapplikationen,*

[16] Als *Pangea* oder auch *Gondwana* wird der Urkontinent vor der Trennung in die heutigen Kontinente bezeichnet, online unter: http://www.lonlygunmen.de/natur/erde/pangea/pangea.html.

[17] Mark Hapner ist Chief Web Service Strategist von Sun, Zitat aus einem Interview mit ZDNet vom August 2003; online unter: http://www.zdnet.de/itmanager/tech/0,39023442,2138433,00.htm

sonst wird das Semantische Web zum Rohrkrepierer. Ich bin zwar sicher, dass RDF die Kurve bekommen wird, aber das kann noch dauern."

Eine gewisse Skepsis gegenüber dem Semantischen Web ist sicherlich gerechtfertigt. So vergleicht R. V. Guha laut [Fra01] die oft propagierte Möglichkeit des Ableitens (Inferenz) neuer Fakten aus Semantischen Web Daten mit dem *Traveling-Salesman Problem*[18]. Hierdurch wird klar, dass zwar vieles möglich ist, aber auch die Kosten hierfür beträchtlich sein können.

Ein weiteres Problem stellt die Komplexität des Semantischen Webs dar, die mittlerweile erreicht wurde. Zwar sind die Grundgedanken des Semantischen Webs noch einfach zu verstehen, jedoch stößt man allein beim Resource Description Framework auf mittlerweile sechs W3C-Vorschläge. Eine breite Nutzung im privaten Bereich ist daher in naher Zukunft nicht zu erwarten. Man beachte, dass RDF mit seinen sechs Standards dabei nur das Fundament (siehe Abbildung 3) des Semantischen Webs darstellt, und höhere Ebenen auf RDF aufbauen, welche die Komplexität des Semantischen Webs weiter erhöhen.

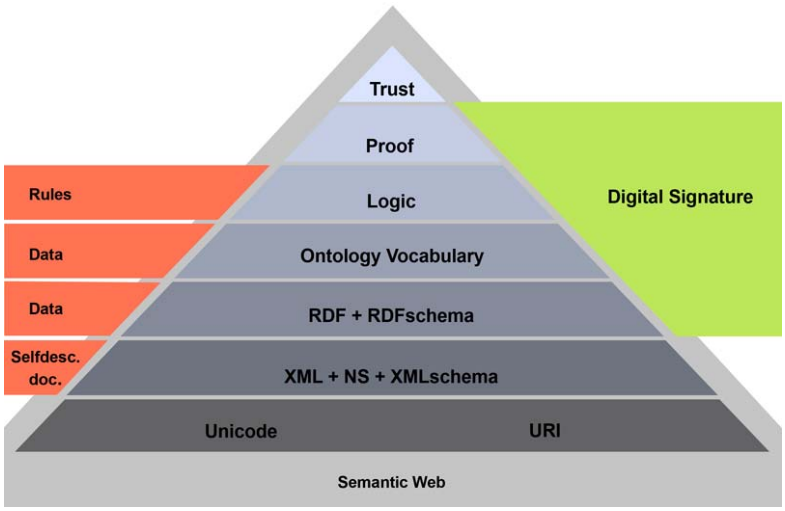


Abbildung 3 Schichtmodell der Standards für das Semantische Web.

Auch im kommerziellen Bereich gibt es Bedenken gegen das Semantische Web. Eine erhöhte Markttransparenz, zu der das Semantische Web beitragen kann, könnte zu zusätzlichem Preisdruck in den jeweiligen Märkten führen. Es ist durchaus nicht offensichtlich, wer letztendlich auf der Seite der Anbieter einen positiven Effekt aus dem Semantischen Web erzielen kann und wer nicht. Auch muss ein Umdenken im Bereich der Erfolgsmessung stattfinden. Erfolgsmessungen beruhen mit auf den sogenannten „Hits" – der Häufigkeit der Anzeige einer Webseite durch die Besucher. Johan Hjelm sagt hierzu in [Hje01]:

„..., this way of using metadata is scary to old-economy Chief Information Officers (CIOs) and Web designers. You are giving control to the users. You are giving information away, and it might decrease the number of hits on your pages. If your company still measures success on the Web in number of hits, fire the Webmaster and the CIO. …"

Trotz dieser angebrachten Skepsis ist der Nutzen des Semantischen Webs bzw. von RDF offensichtlich. So gibt es eine Vielzahl verschiedener Anwendungsbereiche für einen Gewinn bringenden Einsatz des Semantischen Webs. In [RCDS01] werden die Anwendungsbereiche in elf Kategorien unterschieden, wobei diese Kategorisierung nicht als endgültig angesehen werden sollte. Auch ist zu beachten, dass zwischen den Kategorien mehr oder weniger starke Überlappungen existieren. Um eine Vorstellung für das Potenzial des Semantischen Webs zu wecken, möchte ich die elf Kategorien hier nennen und kurz beschreiben:

[18] Auch „Problem des Handlungsreisenden" genannt, siehe online http://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden.

1. **Datenintegration** – Daten aus verschiedenen Quellen werden oftmals durch unterschiedliche Modelle beschrieben, obwohl gleiche oder ähnliche Sachverhalte zugrunde liegen. Um die Daten austauschen zu können, müssen die Modelle aufeinander abgebildet werden. Durch die RDF-Darstellung kann eine zusätzliche Schicht (ontology middleware) bereitgestellt werden. In ihr soll die Abbildung der Modelle durch Ableitungen und (semiautomatische) Abläufe erleichtert werden.
2. **Datenabhängige Agenten** – hiermit sind autonome Software-Systeme gemeint, die ihren Benutzern (Menschen oder anderen Agenten) durch das Verständnis der Daten und ihrer Verknüpfungen neuartige oder verbesserte Dienste anbieten.
3. **Wissensmanagement** – Ziel hierbei ist es, für ein Problem Informationen zu beschaffen, sodass das Problem gelöst werden kann. Hierfür ist der Zugriff auf verschiedenste Daten erwünscht. Oft liegen diese in unterschiedlicher Form vor, z. B. als Fallstudie, einfache Notiz, Projektbericht, usw. Wichtig ist dabei die Klassifikation und Strukturierung der unterschiedlichen Daten, sodass alle relevanten Daten zur Problemlösung genutzt werden können.
4. **Semantische Indizes und semantische Portale** – durch die Eindeutigkeit der Namensgebung in RDF können Objekte eindeutig identifiziert werden. Die Eindeutigkeit wird über *Uniform Resource Identifier (URI)* gewährleistet (mehr hierzu folgt in Abschnitt 2.2). Weiterhin können Zusatzinformationen mitgenutzt werden, um die Genauigkeit einer Anfrage zu erhöhen, unterschiedliche Ergebnisse besser nach ihrer Relevanz zu gewichten (*Ranking*) und exaktere Informationen zurückzugeben.
5. **Verwaltung persönlicher Daten** – ein klassisches Beispiel hierfür ist die Terminverwaltung, wobei Daten von verschiedenen Personen abgeglichen werden. Implementierungen hierfür sind auch im Peer-to-Peer Bereich zu finden. Auch die Erstellung und Handhabung eigener Benutzerprofile zur Personalisierung von Anwendungen gehört mit in diese Kategorie.
6. **Metadaten für Annotationen** – eine der grundlegenden Ideen des Semantischen Webs, wobei zusätzliche Informationen an Ressourcen „geheftet“ werden. Hierzu zählen Kommentare, Meinungen oder Bewertungen, die zur Bereicherung und zum besseren Verständnis der Ressourcen beitragen (z. B. Webseiten, Dokumente oder Grafiken).
7. **Metadaten für Beschreibung, Suche und Auswahl** – durch die semantischen Beschreibungen, die von Maschinen verstanden werden, können Anfragen beantwortet werden, die mit der zurzeit gängigen Volltextsuche der Internetsuchmaschinen nicht beantwortet werden können.
8. **Metadaten für Medien und Inhalte** – Metadaten werden intensiv für die Beschreibung von Medien wie Filmen oder Bilder genutzt, um diese zu verwalten oder zu bearbeiten. So wird RDF z. B. bereits heute von der Software Adobe Acrobat genutzt, um Dokumentmetadaten zu speichern (XMP[19]).
9. **Wissensgenerierung** – hierbei geht es um die Ableitung und das Finden neuer Erkenntnisse aus den vorhandenen Einzelinformationen.
10. **Management von Katalogen und Thesauri** – durch die Nutzung von RDF wird sich eine vereinfachte Handhabung von Katalogen und Thesauri erhofft.
11. **"Syndikation"** – hiermit ist die Veröffentlichung und Verbreitung von Metadaten gemeint, z. B. in Form von RSS[20]. Neben der Inhaltsbeschreibung ist hierbei auch die

[19] Extensible Metadata Platform (XMP), online unter: http://www.adobe.com/products/xmp/main.html

[20] RDF Site Summary (RSS), online unter: http://web.resource.org/rss/1.0/spec

Beschreibung interessant, wie und auf welchen Wegen (Kanälen) die Metadaten verteilt werden.

Auch in [FHLW03] wird in der Einleitung darauf eingegangen, welchen Nutzen das Semantische Web bringen kann. Dort werden vorwiegend die Bereiche Wissensmanagement und E-Commerce (B2C und B2B) angeführt und erläutert.

Das allgemein am weitesten verbreitete Argument, warum die Entwicklung des Semantischen Webs so wichtig ist, ist sicherlich die Datenflut (*data smog*) im Internet. Diese Masse an unstrukturierten Daten bereitet Probleme, was durch ständig neue Webseiten weiter verschärft wird. Der einfachen Volltextsuche gängiger Internetsuchmaschinen sind enge Grenzen gesetzt, die nahezu jedermann schon selbst erfahren hat. Oftmals ist das Ergebnis einer Suchanfrage überlagert von Informationen, die einen nicht interessieren – von Webseiten, welche die Suchbegriffe nur zufällig und in einem anderen Zusammenhang verwenden. Durch *Homonyme*[21] wird dieses Problem auf die Spitze getrieben. Der Anfragesteller muss sich daher oftmals aufwendig durch eine Vielzahl von Fundstellen kämpfen. Auch ist nicht klar, ob die gewünschte Information vielleicht durch eine andere Schreibweise oder durch Verwendung von Synonymen beschrieben wurde. Man findet sie daher nicht, obwohl sie im Internet verfügbar ist, da man nicht weiß, welche Schlagwörter man nutzen muss. Sucht man z. B. nach „Bank“, findet man all jene Webseiten nicht, welche nur das Wort „Kreditinstitut“ enthalten – wohl aber jene, die von einer „Bank im Park“ handeln: es werden zugleich „zu viele“ und „zu wenige“ Fundstellen ermittelt[22].

Es stellt sich nun die Frage nach der technischen Umsetzung des Semantischen Webs. Auf den Webseiten des W3Cs ist die folgende Beschreibung für das Semantische Web zu finden:

> *„The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming.”*[23]

Die einzelnen Standards zur Realisierung des Semantischen Webs kann man sich hierbei als Schichtenmodell vorstellen, wie es oben in Abbildung 3 dargestellt ist. RDF gilt als unterste Schicht des Semantischen Webs, welches seinerseits auf gängigen (Web-) Standards aufbaut. Man kann RDF daher auch als *Fundament des Semantischen Webs* bezeichnen. Für die darauf aufbauenden Standards ist es wichtig, dass dieses RDF-Fundament gut durchdacht und ausreichend getestet ist. Tim Berners-Lee wird in [Fra01] zitiert mit den Worten:

> *„In one way we don’t need to move too fast, because the theory people need to look at it to make sure we’re not too crazy, and other people need to check out the ideas in practice before they’re picked up and used too extensively.“*

Aus diesem Grund wird sich in dieser Arbeit auf RDF konzentriert. Die entstandenen Anwendungen *RDF-Source related Storage System (RDF-S3)* und *easy RQL (eRQL)* beziehen sich entsprechend direkt auf RDF und nicht auf die darauf aufbauenden Standards wie z. B. die *Web Ontology Language (OWL)*. Die Untersuchung der höheren Schichten bleibt aber wichtig. Nur hierdurch kann es ermöglicht werden Ziele, wie z. B. eine automatische Datenintegration, erreichen zu können. Es ist jedoch zu bedenken, dass mit jeder Schicht die Komplexität weiter steigt und so die Akzeptanzprobleme des Semantischen Webs weiter wachsen können. Ich halte es daher für sinnvoll zu versuchen, das volle Potenzial von RDF auszuschöpfen und es dadurch

[21] *Homonyme* sind Ausdrücke gleicher Laut- und Schreibweise, die dennoch unterschiedliche Bedeutungen haben.

[22] Für weitere Informationen und Beispiele hierzu sei auf die Einleitung von [Wle03] verwiesen.

[23] Semantic Web-Seite des W3C, online unter: http://www.w3.org/2001/sw/

populärer zu machen. Auch aus kommerzieller Sicht scheint die Ausnutzung des vollen Potenzials sinnvoll, nicht umsonst wird laut dem Zitat oben von Mark Hapner momentan für XML in der Wirtschaft genau dies getan.

2.2 Einführung in das Resource Description Framework (RDF)

Die *Resource Description Framework (RDF)* Standard-Familie des *World Wide Web Consortiums (W3C)* besteht insgesamt aus sechs W3C-Vorschlägen, die seit Februar 2004 gültig sind. In diesem Abschnitt werden die zum Verständnis dieser Arbeit nötigen Informationen gegeben. Eine ausführlichere Einführung in RDF ist durch den *RDF Primer* [W3C Primer 04] gegeben, der einer der W3C-Vorschläge ist. Alle sechs W3C-Vorschläge und weitere nützliche Informationen können unter der RDF-Startseite des W3C gefunden werden.

RDF dient zur Beschreibung und zum Austausch von Metadaten über sogenannte *Ressourcen*. Hierbei kann alles als Ressource interpretiert werden. Neben den klassischen *Web-Ressourcen* wie Internetseiten, die über einen eigenen *Uniform Resource Identifier (URI)*[24] und damit über eine eindeutige Adresse im Internet verfügen, können auch gegenständliche Dinge wie Bücher aber auch Imaginäres wie z. B. Märchenfiguren als Ressourcen angesehen werden. Um auf diese verweisen zu können, muss für sie eine URI erzeugt werden. Auch einfache Zeichenketten oder Werte, die in RDF *Literale* genannt werden, zählen zu den Ressourcen[25], da auch für diese eine URI erzeugt werden kann, die das Literal repräsentiert.

Die Grundidee von RDF ist, durch Zusammenfügen einfacher Aussagen komplexere Sachverhalte darzustellen. Die *RDF-Aussagen* entsprechen dabei einfachen Sätzen mit Subjekt, Prädikat und Objekt. Das einfache Datenmodell von RDF (*RDF-Modell*) basiert dabei formal auf einem gerichteten Grafen. Hierbei bilden die Subjekte und Objekte die Knoten. Die Prädikate bilden gerichtete und benannte Kanten, die vom Subjekt zum Objekt weisen. Sowohl die Knoten als auch die Kanten werden durch Ressourcen repräsentiert. Als Einschränkung gilt, dass für Subjekte und Prädikate *URI-Referenzen* verwendet werden müssen. Daraus folgt, dass Literale (auch wenn dies Ressourcen sind) nicht direkt als Subjekt oder Prädikat auftreten dürfen. Literale werden bei den gängigen grafischen Darstellungen als Rechtecke dargestellt, wohingegen anderen Knoten als Ovale repräsentiert werden (siehe Abbildung 4).

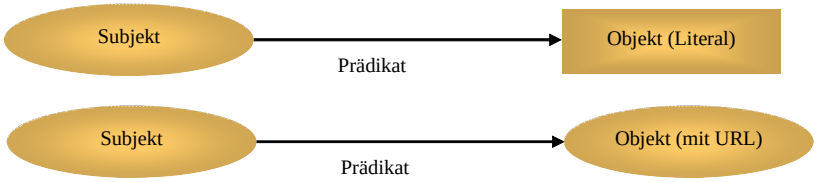


Abbildung 4 Grafische Darstellung von RDF-Aussagen.

Es wird also die „Subjekt-Ressource“ über die „Prädikat-Ressource“ mit der „Objekt-Ressource“ verbunden. Hierbei kann das Prädikat als Attribut (falls ein Literal als Objekt verwendet wird) oder auch als Relation angesehen werden. Es wird daher auch der Begriff der *Eigenschaft* anstelle von Prädikat verwendet. Die Eigenschaften sind selbst Ressourcen und müssen, wie oben erwähnt, über eine URI-Referenz verfügen. Diese verweist auf einen sogenannten *RDF-Namensraum*, in dem die Eigenschaft weiter beschrieben ist. Durch die Beschreibung im RDF-Namensraum kann die *Bedeutung (Semantik)* der Eigenschaft beschrieben werden. Man beachte, dass RDF-Namensräume hierdurch eine weitreichendere

[24] Die Menge der *Uniform Ressource Locator (URL)* stellen eine echte Teilmenge der URIs dar. Für weitere Informationen sei auf RFC2396 mit dem Titel *Uniform Resource Identifiers (URI): Generic Syntax* verwiesen, online unter: http://www.ietf.org/rfc/rfc2396.txt.

[25] Literale wurden in der Spezifikation von 1999 [W3C M&S 99] nicht als Ressourcen angesehen. Dies hat sich in der Entwicklung zum Standard von 2004 geändert.

Bedeutung haben, als dies bei Namensräumen in XML der Fall ist. RDF-Namensräume sollten existieren und ihre URL zugänglich sein, damit die Bedeutung des Vokabulars verstanden werden kann. Es wird weiter nur noch von Namensräumen gesprochen, womit RDF-Namensräume gemeint sind.

Ein Namensraum stellt ein *Vokabular* in Form einer Menge von RDF-Ressourcen bereit und beschreibt deren Bedeutungen. Namensräume werden ebenfalls mittels RDF beschrieben. Die Namensräume können auf andere Namensräume verweisen, wodurch die Wiederverwendbarkeit und Wartbarkeit der Namensräume erhöht wird. Neben den Eigenschaften können weitere RDF-Konzepte wie z. B. Klassen in den Namensräumen beschrieben werden. Auf diese RDF-Konzepte wird später noch eingegangen.

Mit der RDF-Standard-Familie werden zwei Namensräume vorgegeben, die ein Basisvokabular zur Erstellung und Beschreibung eigener Konstrukte bereitstellen und die für RDF notwendigen Strukturen beschreiben. Man spricht vom *RDF-Namensraum*, identifiziert über die URL *http://www.w3.org/1999/02/22-rdf-syntax-ns#*, in dem RDF-Basiskonstrukte enthalten sind. Werden Präfixe verwendet, wird dieser in dieser Arbeit mit *rdf* abgekürzt. Der zweite Namensraum dient zur Beschreibung und Erstellung eigener Vokabularien. Dieser wird auch *RDF-Schema-Namensraum* genannt. Er wird über die URL *http://www.w3.org/2000/01/rdf-schema#* identifiziert und mit dem Präfix *rdfs* abgekürzt. Die RDF-Ressourcen dieser beiden Namensräume bilden das *RDF-Kernvokabular*. Eine Übersicht des RDF-Kernvokabulars ist in [W3C Schema 04] zu finden. Jede Anwendung des Semantischen Webs sollte mindestens dieses Vokabular *verstehen* – es ist die Aufgabe des Entwicklers, dies zu realisieren. Auch die Erfassung der Bedeutung weiterer Vokabularien muss durch Entwickler in der Anwendung realisiert werden.

Um die Daten austauschen zu können, wurde eine verlustfreie XML-Serialisierung (oder XML-Kodierung) der Aussagen entwickelt, genannt *RDF/XML*. Diese wird in [W3C Syntax 04] festgelegt. Sie erlaubt Spielraum für z. B. verkürzte Schreibweisen, wodurch ein RDF-Graf zu unterschiedlichen XML-Codierungen führen kann. Bei einer Rückübersetzung dieser wird jedoch jeweils der Ausgangsgraf erzeugt. Wichtig ist die Unterscheidung, dass es sich bei RDF nicht direkt um eine XML-Anwendung handelt, sondern das RDF-Modell lediglich von und nach XML serialisiert werden kann.

Alternativ zur XML-Serialisierung existiert die bereits erwähnte eine Graf-Darstellung von RDF, die aber in der Regel von Menschen konsumiert wird und nicht für maschinelles Lesen geeignet ist.

Zusätzlich zur Darstellung als Graf oder als RDF/XML existiert die *Tripeldarstellung*. Hierbei werden die einzelnen RDF-Aussagen zeilenweise aufgeschrieben, wobei die Reihenfolge Subjekt, Prädikat und Objekt (*spo*) üblich ist. Es sind aber auch andere Reihenfolgen zu finden. Die Tripeldarstellung bietet die Möglichkeit kleinere Beispiele aufzuschreiben und auszutauschen, ohne Nutzung von Visualisierungstools für die Grafendarstellung oder aufblähende RDF/XML Syntax. Es gibt verschiedene Standards, welche die Tripeldarstellung optimieren und abkürzende, redundanzärmere Schreibweisen erlauben, z. B. Notation3, TriG, TriX. Der Einfachheit halber werden für die Tripeldarstellung in dieser Arbeit die einzelnen Aussagen durch eckige Klammern begrenzt. Als Trennzeichen zwischen den einzelnen Aussagekomponenten dient ein Leerzeichen. Literale werden durch doppelte Anführungszeichen begrenzt.

Bevor ein etwas ausführlicheres Beispiel gezeigt wird, sollen die drei Darstellungsarten an zwei kleinen Beispielen demonstriert werden. Angenommen wir wollen die folgende Aussage in RDF darstellen:

http://www.dbis.cs.uni-frankfurt.de/~tolle/RDF/index.html wurde erstellt von Karsten Tolle.

Bei dieser Aussage ist das Subjekt eine Webseite und kann über seine *Uniform Resource Locator (URL)* identifiziert werden. Das Objekt, hier *Karsten Tolle*, kann als Literal repräsentiert werden. Um das Subjekt und das Objekt zu verbinden, muss eine entsprechende Eigenschaft generiert oder gefunden werden, welche die Bedeutung „wurde erstellt von“ repräsentiert. Gehen wir davon aus, die entsprechende Bedeutung wird von der Eigenschaft http://www.my.de/vok#creator repräsentiert. Deren URI wird in diesem Fall durch das Anker Zeichen ‚#' aufgeteilt. Die voranstehende URL verweist auf den Namensraum, in dem die Eigenschaft definiert wird. Anschließend folgt der Name der Eigenschaft, hier „creator“. In Abbildung 5 werden die verschiedenen Darstellungsarten für die oben genannte Aussage präsentiert.

Tripeldarstellung:

[http://www.dbis.cs.uni-frankfurt.de/~tolle/RDF/index.html http://www.my.de/vok#creator „Karsten Tolle“]

Graf:

RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xmlns:my="http://www.myns.de/vok#" >
<rdf:Description rdf:about="http://www.dbis.cs.uni-frankfurt.de/~tolle/RDF/index.html">
        <my:creator>Karsten Tolle</my:creator>
</rdf:Description>
</rdf:RDF>
```

Abbildung 5 Eine einfache Aussage in Tripeldarstellung, Grafrepräsentation und RDF/XML.

Wird diese Aussage von einem Menschen gelesen, werden automatisch und unterbewusst zusätzliche Informationen und Erfahrungswerte genutzt, um die Aussage zu verstehen. Diese ermöglichen es, die Zeichenkette „Karsten Tolle“ mit hoher Wahrscheinlichkeit als Person zu interpretieren. Da das Ziel ist, einem Computer das Verstehen der Daten zu ermöglichen, sollte die Formulierung der RDF-Aussage nachgebessert werden. Etwas verfeinert könnte man folgende Aussagen treffen:

> *http://www.dbis.cs.uni-frankfurt.de/~tolle/RDF/index.html wurde von P erstellt.*
>
> *P ist eine Person.*
>
> *P hat den Namen ‚Karsten Tolle'.*

Dabei ist zu beachten, dass eine Person (hier *P*) an sich keine URI besitzt. Man könnte in diesem Fall eine Ressource definieren, welche die gegebene Person darstellen soll oder aber eine so genannte *anonyme Ressource* (*Blank Node*) verwenden. Eine anonyme Ressource besitzt zwar – wie jede andere Ressource auch – eine URI, allerdings ist die konkrete URI nicht wichtig, da sie nicht von außerhalb des Dokumentes referenziert wird. Die URI einer anonymen Ressource wird auch *Blank Node Identifier* genannt. Sie wird on-thy-fly vom Übersetzer beim Einlesen eines RDF-Dokumentes erzeugt. Deren genaue Struktur ist nicht vorgegeben, so dass sie in Abhängigkeit des Übersetzers unterschiedlich sein kann. Die URI einer nicht-anonymen Ressource andererseits darf nicht geändert werden, da sonst Referenzen auf diese URI verloren gehen.

Eine anonyme Ressource wird in der Grafrepräsentation als leeres Oval repräsentiert. Hierbei stehen zwei verschiedene leere Ovale auch für zwei verschiedene anonyme Ressourcen. Im folgenden Beispiel (siehe Abbildung 6) soll für *P* abkürzend als Blank Node Identifier „_:1“ verwendet werden. Auch werden die Namensräume in der Tripeldarstellung durch Präfixe substituiert, wodurch sich die Aussagen einfacher lesen lassen.

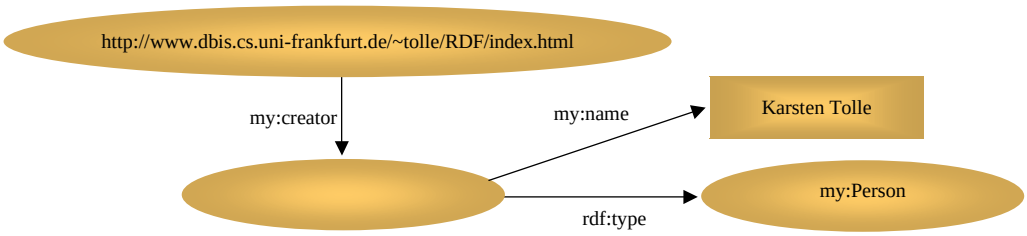
Tripeldarstellung:

[http://www.dbis.cs.uni-frankfurt.de/~tolle/RDF/index.html my:creator _:1]

[_:1 rdf:type my:Person]

[_:1 my:name “Karsten Tolle”]

Graf:



RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:my="http://mytestnamespace#" >
<rdf:Description rdf:about="http://www.dbis.cs.uni-frankfurt.de/~tolle/RDF/index.html">
       <my:creator>
              <my:Person>
                     <my:name>Karsten Tolle</my:name>
              </my:Person>
       </my:creator>
</rdf:Description>
</rdf:RDF>
```

Abbildung 6 Mehrere Aussagen mit der Verwendung einer anonymen Ressource in Tripeldarstellung, Grafrepräsentation und RDF/XML.

Die Eigenschaft *rdf:type*, mit der die anonyme Ressource mit dem Knoten *my:Person* verbunden ist, gehört zum RDF-Kernvokabular – erkennbar daran, dass sie im Namensraum mit dem Präfix „rdf“ untergebracht ist. Mit dieser Eigenschaft wird die anonyme Subjekt-Ressource als Instanz der Klasse *my:Person*, die als Objekt verwendet wird, deklariert.

Eine *Klasse* in RDF dient zum Zusammenfassen von gleichartigen Ressourcen. Eine Klasse wird dabei ihrerseits durch eine Ressource dargestellt. Ein Beispiel wäre die Klasse my:Person, die eine Menge von Personen repräsentiert. RDF-Klassen können hierarchisch strukturiert werden, wobei Instanzen einer Unterklasse ebenfalls Instanz der Oberklasse sind. RDF-Klassen können daher mit Klassen der objektorientierten Programmierung verglichen werden.

Im Namensraum *rdfs* ist die Klasse *rdfs:Class* definiert. Sämtliche Instanzen dieser Klasse (oder ihrer Unterklassen) sind Klassen im Sinne von RDF. Um RDF-Klassen zu erstellen, gibt es also kein spezielles Schlüsselwort oder einen ähnlichen Mechanismus, sondern es wird RDF-

konform einfach eine Ressource erstellt und via *rdf:type* der Klasse *rdfs:Class* zugewiesen. Die Klasse *rdfs:Class* ist dabei ein Element von sich selbst. Da dies an sich nicht geht, wird semantisch zwischen der Klasse als Ressource und deren ‚Abbild', also die Menge von Ressourcen, die eine Klasse zusammenfasst, unterschieden.

Neben Eigenschaften und den gerade erwähnten Klassen gibt es noch weitere Konzepte in RDF, die im Unterabschnitt 2.2.2 erläutert werden. Vorher soll ein etwas ausführlicheres Beispiel einige Besonderheiten des objektorientierten Modells von RDF verdeutlichen.

2.2.1 Ausführlicheres Beispiel – Kultur-Portal-Katalog[26]

Für ein besseres Verständnis von RDF und dessen Besonderheiten wird in diesem Abschnitt ein ausführlicheres Beispiel anhand eines elektronischen Kataloges für ein Kultur-Portal gegeben. Um den Katalog zu bilden, müssen die im Portal enthaltenen oder referenzierten Ressourcen, wie Museen, Ausstellungsstücke oder Künstler, beschrieben werden. In der Beschreibung können verschiedene Perspektiven kombiniert werden. Im Beispiel ist die Sicht eines Portal-Administrators zu finden, der sich für Dateigrößen und Dateiformate interessiert. Die zweite, enthaltene Sicht ist die Perspektive für Kunstinteressierte, deren Interesse den Kunstwerken und Künstlern gilt, jedoch nicht Dateigrößen oder Dateiformaten.

In Abbildung 7 wird eine entsprechende RDF-Beschreibung als Graf dargestellt. Der Graf ist in zwei Ebenen eingeteilt. Der untere Bereich enthält die Beschreibungen zweier Museen (Ressourcen &r4 und &r7) und drei Bilder von Kunstwerken, die auch auf den Webseiten der Museen zu finden sind (Ressourcen &r2, &r3 und &r6). Hierbei wird das Präfix & genutzt, um zu kennzeichnen, dass die URI der Ressource abgekürzt wurde. Die entsprechenden URIs lassen sich in Abbildung 7 unten rechts finden.

Betrachten wir zunächst die Ressource &r4. Einerseits ist sie als Klasse `ExtResource` beschrieben und besitzt Ausprägungen für die beiden zu dieser Klasse definierten Eigenschaften *`title`* und `last_modified`. Auf der anderen Seite ist &r4 auch Instanz der Klasse `Museum` und stellt so auch ein Museum dar, in welchem Kunstwerke ausgestellt werden. Dieses kann durch die Verknüpfung mithilfe der Eigenschaft `exhibited` geschehen, wie mit &r2 dargestellt. Die Ressource &r2 ist Instanz der Klasse `Painting`. Es existieren also Mehrfachklassifikationen für die Ressourcen &r2, &r3, &r4 und &r6 unter der Klasse `ExtResource`, sowie den Klassen `Painting` (&r2, &r3) beziehungsweise `Museum` und `Sculpture`. Schließlich werden Informationen zu den einzelnen Bildern hinzugefügt, wofür Ressourcen für die jeweiligen Künstler erstellt werden. Man beachte, dass die URIs &r1 und &r5 für die Künstler, welche natürliche Personen und somit keine Web-Ressourcen sind, innerhalb des Portals erzeugt wurden. Etwas genauer ist die Ressource &r1 eine Instanz der Klasse `Painter`. Verbunden mit &r1 sind über die Eigenschaft `fname` der Wert *Pablo* und über `nname` der Wert *Picasso*. Weiterhin findet eine Verknüpfung von &r1 zu den Ressourcen &r2 und &r3 mittels der Eigenschaft `paints` statt, worüber eine Verbindung zwischen &r2 und &r3 entsteht. Die Eigenschaften und Klassen werden dabei in RDF-Namensräumen definiert.

Fazit: Der untere Bereich der Abbildung 7 zeigt einen Teil des RDF-Modells, wie es für das Kultur-Portal definiert werden könnte. Es wird ersichtlich, dass RDF-Modelle verschiedene Perspektiven kombinieren können und beliebig granulare Informationen abbilden können.

Der obere Bereich zeigt zwei RDF-Namensräume, der Namensraum links mit dem Präfix *ns1* bzw. der URI http://www.icom.com/schema1.rdf definiert die für Administratoren interessanten RDF-Eigenschaften, wohingegen der Namensraum rechts mit Präfix *ns2* bzw. der URI

[26] Freie Übersetzung des Abschnitt 2 aus [Kar+03]. Die RDF/XML Dateien, die dieses Beispiel umfassen sind als RDF-Beispieldateien in der Distribution von RDF-S3 bzw. von VRP enthalten.

http://www.oclc.com/schema2.rdf die für Kunstinteressierte interessanten Eigenschaften definiert.

Die Eindeutigkeit der Namensgebung von Ressourcen (Klassen, Eigenschaften oder andere Ressourcen) wird durch Voranstellen der Namensraum-URL (üblicherweise gefolgt von einem Ankerzeichen ‚#') gewährleistet. Da es hier keine Namenskonflikte gibt, werden zum einfacheren Lesen die URLs der Namensräume im Folgenden weglassen.

Ressourcen, die als Eigenschaften verwendet werden, sind ihrerseits durch die Eigenschaft *rdfs:type* mit der Ressource *rdf:Property* verbunden. Dieses Detail ist in der Abbildung allerdings nicht enthalten. Der Leser kann es implizit entnehmen, da z. B. `creates` ganz offensichtlich als Eigenschaft verwendet wird (d. h. als Kante zwischen zwei Knoten der Grafrepräsentation).

Im Namensraum ns2 wird die Eigenschaft `creates` mit dem Definitionsbereich (rdfs:domain) `Artist` und dem Wertebereich (rdfs:range) `Artifact` definiert. Man beachte, dass Eigenschaften als Attribute (oder Charakteristika) von Ressourcen als auch für die Beschreibung von Beziehungen (oder Rollen) zwischen Ressourcen genutzt werden können. Zusätzlich können sowohl Klassen als auch Eigenschaften hierarchisch organisiert werden, wodurch auch eine semantische Inklusion inkl. Mehrfachvererbung ausgedrückt werden kann. Die Klasse `Painter` z. B. ist Unterklasse von `Artist` und die Eigenschaften `paints` (oder `sculps`) verfeinern die Eigenschaft `creates`.

In aller Kürze gesagt sind RDF-Eigenschaften *eigenständig* (d. h. unabhängig von Klassendefinitionen), *unsortiert* (z. B. es gibt keine Reihenfolge für die Eigenschaften `fname` und `lname`), *optional* (z. B. wird die Eigenschaft `material` nicht verwendet), *mehrfach verwendbar* (z. B. wird die Eigenschaft `paints` mit der Ressource &r1 zwei mal verwendet) und sie können *vererbt* werden (z. B. `creates`).

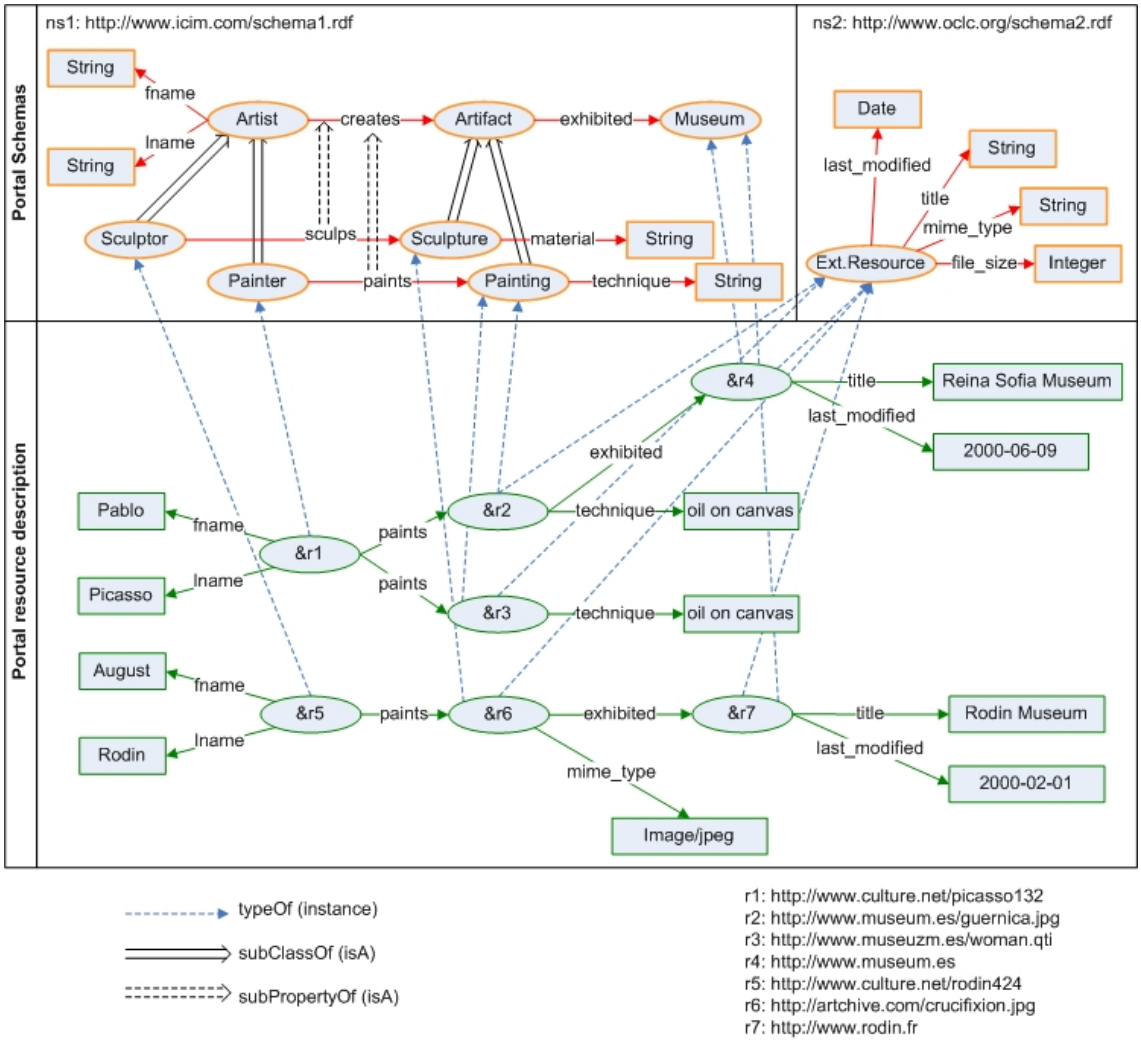


Abbildung 7 Der RDF-Graf für das Beispiel des Kultur-Portal-Katalogs.

2.2.2 Weitere Konzepte in RDF

In diesem Abschnitt werden weitere Konzepte von RDF erläutert, die zum Verständnis der weiteren Arbeit wesentlich sind[27].

Vergegenständlichte Aussagen (reified statements)

Durch vergegenständlichte Aussagen wird eine Aussage durch eine Ressource repräsentiert, wodurch sich Aussagen über Aussagen beschreiben lassen. Hierdurch lassen sich unter anderem indirekte Aussagen der Form „*A hat gesagt, dass ...*“ formulieren. Wie wir im Abschnitt 3.2.3 sehen werden, kann dies zur Darstellung von Kontextinformationen genutzt werden. Vergegenständlichte Aussagen werden dabei in der Klasse *rdf:Statement* zusammengefasst. Die Verknüpfung zu den einzelnen Teilen der Aussage, die vergegenständlicht wird, findet über die Eigenschaften *rdf:subject*, *rdf:predicate* und *rdf:object* statt.

[27] Als zusätzliche Lektüre empfiehlt sich weitere Fachliteratur, z. B. [W3C Primer 04, EE04].

Container

In RDF gehören die drei Containerarten *Multimenge*, *Sequenz* und *Alternative* zum Standard. Sie stellen eine einfache Möglichkeit dar, Ressourcen zu gruppieren. Zum Beispiel kann die Containerart *Multimenge* nützlich sein, wenn es darum geht, einer Ressource mehrere Autoren zuzuweisen. Die Containerart *Alternative* wiederum kann Verwendung finden, um einer Person mehrere Telefonnummern zuzuordnen.

Mit der mittlerweile aus dem RDF-Kernvokabular gestrichenen Eigenschaft *rdf:aboutEach* konnten Aussagen über alle in einem Container gruppierten Ressourcen aufgestellt werden. Auch wenn diese Eigenschaft nicht mehr zum RDF-Kernvokabular gehört, so ist die Grundidee sicherlich sehr hilfreich. Wir werden dies im Zusammenhang mit der Kontextdarstellung im Abschnitt 3.2.3 noch sehen. Von ihrer Bedeutung her können die einzelnen Containerarten wie folgt beschrieben werden:

- *Multimenge (Bag)* – eine unsortierte Menge, dabei wird nicht vorgeschrieben, wie mit Duplikaten umzugehen ist. Dies wird der jeweiligen Anwendung überlassen. Multimengen werden in der Klasse *rdf:Bag* zusammengefasst.
- *Sequenz* – eine Menge, wobei die Position der Elemente von Bedeutung ist. Sequenzen werden in der Klasse *rdf:Seq* zusammengefasst.
- *Alternative* – repräsentiert eine Wahlmöglichkeit, wobei das Element an der 1. Position als Standardwert angesehen wird. Alternativen werden in der Klasse *rdf:Alt* zusammengefasst.

Zum Einfügen einer Ressource in einen Container existieren Eigenschaften der Form *rdf:_nnn*, wobei *nnn* eine natürliche Zahl sein muss, also *rdf:_1, rdf:_2*, usw. Da dies unendlich viele sind, können sie nicht direkt im RDF-Namensraum definiert werden. Sie werden daher indirekt über ihre Obereigenschaft *rdfs:member* beschrieben. In der RDF/XML-Darstellung gibt es zusätzlich das Element *rdf:li*, welches bei der Überführung in eine andere Repräsentation vom jeweiligen Übersetzer interpretiert wird.

Listen (Collections)

Durch Listen werden einfach verkettete Listen realisiert. Dabei bilden Ressourcen vom Typ *rdf:List* die Listenelemente. Diesen Listenelementen können durch die Eigenschaft *rdf:first* Werte zugewiesen werden. Auf die folgenden Listenelemente kann mittels der Eigenschaft *rdf:rest* verwiesen werden. Weiterhin kann das Ende einer Liste angezeigt werden, indem mit *rdf:rest* auf das Listenelement *rdf:nil* verwiesen wird.

Listen werden in den RDF-Standards semantisch nicht weiter erläutert. Auch gibt es keine festgelegten Strukturvorgaben. Dies kann dazu führen, dass Listen entarten. Einem Listenelement können mehrere Werte zugewiesen werden. Auch ist die Existenz von *rdf:nil* als Listenende nicht vorgeschrieben bzw. muss nicht eindeutig sein. Es wird den RDF-Anwendungen überlassen, wie mit diesen Entartungen umzugehen ist.

Auf die Problematik von entarteten Listen, aber auch auf die Probleme allgemeiner Art, wird im nun folgenden Abschnitt weiter eingegangen.

2.3 Probleme und Anmerkungen zur Entwicklung von RDF

Im Rahmen dieser Arbeit und durch die Entwicklung und Weiterentwicklung des *Validating RDF Parser* (VRP[28]) seit 1999 bin ich auf viele Probleme aufmerksam geworden und habe sie in den entsprechenden Foren[29] des W3Cs diskutiert. In diesem Abschnitt möchte ich einige

[28] Der Validating RDF Parser (VRP) ist Teil der ICS-FORTH RDFSuite, online unter: http://athena.ics.forth.gr:9090/RDF/

[29] *W3C RDF Comments* Archiv, online unter: http://lists.w3.org/Archives/Public/www-rdf-comments/ und *W3C RDF Interest* Archiv, online unter: http://lists.w3.org/Archives/Public/www-rdf-interest/

Probleme aufzeigen, die ich für besonders wichtig halte. Einige beziehen sich auf Änderungen, die meiner Meinung nach überhastet durchgeführt wurden und nun zu neuen Problemen führen. Andere Probleme sind schon länger bekannt, jedoch noch nicht geklärt. Dabei möchte ich auch auf die Entwicklung und Veränderungen von RDF eingehen. Dies soll in erster Linie das Gesamtverständnis für RDF erhöhen, aber auch deutlich machen, dass die Arbeit mit sich noch entwickelnden Standards zwar sehr spannend aber auch sehr umfangreich ist.

Im Jahr 1999 wurde die *Resource Description Framework (RDF) Model and Syntax Specification* [W3C M&S 99] als W3C-Vorschlag verabschiedet. Neben dieser Spezifikation lag die Beschreibungssprache *RDF-Schema* damals als *W3C Proposed Recommendation* vor [W3C Schema 99], welches ein Vorläufer des heutigen W3C-Vorschlages [W3C Schema 04] ist. Bis zum Februar 2004 gab es immer wieder neue Versionen mit unterschiedlichen Empfehlungsstufen. Auch wurden zusätzliche Dokumente erstellt, welche die jetzigen sechs Standards der RDF-Standard-Familie ausmachen. Grafisch ist dies in Abbildung 2 der Einführung bereits verdeutlicht worden. Die Standards verfügen über Verweise zu ihren Vorgängerversionen, sodass deren Entwicklung zurückverfolgt werden kann. Unter [W3C Tracking] können neben der bisherigen Entwicklung auch weitere Änderungsanregungen gefunden werden, die beim W3C eingereicht wurden. Entsprechend den Entscheidungen wurden die Änderungsanregungen berücksichtigt, verschoben oder zurückgewiesen. Die jeweiligen Begründungen dieser Entscheidungen und weitere Informationen werden hier ebenfalls mit angegeben.

Einige Veränderungen von RDF möchte ich hier kurz aufführen. Die Veränderungen 3. – 6. und ihre Auswirkungen werden in den folgenden Anmerkungen genauer beschrieben.

Einige Veränderungen von RDF im Zeitraum 1999 bis 2004

1. Attribute nur mit Namensraumpräfix – Attribute ohne Namensraumpräfix in RDF/XML führten zu Verwirrungen zu welchem Namensraum diese gehörten (siehe auch [Tol01]). Die RDF-Arbeitsgruppe des W3Cs hatte daher im Mai 2001 entschieden, nur Attribute mit Namensraumpräfix zuzulassen.
2. Entfernung von *rdf:aboutEach* und *rdf:aboutEachPrefix* – mit *rdf:aboutEach* konnten Aussagen über alle in einem Container zusammengefassten Elemente getroffen werden. Die Eigenschaft *rdf:aboutEachPrefix* war noch allgemeiner und ermöglichte Aussagen über alle Ressourcen, deren URI-Referenzen mit einem gegebenen Präfix beginnen. Beide Eigenschaften wurden aufgrund mangelnder Unterstützung aus dem RDF-Vokabular entfernt. Aus eigener Erfahrung bezüglich der Entwicklung von VRP kann ich die Entscheidung nachvollziehen, das Attribut *rdf:aboutEachPrefix* zu streichen. Bei der Streichung von *rdf:aboutEach* ist dies anders, da seine Umsetzung in VRP keine Probleme bereitet hat. Auch wird im Kapitel 3 erwähnt, dass ein entsprechendes Attribut oder Eigenschaft für einige Kontextdarstellungsarten sinnvoll ist. Die Entfernung aus dem RDF-Kernvokabular heißt aber nur, dass das Attribut bei Bedarf als Eigenschaft selbst erstellt werden muss. Jedoch ist die Unterstützung einer selbst erstellten Eigenschaft durch vorhandene RDF-Werkzeuge nicht zu erwarten.
3. Zyklen in RDF-Klassen- bzw. Eigenschaftshierarchien – diese waren anfangs in RDF nicht erlaubt. Im Oktober 2001 wurde entschieden, Zyklen in Klassen- bzw. Eigenschaftshierarchien zuzulassen.
4. Listen (Collections) in RDF – am 08. November 2002 wurden in [W3C Syntax 02] erstmals Listen eingeführt.
5. Abschaffung von *rdfs:range* und *rdfs:domain* als Einschränkungen – beide Eigenschaften werden nur noch für zusätzliche Erzeugungen von *rdf:type* Aussagen verstanden.

6. Aufhebung der Trennung der Literale von den Ressourcen – während 1999 diese noch getrennt waren, zählen die Literale mittlerweile auch zu den Ressourcen.

Anmerkungen zu 3. – 6.:

zu 3.: Das Zulassen von Zyklen in RDF-Klassen- bzw. Eigenschaftshierarchien wurde von uns heftig kritisiert[30]. Trotzdem sind sie heute zulässig. Semantisch bedeutet ein Zyklus in einer Klassen- oder Eigenschaftshierarchie, dass alle beteiligen Klassen bzw. Eigenschaften gleich sind. In einer Datensenke kann so ein einfacher Eintrag eine komplette Klassen- oder Eigenschaftshierarchie *zerstören*. In einem so offenen System wie dem Semantischen Web, wo jeder alles sagen kann, ist dies sehr gefährlich[31]. Die Einführung von Zyklen für diesen Zweck ist auch daher nicht nachvollziehbar, da es mittels der OWL-Eigenschaften *owl:sameClassAs* bzw. *owl:samePropertyAs* die Möglichkeit gibt, die Gleichheit von Klassen und Eigenschaften unabhängig von der Hierarchie ausdrücken zu können. Diese Trennung vom Aufbau einer Hierarchie und dem Gleichsetzen von Klassen halte ich für weniger fehleranfällig. Auch haben wir in VRP gezeigt, dass das Abfangen von Zyklen in den Klassen- und Eigenschaftshierarchien auch über mehrere Namensräume hinweg möglich ist.

Ich würde daher in Anwendungen empfehlen Zyklen in Klassen- und Eigenschaftshierarchien stets abzufangen. Man könnte sie dann zulassen, wenn für alle beteiligen Klassen bzw. Eigenschaften mittels *owl:sameClassAs* bzw. *owl:samePropertyAs* die Gleichheit bestätigt wird.

zu 4.: Als Hauptargument für die Einführung von Listen in RDF, ist in [W3C Primer 03] zu lesen:

> *„A limitation of the containers is that there is no way to* close *them, i.e., to say, "these are all the members of the container". This is because, while one graph may describe some of the members, there is no way to exclude the possibility that there is another graph somewhere that describes additional members."*

Neben diesem Grund spielte auch der umgekehrte Fall eine Rolle. Wird ein Teilgraf aus einem RDF-Grafen extrahiert, gibt es ohne weiteres keine Möglichkeit bei einem Container zu prüfen, ob alle Elemente eines Containers, der im Teilgrafen enthalten ist, mit extrahiert wurden. Durch die Konstruktionsweise von Listen sollten diese Probleme behoben werden. Dabei wurde argumentiert, dass durch die Verwendung von anonymen Ressourcen als Listenelemente keine weiteren Einträge von anderen Dokumenten eingetragen werden können. Weiterhin kann durch *rdf:nil* das Ende einer Liste deutlich gemacht werden. Sollten beim Extrahieren Elemente der Liste entfernt werden, so wäre die Liste an einer Stelle unterbrochen oder der *rdf:nil* Eintrag nicht mehr vorhanden.

Leider wurde es versäumt Bedingungen bezüglich der Wohlgeformtheit oder Gültigkeit von Listen zu erstellen[32]. So ist nirgends vorgeschrieben, dass es in einer Liste nur genau ein *rdf:nil* geben darf. Auch können Listen Ressourcen als Listenelementen enthalten, die nicht anonym sind. Hierdurch kann von anderen Dokumenten aus auf diese verwiesen und damit weitere Listeneinträge angehängt werden. Weiterhin ist es möglich einem Listeneintrag verschiedene Werte zuzuweisen, indem mehrfach die Eigenschaft *rdf:first* für den

[30] Vassilis Christophides, Karsten Tolle: CYCLES IN CLASS/PROPERTY HIERARCHIES. Mail vom 26. Oktober 2001, online unter: http://lists.w3.org/Archives/Public/www-rdf-interest/2001Oct/0131.html

[31] Dies gilt umso mehr für Datensenken, die mit den Aussagen keine Quellinformationen speichern und so ein späteres Lösen eines solchen Problems erschweren.

[32] Karsten Tolle: Clarifications needed for the Collection construct (with CR). Mail vom 21. Februar 2003, online unter: http://lists.w3.org/Archives/Public/www-rdf-comments/2003JanMar/0345.html

Listeneintrag genutzt wird. Dies müsste nach Pat Hayes[33] so interpretiert werden, dass die verschiedenen eingetragenen Werte als gleich anzusehen sind. Er bezieht sich dabei aber bereits auf die OWL-Ebene. Die Diskussion wurde als *Last Call Comment* in [W3C LC 03] akzeptiert und aufgenommen.

Durch die fehlende Definition der Wohlgeformtheit bzw. Gültigkeit bin ich der Überzeugung, dass keines der beiden Ziele für Listen wirklich erfüllt wird. Dies wurde von mir in [TAC03] noch einmal ausdrücklich klargestellt und mit Beispielen untermauert. Zusätzlich zeige ich dort, dass Container genauso abgeschlossen werden können wie Listen, indem zur Erstellung des Containers eine anonyme Ressource genutzt wird. Graham Klyne zeigt schließlich wie Container mittels OWL-Beschränkungen beschrieben werden können, sodass auch die Anzahl der Elemente überprüft werden kann, auch wenn mit Teilgrafen des Ursprungsgrafen gearbeitet wird[34]. Damit zeigt sich, dass beide Ziele bereits mit Containern dargestellt werden können und Listen diesbezüglich keinen Vorteil bieten.

Im aktuellen RDF Primer [W3C Primer 04] wird auf die Möglichkeit Listen offen zu gestalten deutlich besser hingewiesen, als dies noch in der Version von Januar 2003 [W3C Primer 03] der Fall war. Weiterhin liegt die Verantwortung jedoch bei den Anwendungen zu prüfen, ob die Listen den jeweiligen Kriterien genügen. Dies ist zwar immer möglich, sorgt jedoch für deutlich mehr Aufwand bei der Anwendungsentwicklung und verringert die Kompatibilität zwischen RDF-Dokumenten und Anwendungen.

Zusätzlich wirft die Existenz von Containern und Listen ein schwerwiegendes Problem auf:

Es existieren nun in RDF zwei Konstrukte, welche semantisch sehr ähnlich (in vielen Fällen sogar gleich) sind.

Beide können zur Gruppen- oder Mengenbildung zusammengehöriger Ressourcen genutzt werden. Eine genauere semantische Definition für Listen fehlt jedoch, im Gegensatz zu den Containern, vollständig. Möchte jemand eine Menge oder Sequenz in RDF darstellen, hat er die Wahl zwischen beiden Konstrukten. Leider fehlt auch eine Abbildung von Listen auf Container. Immerhin wurde diese Anregung von mir in [W3C Tracking] aufgenommen, bis jetzt jedoch immer verschoben.

Ich denke hier wurde ein wenig voreilig eine Änderung eingeführt, die viele Probleme bereitet hat und noch bereiten wird. Ein weiterer Grund für die Einführung von Listen ist im Kommentar unter Anhang A.3 von [W3C Syntax 02] zu entdecken. Dort findet man:

> *„The ordinal property representation of containers does not support recursive processing of containers in languages such as Prolog."*

Wenn man sich hierzu den Bericht der entsprechenden Telefonkonferenz ansieht, wird jedoch auch klar, dass neben dem Vorteil für einzelne Programmiersprachen, auch die Konsistenz mit den höheren Ontologie-Standards (DAML[35]) erhöht werden sollte. Dort existierte bereits die *daml:collection*, welche mit dem Konstrukt der Listen in RDF vergleichbar ist.

---

[33] Pat Hayes: Re: Clarifications needed for the Collection construct (with CR). Mail vom 21. Februar 2003, online unter: http://lists.w3.org/Archives/Public/www-rdf-comments/2003JanMar/0364.html

[34] Graham Klyne: Closing RDF containers. Mail vom 10. November 2003, online unter: http://lists.w3.org/Archives/Public/www-rdf-logic/2003Nov/0021.html

[35] DARPA Agent Markup Language (DAML), bildet zusammen mit dem Ontology Inference Layer (OIL) die Vorstufe zur jetzigen Web Ontology Language (OWL).

Auch finde ich bedenklich, dass die Interpretation, die Pat Hayes[36] für mehrere Werte eines Listenelementes gibt, nämlich dass diese als gleich interpretiert werden müssen, rein nach [W3C Semantic 04] nicht getroffen werden kann. Gleiches gilt auch für Sequenzen in RDF. In der Beschreibung für die einzelnen Containertypen wird in [W3C Schema 04] mittlerweile so weit gegangen, dass die gegebene Typeinteilung nur als Hilfe für den menschlichen Leser gedacht ist. Dies ist für mich absurd, da das Ziel ja gerade ist, die Daten für Maschinen verständlich zu machen.

zu 5.: Die Eigenschaften *rdfs:range* und *rdfs:domain* wurden anfangs als Einschränkungen eingeführt. Damit sollte eine Eigenschaft im Gebrauch auf die entsprechenden Klassenelemente für den Werte- bzw. Definitionsbereich beschränkt werden. Diese Interpretation ist nur noch für eine informelle semantische Erweiterung, wie sie in [W3C Semantic 04] unter 4.2 beschrieben ist, gültig. Generell wurde hier entschieden allen Möglichkeiten aus dem Weg zu gehen, die einen RDF-Grafen ungültig machen könnten. Daher gilt nun, dass falls eine Eigenschaft über *rdfs:range* und/oder *rdfs:domain* Beschreibungen verfügt, dies bei der Nutzung der Eigenschaft zu entsprechenden *rdf:type*-Aussagen für Subjekt bzw. Objekt führt. Für die Datenqualität und Überprüfbarkeit von RDF-Daten ist diese Entscheidung mehr als bedenklich. Wird dies von RDF-Standard-Anwendungen (z. B. Übersetzern) umgesetzt, haben Anwendungen, die auf diesen aufbauen keine Möglichkeit mehr eine Überprüfung (entsprechend einer semantischen Erweiterung wie in [W3C Semantic 04] beschrieben) durchzuführen. Auf jeden Fall ist die Anwendungsentwicklung jedoch erschwert worden, da Einschränkungen in den jeweiligen Anwendungen implementiert und gepflegt werden müssen.

Eine zusätzliche Problematik sehe ich hier im Bezug auf Literale. Diese dürfen nicht als Subjekt oder Prädikat innerhalb einer RDF-Aussage verwendet werden. Seien nun die folgenden RDF-Aussagen gegeben:

[ex:p1 rdf:type rdf:Property]

[ex:p1 rdfs:range ex:Person]

[ex:Person1 ex:p1 “Karsten Tolle”]

Die Aussagen beschreiben eine Eigenschaft, die als Wertebereich die Klasse *ex:Person* besitzt. In der dritten Aussage wird diese Eigenschaft jedoch mit einem Literal verwendet. Durch die neue Interpretation von *rdfs:range* müsste daraus folgen, dass die folgende Typzuweisung gültig wäre, die jedoch ein Literal als Subjekt enthält:

[“Karsten Tolle” rdf:type ex:Person]

Neben der ehemals gültigen Einschränkung, dass es für Eigenschaften nur eine Klasse als Definitionsbereich geben darf, wurden auch die Klassen *rdfs:ConstraintResource* und *rdfs:ConstraintProperty* mittlerweile gestrichen.

zu 6.: Aufhebung der Trennung der Literale von den Ressourcen – während 1999 diese noch getrennt waren, zählen die Literale mittlerweile ebenfalls zu den Ressourcen. Zwar bleiben Literale ohne URI-Referenz von Ressourcen mit URI-Referenz unterscheidbar, jedoch werden beide als Ressourcen aufgefasst. Einzige Einschränkung ist, dass Literale weiterhin nur als Objekte auftreten dürfen. Es gibt jedoch auch Bestrebungen Literale als Subjekte zuzulassen. In [W3C Semantic 04] wird unter 4.3 noch einmal gesondert auf die Interpretation von Literale und der Klasse *rdfs:Literal* eingegangen. Hiernach ist es auch erlaubt URI-Referenzen als Instanzen der Klasse *rdfs:Literal* zu deklarieren.

[36] Pat Hayes: Re: Clarifications needed for the Collection construct (with CR). Mail vom 21. Februar 2003, online unter: http://lists.w3.org/Archives/Public/www-rdf-comments/2003JanMar/0364.html

Die Aufhebung hat, denke ich, auch mit dem vorherigen Problem bezüglich Container und Listen zu tun. Da es keine Wohlgeformtheit und Gültigkeit gibt, kann durch die Mehrfachnutzung bestimmter *ContainerMembershipProperties* (*rdf:_nnn*) oder von *rdf:first* für ein Listenelement, die Gleichheit eines Literals zu einer URI-Referenz dargestellt werden[37]. Durch die Aufhebung der Trennung und die spezielle Interpretation in [W3C Semantic 04], bereitet es kein Problem mehr zu sagen, eine Ressource mit URI-Referenz und ein Literal seien gleich. Man beachte jedoch, dass zur Gleichheit der Literale in [W3C Concept 04] unter 6.5.1 zu finden ist:

„*Two literals are equal if and only if all of the following hold:*

- *The strings of the two lexical forms compare equal, character by character.*
- *Either both or neither have language tags.*
- *The language tags, if any, compare equal.*
- *Either both or neither have datatype URIs.*
- *The two datatype URIs, if any, compare equal, character by character.* "

Würde man sich hieran halten, würde das Problem erneut für mehrere Literale an einem Listeneintrag auftreten. Die obige Definition ist jedoch mehr auf syntaktischer Ebene zu sehen. Auf semantischer Ebene ist es sinnvoll auch verschiedene Literale als gleich bezeichnen zu können, z. B. wenn der gleiche Wert in verschiedenen Sprachen dargestellt wird.

Es ist hierzu zu bemerken, dass auch ein Zusammenlegen der Knoten syntaktisch gleicher Literale innerhalb eines RDF-Grafen problematisch ist. Hat man z. B. zwei Personen, die beide mit Nachnamen *Müller* heißen, so ist nach einem Zusammenlegen der Knoten ein getrenntes Ändern nicht mehr möglich (siehe auch [Kar+03]).

Weitere Probleme:

- Identifikation von Objekten mittels URI – ein durchaus bekanntes, jedoch wenig publik gemachtes Modellierungsproblem, stellt die Identifikation von Objekten mittels der URI-Referenzen dar. In RDF ist nicht erkennbar, ob mit einer Ressource eine Web-Ressource gemeint ist, oder auf ein Objekt außerhalb des Webs verwiesen werden soll. Dies könnte durch zusätzliche Klassen, z. B. *RealWorldObject* und *WebResource*, wie wir sie in der *Metaschemaebene* in [Kar+04] eingeführt haben, behoben werden.

  Dieses Problem wird ausführlich in [PS03] beschrieben, weshalb hier auf eine genauere Beschreibung verzichtet wird. Dort wird auch auf *Topic Maps*[38] eingegangen, die ein ähnliches Ziel wie RDF verfolgen. In Topic Maps wird dieses Problem durch die Nutzung verschiedener Elemente behoben. So gibt es das Element *subjectIdentity*, welches zusammen mit dem Element *resourceRef* zur Beschreibung einer Web-Ressource dient. Weiterhin gibt es das Element *subjectIdenticatorRef* zur Beschreibung eines Objektes, welches nur über die URI-Referenz identifiziert wird.

  Um eine verbesserte direkte Kommunizierfähigkeit von RDF und Topic Maps wird sich bemüht. Jüngst erschien hierzu das W3C-Arbeitspapier [W3C RDF-TM 05], welches verschiedene Vorschläge untersucht. Hieran ist auch Steve Pepper als Mitentwickler der Topic Maps beteiligt.

- Fehlende Definition der *Wohlgeformtheit* bzw. *Gültigkeit* für RDF – in [W3C Semantic 04] wird die Interpretation von RDF-Grafen behandelt. Leider wird dort und auch sonst

[37] Karsten Tolle: Re: Clarifications needed for the Collection construct (with CR). Mail vom 26. Februar 2003, online unter: http://lists.w3.org/Archives/Public/www-rdf-comments/2003JanMar/0422.html

[38] Für einen Vergleich von Topic Maps und RDF sei auf [Gar02] verwiesen.

nirgends definiert, wann ein RDF-Graf *wohl geformt* bzw. *gültig* ist. Solche Begriffe, wie wir sie aus XML kennen, gibt es für RDF-Grafen nicht. Dies führt zu vielen Fällen, in denen die Anwendungen in die Pflicht genommen werden die Überprüfungen so durchzuführen, dass es nicht zu Problemen kommt. Wie dies bereits für Listen, Container und den Eigenschaften *rdfs:range* und *rdfs:domain* erläutert wurde.

Das grundsätzliche Vorgehen alles zu erlauben, um keine Fehler zu erhalten, halte ich für sehr fragwürdig, da es keine Überprüfungen und ein Abfangen von Fehlern zulässt. Wenn wie hier versucht wird eine Grundlage zu entwickeln, mit der Bedeutungen von Maschinen verstanden werden sollen, so muss es opportun sein, einige grundlegende Regeln einzuführen. Man muss sich nur einmal vorstellen, dass zwei Personen zwar das gleiche Vokabular, jedoch eine andere Grammatik verwenden. Dies führt unweigerlich zu Kommunikationsfehlern. Durch Regeln würde die Kompatibilität der Daten erhöht und die Entwicklung von Anwendungen erleichtert werden.

Auch spreche ich hier aus Erfahrung durch die Entwicklung von VRP, wenn ich sage, dass die Datenqualität ohne Regeln und Einschränkungen deutlich abnimmt. Fehler bei der Erstellung von RDF-Daten werden immer wieder vorkommen. Wenn man hier über wenige Regeln viele dieser Fehler abfangen kann, sollte man dies tun. In [Kar+03] haben wir daher ein etwas abweichendes Datenmodell für RDF erstellt, welches strenger ist. Neben dem Verbot von Zyklen in Klassen- und Eigenschaftshierarchien werden hierbei die Eigenschaften *rdfs:domain* und *rdfs:range* als Einschränkungen verstanden. Zusätzlich wird gefordert, dass für jede Eigenschaft die eindeutige Definition des Definitions- und Wertebereiches innerhalb des gleichen Namensraums erfolgen muss.

2.4 Ein Blick in die Zukunft

Trotz vieler Vorteile von RDF fiel es immer schwer bei Projekten insbesondere mit Firmen, diese für RDF zu gewinnen. In den Fällen, in denen es gelang, wie dem *SemFunds*[39] Projekt, war es aussichtslos gegen den sich entwickelnden *FundsXML*[40] Standard durchzusetzen, der über eine entsprechende Lobby verfügte. Auch spielte das *Bootstrapping Problem* ebenfalls eine Rolle. Hierunter versteht man die fehlende kritische Masse an RDF-Daten, sodass es schwer fällt, den wirklichen Nutzen des Semantischen Webs demonstrieren zu können. Dies führt dazu, dass Unternehmen ihre Lösungen nur lokal angehen und vorzugsweise andere Standards verwenden, wie XML-Branchenstandards, für die eine große Auswahl an professionellen XML-Anwendungen zur Verfügung steht. Da dadurch die Nachfrage nach Anwendungen für RDF nicht unterstützt wird und auch keine zusätzlichen RDF-Daten erstellt werden, entsteht ein Zyklus, der nur schwer zu durchbrechen ist.

Durch die hohe Varianz in den RDF-Spezifikationen der vergangenen Jahre wurde dieses verschärft. Da die Standards seit nun über einem Jahr nicht verändert wurden, steigt die Hoffnung, dass hier ein wenig Stabilität erreicht wurde. Wie jedoch im Abschnitt 2.3 gezeigt, gibt es immer noch Probleme und offene Punkte. Einige können vielleicht durch klarere Strukturen in höheren Ebenen behoben werden – wie Ontologie- oder Logik-Ebene (siehe Abbildung 3). Hierdurch steigt jedoch die Gesamtkomplexität weiter an und somit auch die Hemmschwelle in das Semantische Web einzusteigen.

Da die Datenmenge im Internet stetig wächst, gehe ich davon aus, dass das Semantische Web eines Tages auch Realität werden wird. Ob dies dann auf RDF, OWL etc. aufbauen wird, oder sich ein kleinerer weniger komplexer Standard als Teilmenge von diesen (vergleichbar mit SGML und XML) durchsetzen wird, wird die Zukunft zeigen.

[39] SemFunds Projekt, online unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/FondsSchema/index.html

[40] FundsXML, online unter: www.funds-xml.org

Ich persönlich denke, dass die Grundideen von RDF richtig sind, jedoch klarere Vorgaben, wie wir sie in [Kar+03] geben für einen Erfolg nötig sind. Man darf den Web-Benutzern zwar nicht die Freiheit nehmen alles ausdrücken zu können, aber einige Regeln erleichtern auch diesen den Umgang mit RDF.

Die breite Nutzung von RDF im privaten Sektor halte ich dabei mindestens für die nächsten fünf bis zehn Jahren für unrealistisch. Die Entwicklung muss hier von Unternehmen her kommen, die meiner Meinung nach, für die Bildung interner wissensbasierter Systeme einen langfristig positiven Effekt aus einer Umstellung auf RDF erzielen können. Dies vorwiegend durch die bessere Wartbarkeit und erhöhte Flexibilität im Vergleich zu XML. Nachdem sich RDF in den Intranets der Unternehmen etabliert hat, wird es für diese naheliegend sein, unkritische Bereiche frei zugänglich zu machen. Wodurch schließlich das Bootstrapping Problem aufgehoben werden kann. Hiermit rechne ich jedoch frühestens in fünf Jahren, wobei das Fehlen von genügend Fachkräften im Bereich des Semantischen Webs dies deutlich verlangsamen kann.

Die Lukrativität des Semantischen Webs zeigt die steigende Anzahl von Firmen wie Ontoprise[41], Intellidimension [42] und andere, aber auch die Höhe der Fördergelder, die in diesem Bereich fließen. Das Bundesministerium für Bildung und Forschung (BMBF) fördert Semantische Web bis 2006 mit 13,7 Millionen Euro[43].

Wenn sich das Semantische Web und RDF durchsetzen, spielt die Glaubwürdigkeit und Aussagekraft der Daten eine Entscheidende Rolle. Hierfür sind Herkunftsinformatinen oder noch weiter gesehen Kontextinformationen entscheidend, auf die im folgenden Kapitel eingegangen wird.

[41] Ontoprise GmbH, online unter: http://www.ontoprise.de/home

[42] Intellidimension, online unter: http://www.intellidimension.com/

[43] Pressemeldung des BMBF vom 28.06.2004, online unter: http://www.bmbf.de/press/1190.php

3 Kontext im Semantischen Web

Ein wesentliches Ziel im Aufbau und Betrieb des *Semantischen Webs* ist, dass Maschinen die Daten nicht nur syntaktisch, sondern auch semantisch verstehen. Die Bedeutung, also die Semantik, wird jedoch vom Kontext beeinflusst. Es ist daher wichtig, den Kontext zu berücksichtigen und entsprechend handhaben zu können. In diesem Kapitel wird darauf eingegangen, was unter Kontext zu verstehen ist. Dabei wird in Abschnitt 3.1 die Betrachtung nicht nur auf das Semantische Web begrenzt, sondern auch einige Brücken zu anderen Bereichen der Informatik aber auch der Philosophie geschlagen. Hierdurch soll die Komplexität verdeutlicht werden, die mit dem Umgang von Kontext einhergeht. Anschließend wird in Abschnitt 3.2 der Fokus wieder auf das Semantische Web gelegt. Es werden die wichtigsten zurzeit vorkommenden Anwendungsfälle und Darstellungsarten aufgezeigt.

Abschließend werden die zentralen Aussagen dieses Kapitels im Abschnitt 3.3 noch einmal zusammengefasst.

3.1 Kontext Verständnis

Der hohe Stellenwert des Kontextes wird klar, wenn wir uns die Definitionen von Daten, Informationen und Wissen ansehen (frei übersetzt aus [Bro99]):

> ***Daten*** *:= Fakten, Bilder, Nummern, die ohne Kontext präsentiert werden.*
>
> ***Informationen*** *:= organisierte Daten, die in ihrem Kontext repräsentiert werden.*
>
> ***Wissen*** *:= Informationen zusammen mit dem Verständnis, wie diese zu nutzen sind.*

Aus diesen Definitionen folgt sofort, dass wir ohne Kontext kein Wissen, ja noch nicht einmal Informationen haben, sondern nur Daten. Dies liegt an dem Einfluss, den der Kontext auf die Bedeutung hat. Bereits Ogden und Richards [OR23] diskutieren in *The Meaning of Beauty*, welche unterschiedlichen Bedeutungen hinter dem Wort *Beauty* stecken können. Um daher Informationen darzustellen oder zu speichern, spielt der Kontext eine entscheidende Rolle.

Der Begriff *Kontext* ist an sich jedem bekannt und jeder kann sich hierunter etwas vorstellen. Diese Vorstellungen sind durchaus ähnlich, jedoch gibt es auch viele Unterschiede, was durch eine Vielzahl existierender Definitionen für Kontext verdeutlicht wird. Oft werden hierbei Definitionen aufgestellt, die Kontext für spezielle Bereiche definieren. Beginnen wir jedoch mit einer allgemeinen Definition, wie sie z. B. in dem Online-Lexikon Wikipedia[44] zu finden ist:

> *„Als* **Kontext** *wird ein Zusammenhang oder Umfeld beispielsweise eines Wortes oder einer Handlung bezeichnet. Im Falle eines Wortes bezieht sich der Kontext auf das sprachliche Umfeld wie den Text.*
>
> *Kontext ist ein Fremdwort aus dem Lateinischen (contexo = zusammenweben, zusammensetzen; contextus = verflochten, fortlaufend). Die Vorsilbe Kon heißt auf deutsch "zusammen", das Wort text heißt so etwas wie Textur, auf deutsch übersetzt "das Gewebe".“*

In dieser Definition ist von *Zusammenhang oder Umfeld* die Rede. Dies ist sehr vage und gibt keine klaren Grenzen vor, wodurch das Hauptproblem bezüglich Kontext bereits deutlich wird:

> *Wo sind die Grenzen des Kontextes?*

Das Problem wird klarer, wenn wir einen spezielleren Bereich untersuchen. Betrachtet man den Kontext eines Ereignisses, so wird dieser allgemein als kausaler Zusammenhang (Ursache-Wirkungs-Prinzip) angesehen. Als Beispiel könnte man die globale Erwärmung als Ursache für

[44] Definition für Kontext bei Wikipedia:, online unter: http://de.wikipedia.org/wiki/Kontext; Stand 25.11.2004

den starken Gletscherrückgang nennen. Geht man jedoch extrem ins Detail, gibt es auch den Ansatz, alles in Verbindung zueinander zu setzen, nach dem Motto: „Wenn in China ein Sack Reis umfällt, ...“. Dies ist sicherlich übertrieben. Trotzdem ist festzuhalten, dass es viele Ereignisse gibt, bei denen nicht klar ist, was alles zum kausalen Zusammenhang gehört und was nicht. Z. B. sind sich die Wissenschaftler nicht einig, welche Faktoren im Einzelnen zur globalen Erwärmung führen und damit für den Gletscherrückgang verantwortlich sind. B. Russel kommt in [Rus48] zu dem Ergebnis, dass es nicht möglich ist zu entscheiden, ob wirklich alle relevanten Zusammenhänge berücksichtigt wurden. Er beruft sich dabei auf Heisenbergs Unschärferelation.

Bei einer verbalen oder schriftlichen Kommunikation ist festzustellen, dass durch den Kontext Aussagen völlig unterschiedliche sogar gegensätzliche Bedeutungen haben können. Nehmen wir die Aussage: „Dirk ist der Größte!“ als Beispiel. Ohne Kontext wissen wir weder was hierbei das Vergleichskriterium ist, noch mit wem verglichen wird. Deutlich ausdrucksstärker wird der Satz, wenn Zusatzinformationen hinzukommen:

- „Dirk ist der Größte *in seiner Klasse*.“ – Bei dieser Aussage wird der Vergleichsrahmen (die Menge aller, die in der gleichen Klasse sind wie Dirk) mit angegeben. Unser sprachlicher Kontext würde uns vermutlich die Körpergröße als Vergleichskriterium aufdrängen, falls kein anderer Kontext angegeben wird.
- „Dirk ist der größte *Basketballspieler für mich*.“ – Hierbei wird eine subjektive Einschätzung als Stellenwert für eine Person im Bezug auf „*Basketballspielen*“ ausgedrückt.

In einem persönlichen Gespräch werden die Zusatzinformationen oft nicht im Satz mit angeführt, sondern werden durch den äußeren Rahmen oder aus früheren Gesprächen etc. ersichtlich. Dort könnte bereits der Name *Nowitzki* gefallen sein, wodurch klarer wird, wer mit dem Vornamen *Dirk* gemeint ist.

Zu beachten ist bei einer verbalen Kommunikation auch die Art und Weise wie eine Aussage ausgesprochen wird, sprich Tonfall, Mimik, Gestiken, Pausen, usw. Oft kann man nur hierüber feststellen, ob eine Aussage inhaltlich auch so gemeint ist, oder ob es sich zum Beispiel um eine ironische Bemerkung handelt, die genau das Gegenteil ausdrücken möchte. Ebenso spielen äußere Umstände eine Rolle. Bei dem Ausspruch: „Das Wetter ist ja klasse!“ hilft neben dem Tonfall auch ein Blick aus dem Fenster, um die Bedeutung zu evaluieren (ironisch gemeint oder nicht). Der Kontext ist also wichtig, um die Semantik der Aussagen richtig interpretieren zu können. Dies umfasst auch entscheiden zu können, ob Aussagen wahr sind oder nicht. Der Ausspruch: „Alle Vögel können fliegen!“ mag für einen gegebenen Rahmen gelten, allgemein gilt es nicht.

Für den Bereich der Linguistik (Sprachwissenschaft) finden wir in [Wei66]:

> *„Der Satz, mitsamt dem weiteren Kontext und der umgebenden Situation, grenzt die (weitgespannte, vage, soziale, abstrakte) Bedeutung auf die (engumgrenzte, präzise, individuelle, konkrete) Meinung ein. Wenn man ein isoliertes Wort hört, kann der Geist im ganzen Umkreis der Bedeutung schweifen. Hört man das Wort im Text, geht das nicht mehr. Der Kontext stellt fest. Er stellt nämlich die Bedeutung fest.“*

Zu bemerken bleibt jedoch, dass die Wahrnehmung einer Situation und auch das Verständnis einzelner Wörter vom Individuum abhängen. Es ist abhängig von der sensorischen Wahrnehmung, welcher Gemütszustand vorliegt, welcher kulturelle Hintergrund gegeben ist, etc. Trotzdem können wir jedoch im Allgemeinen von einer Grundübereinstimmung ausgehen. Russel schreibt in [Rus48]:

> *„A given form of words will usually be interpreted by competent hearers in such a way as to be true for all of them or false for all of them, but in spite of this it will not have the same meaning for all of them.“*

Dies stellt die eigentliche Grundlage der Kommunikation im Allgemeinen und der Informatik im Besonderen dar. Ohne diese wäre kein Informationsaustausch möglich. In der Evolution des Menschen hat sich die Nutzung des Kontextes in der Kommunikation bewährt. Schnittstellen zwischen Mensch und Maschine sollten dies entsprechend berücksichtigen und darauf eingehen. Durch die voranschreitende Entwicklung in der Informatik gewinnt daher die Berücksichtigung des Kontextes eine immer höhere Bedeutung. Dies wird nicht zuletzt durch die steigende Personalisierung einzelner Anwendungen in den letzten Jahren deutlich. Hierbei kann die Personalisierung an sich als Einbeziehung des Nutzers in den Kontext betrachtet werden.

In neueren Bereichen wie dem *Ubiquitous Computing* spielen Kontextinformationen daher eine zentrale Rolle. Neben dem klassischen Eingabe-Ausgabe-Prinzip werden hier zusätzlich zur Personalisierung vermehrt sensorische Daten (Ort, Temperatur, Lichtverhältnisse etc.) berücksichtigt, um der jeweiligen Situation angepasst reagieren zu können. Zur einfacheren Handhabung wird oft eine Kategorisierung von Kontext durchgeführt. So werden Daten bezüglich des Benutzers, der Umgebung, der Zeit oder ähnlichen Kriterien zusammengefasst und separat behandelt. Marie-Luise Moschgath geht in ihrer Dissertation [Mos02] (insbesondere im Kapitel 3) hierauf weiter ein. Dort sind auch verschiedene Definitionen des Kontextes im Bezug auf Ubiquitous Computing genannt, wodurch die Schwierigkeiten beim Umgang mit Kontext auch in diesem Bereich verdeutlicht werden.

Einer der mittlerweile klassischen Bereiche, in denen der Kontextbegriff untersucht wurde, ist der Bereich der wissensbasierten Systeme (*KnowledgeBased Systems*). Auch hier wurde jedoch weder eine einheitlich akzeptierte Definition für Kontext, noch eine einheitliche Form der Darstellung gefunden und auch nicht gelöst, wie Kontext die Wissensverarbeitung beeinflussen sollte. Bob Jansen fasst die wichtigsten Arbeiten, Interpretations- und Handhabungsansätze über Kontext bezüglich wissensbasierter Systeme in [Jan93] zusammen. Die Schwierigkeit der Abgrenzung des Kontextes sorgt dabei für eine hohe Komplexität und ist ein Grund, weshalb es keine einheitlich akzeptierte Definition existiert. Russel [Rus48] beschreibt Kontext als *endloses ungebundenes Ding*. Selbst die Reduktion des Kontextbegriffes auf einzelne Bereiche, wie Linguistik, Kommunikation oder Ursache-Wirkung ist nicht ohne Kritik. Shanon [Sha90] wendet ein, dass dies zwar einzelne Probleme löst, jedoch auf der anderen Seite wiederum neue Probleme schafft.

Es bleibt festzuhalten, dass durch die fehlenden klaren Grenzen des Kontextes seine Handhabung schwierig bleibt. Viele Autoren verzichten ganz auf eine Definition von Kontext. Die Praxis zeigt, dass dies durchaus legitim ist, solange gute Ergebnisse erreicht werden.

3.2 Kontext für das Semantische Web

In den RDF-Spezifikationen wird unter [W3C Semantic 04] nur auf Ableitungs- und Interpretationsregeln eingegangen. Eine direkte Behandlung von Kontext findet nicht statt und es existieren keine Vorgaben bezüglich seiner Darstellung. In [W3C Tracking] wurde dies bereits im Jahr 2000 als zu klärender Punkt auf Anregung von Graham Klyne aufgenommen. Bisher wurde dieser vom W3C jedoch noch nicht behandelt. Dies hat dazu geführt, dass für die verschiedenen Anwendungsfälle für Kontext im Semantischen Web (siehe 3.2.1) auch verschiedene Darstellungsarten (siehe 3.2.3) existieren. Besonders kritisch ist die Glaubwürdigkeit der Daten, weshalb in 3.2.2 eine Aufteilung des Kontextes nach seiner Herkunft vorgenommen wird.

3.2.1 Anwendungsfälle für den Gebrauch von Kontext in RDF

Die unterschiedlichen Auffassungen und die Komplexität von Kontext erlauben keine vollständige Auflistung aller Anwendungsfälle für die Nutzung von Kontext in RDF. Im Folgenden werden die Anwendungsfälle aufgeführt, die häufig zu finden sind.

- **Darstellung der Kontextgrenzen** – als Möglichkeit des Autors die Kontextgrenzen aufzeigen und damit die beabsichtigte Bedeutung feststellen zu können.
- **Behandlung von Ausnahmen** – wenn Aussagen über Gruppen von Objekten getroffen werden, so gibt es immer wieder wenige Ausnahmen, die von dieser Aussage abweichen. Das klassische Beispiel sind die Pinguine, die als eine von mehreren wenigen Ausnahmen aus der Gruppe der Vögel nicht fliegen können.
- **Darstellung verschiedener Ausprägungen eines Objektes** – Zustände zu bestimmten Zeiten, z. B. kann eine Person unterschiedliche Rollen annehmen, wie Dr. Jekyll und Mr. Hyde oder die Position und Fracht eines Schiffes.
- **Darstellung von Beziehungen zwischen Komponenten** – Ermöglichung eines modularen Aufbaus, z. B. zur Darstellung von Computernetzwerken oder Softwarepaketen.
- **Darstellung verschiedener Sichtweisen** – für jemanden aus dem Vertrieb sind Informationen wichtig, die für die Produktion unwichtig sind.
- **Darstellung von *Herkunftsinformationen* (*provenance information*)** – neben der *Quellinformation* (*source information*) umfasst die Herkunftsinformation auch Informationen über die Veränderungen eines Dokumentes. Wenn z. B. eine Datei aus dem Web mit der URL http://www.ex.de/ex.rdf in eine Datensenke gespeichert wird, entspricht die URL der Quellinformation. Zur Herkunftsinformation würde zusätzlich gehören, wer und wann diese Datei erstellt bzw. verändert hat. Im Allgemeinen können wir nicht davon ausgehen, alle Herkunftsinformationen eines Dokumentes zu kennen. Herkunftsinformationen gelten für Mengen von Aussagen, wobei jede Aussage der Menge die Herkunftsinformation *erbt*.

  Durch die Bereitstellung der Quellinformation kann der Nutzer eines Systems sich an der Informationsquelle weitere Informationen beschaffen. Auch kann hierüber die Aktualität der Informationen überprüft werden. Dies unterstützt die Glaubwürdigkeit der Daten, die damit durch den Nutzer selbst, oder aber durch externe Ranking Systeme erhöht werden kann. Durch entsprechende Speicherung könnten auch verschiedene Versionen eines Dokumentes nebeneinander abgelegt werden und so dessen Entwicklung dokumentieren.

3.2.2 Aufteilung des Kontextes nach seiner Herkunft

Wie in anderen Bereichen ist auch im Semantischen Web keine allgemein gültige Definition für Kontext gegeben. Sie hängt auch vom Anwendungsfall und der Abstraktionsebene des Betrachters ab. Bezüglich der Abstraktionsebenen – diese sind vergleichbar mit dem Schichtenmodell in Abbildung 3 auf Seite 10 – teilt Elena Plaslaru Bontas Kontext in *Kontext für Web-Dokumente, für Ontologien bzw. für Web Services* auf. Für die Web-Dokumente sieht sie vor [Bon04]:

> *„Meta data and meta data of linked documents, ontologies describing the topic of the document, FOAF ontology offering additional facts about the authors of the document (e.g. for trust issues).“*

Was sie hier für die Web-Dokumente nicht berücksichtigt, sind die Daten der Dokumente an sich, die als Grafstruktur auch über mehrere Dokumente dargestellt werden können. Andere sehen gerade diese Grafstruktur der Daten im Vordergrund. Der Kontext einer Aussage wird daher häufig als der *umgebene Graf* bezeichnet (z. B. in [BC04]). Diese Sichtweise ist auf den Grafen begrenzte. Metadaten, die man über einen Grafen besitzt, müssen dabei in den Grafen integriert werden, um zum Kontext zu gehören. Dies ist in RDF zwar möglich, führt jedoch dazu, dass diese Metadaten nicht mehr von den *vorherigen Daten* unterschieden werden können. Eine fehlende Unterscheidungsmöglichkeit führt jedoch zum Verlust der

Glaubwürdigkeit der Daten. Hierauf wird später noch weiter eingegangen. Ich möchte daher hier eine Aufteilung der Kontextdaten wie folgt vornehmen:

interner Kontext *:= Der RDF-Graf, der eine RDF-Aussage oder eine Menge von RDF-Aussagen umgibt.*

externer Kontext *:= Zusätzliche Metadaten, die zu dem RDF-Grafen gemacht werden können, dies umfasst Herkunftsinformationen, äußere Gegebenheiten wie Datum, Zeit, etc.*

Die Behandlung des internen Kontextes ist abhängig von der gewählten Darstellungsart (siehe 3.2.3). Unabhängig hiervon existieren in [W3C Semantic 04] Ableitungs- und Interpretationsregeln für RDF-Grafen.

Zum externen Kontext zählen Informationen, die zusätzlich zu dem RDF-Grafen festgehalten werden können. Wenn wir an eine Datensenke denken, gehört zum externen Kontext:

- Woher stammen die Daten (Quellinformationen)?
- Wer hat sie in die Senke gespeichert?
- Wann wurde dies getan?
- usw.

Neben derartigen Informationen können aber auch, Informationen über äußere Gegebenheiten – entsprechend der Sensordaten beim Ubiquitous Computing – von Interesse sein. Insbesondere für Systeme, die ein menschliches Verhalten nachbilden wollen, ist dies hilfreich, worauf in [Tol04] weiter eingegangen wird. Im Allgemeinen gilt der externe Kontext für den ganzen RDF-Grafen. Es werden also Aussagen über mehrere RDF-Aussagen getroffen.

Zu beachten ist, dass Informationen, die hier explizit zum externen Kontext gezählt werden auch innerhalb von RDF gespeichert werden können. Diese können somit auch im internen Kontext zu finden sein. Eine der häufig gestellten Fragen ist, warum man daher externe Kontextinformationen gesondert behandeln soll. Durch die Trennung bietet sich jedoch die Möglichkeit Metadaten bezogen auf den Grafen, von den eigentlichen Informationen aus dem Graf zu unterscheiden und so die Glaubwürdigkeit der Daten zu erhöhen. Hierbei sei bemerkt, dass die Glaubwürdigkeit der Daten im Semantischen Web mit zu den zentralen Problemen gehört. So sieht Monika Henzinger, Forschungsdirektorin von Google, Suchmaschinen-Spamming als größte Gefahr für das Semantische Web [Sch05]. Hierrunter ist die vorwiegend bewusste Einspeisung von falschen Daten zu verstehen.

Beispiel: Angenommen ein Produkt *P* wird von verschiedenen Anbietern angeboten. Sei weiterhin der in Abbildung 8 abgebildete RDF-Graf gespeichert. Der Graf beinhaltet die Aussagen, dass ein Anbieter *xyz P* für *100 $* anbietet und ein weiterer Anbieter *abc* hierfür nur *80 $* verlangt. Die erstgenannte Aussage mag stimmen, kann jedoch ebenso gut vom Anbieter *abc* kommen, um vorzutäuschen er sei günstiger – wobei in Wirklichkeit der Anbieter *xyz P* für nur *70 $* anbietet. Man kann den Daten nicht ohne weiteres trauen! Mit dem zusätzlichen Wissen, woher die Informationen stammen und wann auf sie zugegriffen wurde, kann die Glaubwürdigkeit erhöht werden. Auch wenn dies nicht automatisch heißt, dass der obere Teilgraf falsch und der untere wahr sein muss.

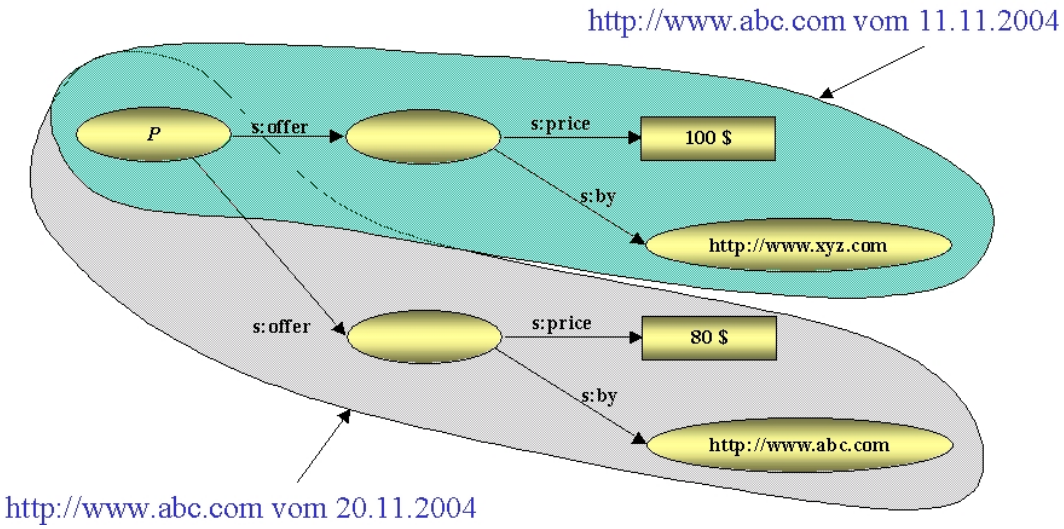


Abbildung 8 Beispiel eines RDF-Grafen, der zwei Angebote für das Produkt *P* darstellt. Zusätzlich sind die Quell- und Zugriffszeitinformationen abgebildet.

Eine Eigenschaft von RDF ist, dass jeder alles ausdrücken kann. Werden die externen Kontext Informationen innerhalb des Modells gehalten, so können diese nicht von Aussagen Dritter unterschieden werden. Damit wäre es möglich falsche Tatsachen vorzutäuschen. Um dies zu verhindern, müsste man die eingehenden Informationen auf die Verwendung intern genutzter Verfahren durchsuchen und gegebenenfalls entfernen. Dies ist mit hohem Aufwand verbunden und würde durch die Entfernung von Daten die Quellen verfälschen.

Neben der Glaubwürdigkeit können weiterhin die externen Kontextinformationen für spezielle Aufgaben dienlich sein. So kann die Quellinformation für das Löschen oder Aktualisieren von einzelnen RDF-Dokumenten in einer Datensenke genutzt werden. Ein Herausfiltern dieser Information, falls diese in den Grafen integriert wurde, würde ebenfalls zusätzliche Arbeit bedeuten. Auch die Performance für Anfragen, die nicht den externen Kontext benötigen, kann bei separater Speicherung, durch die Reduktion der zugrunde liegenden Datenmenge, gesteigert werden.

Es ist eine weitere Strukturierung des externen Kontextes denkbar. Werden Metadaten eines Systems in ein anderes übertragen, wäre eine hierarchische Strukturierung des externen Kontextes möglich. Dies soll hier jedoch nicht weiter betrachtet werden.

3.2.3 Existierende Ansätze Kontext zu modellieren

Es gibt verschiedene Ansätze zur Handhabung der genannten Anwendungsfälle. Einige kommen mit zusätzlichen RDF-Vokabularien aus, während andere eine zusätzliche Komponente zu den RDF-Aussagen und damit zum RDF-Modell hinzufügen. Die existierenden Ansätze werden nun vorgestellt und anschließend in 3.2.4 vergleichend gegenübergestellt.

Vergegenständlichte Aussagen und Container – Graham Klyne beschreibt in [Kly00] seinen Ansatz, der auf den Arbeiten von McCarthy [McC93] und R.V. Guha [Guh91] beruht. Hierfür erstellt er ein RDF-Vokabular für den Umgang mit Kontexten. Er nutzt dabei *vergegenständlichte Aussagen*, die in speziellen Containern gruppiert werden können. Die Container bilden dabei die Grenzen des Kontextes. Neben den einzelnen Aussagen können die Container auch Bedingungen enthalten, die erfüllt sein müssen, damit die Aussagen des Kontextes gültig sind. Über eine spezielle Eigenschaft (*applyToAll*) können Aussagen über alle im Container enthaltene Aussagen getroffen werden[45]. Wichtig ist, dass diese Kontextcontainer

[45] Die Eigenschaft applyToAll ist vergleichbar mit dem aus dem RDF-Kernvokabular entfernten Attribut *rdf:aboutEach*.

auch untereinander in Verbindung stehen können. Hierdurch können größere Kontexte aus Kleineren aufgebaut werden. Auch das Überschreiben von Aussagen ist möglich, wodurch sich Ausnahmen ausdrücken lassen. So kann in einem (*default*) Kontext die Aussage enthalten sein, dass alle Vögel fliegen können. Dies kann in einem weiteren Kontext überschrieben werden, der die Aussage enthält: „Pinguine können nicht fliegen."

Die Aussagen werden in diesem Ansatz mittels vergegenständlichter Aussagen codiert. Graham Klyne nutzt in seinen Beispielen Quadrupel, um die vergegenständlichten Aussagen übersichtlich darstellen zu können. Es ist jedoch wichtig festzuhalten, dass dieser Ansatz innerhalb von RDF bleibt. Er spezifiziert lediglich ein spezielles RDF-Vokabular. In seiner späteren Arbeit [Kly02] geht Graham Klyne tiefer auf modell-theoretische Aspekte ein.

Der Nachteil dieses Ansatzes ist der sehr umständliche Umgang mit vergegenständlichten Aussagen, sowohl für das Erstellen, als auch für Anfragen. Durch die verwendete Schreibweise als Quadrupel umgeht Graham Klyne dies bereits, was direkt zum nächsten Ansatz führt.

Quadrupel (Quads) – hierbei werden die RDF-Aussagen um eine vierte Komponente erweitert. Die zusätzliche Komponente *C* der Quadrupel: *<C, S, P, O>* wird zur Kontextbildung, genauer zur Gruppierung der Aussagen, genutzt. Die weiteren drei Komponenten *S, P* und *O* bilden Subjekt, Prädikat und Objekt der RDF-Aussage. Die Kontextkomponente *C* kann dabei entweder *null* oder eine Ressource sein. Falls *C* eine Ressource ist, kann diese auch als Subjekt oder Objekt in der RDF-Aussage auftreten. Hierdurch können Aussagen über die Gruppe der Aussagen, die es repräsentiert, dargestellt werden.

Das folgende Beispiel[46] trifft die Aussagen: *„Julia mag Ballett."* und *„Julia ist die Tochter von Astrid."*. Die unteren beiden Quadrupel geben an, dass diese Aussagen von *„Karsten"* am *„20.10.2004"*gespeichert wurden.

```
[_:cxt _:Julia ex:likes ex:Ballet]
[_:cxt _:Julia ex:daughterOf _:Astrid]
[null _:cxt ex:storedBy "Karsten"]
[null _:cxt ex:storedDate "20.10.2004"]
```

Robert MacGregor und In-Young Ko gehen in [MK03] auf Quads ein. Sie kommen zu der Behauptung, dass keine weiteren Komponenten nötig sind, ohne dies zu beweisen. In ihrem Projekt, welches sie beschreiben, verwenden sie jedoch keine Quads. Hauptgrund gegen deren Nutzung ist die fehlende Unterstützung von Quadrupel durch existierende RDF-Werkzeuge. Anstelle von Quads entwickeln sie den *Object-Centered Context*-Ansatz, der weiter unten beschrieben wird und welcher ohne Erweiterung des RDF-Modells auskommt.

Named Graphs – dies stellt eine Weiterentwicklung der Quadrupel dar. Eine ausführliche Beschreibung ist in [CBHS04] zu finden. Auch hier werden Gruppen von Aussagen gebildet, die *Grafen* genannt werden. Diese können mittels eines *Namens* benannt werden. Als Name kann eine URI oder eine anonyme Ressource dienen. Damit kann der Name ebenfalls als Ressource betrachtet und innerhalb der RDF-Aussagen verwendet werden. Unter Verwendung der TriG-Syntax [Biz05] kann das obige Beispiel in folgenden *Named Graph* überführt werden:

```
_:G1 (_:Julia ex:likes ex:Ballet.
      _:Julia ex:daughterOf _:Astrid.)
_:G2 (_:G1 ex:storedBy "Karsten".
      _:G1 ex:storedDate "20.10.2004".)
```

In [CBHS04] wird auch beschrieben, wie die Named Graphs bezüglich der *Interpretation I* aus [W3C Semantic 04] zu behandeln sind. Im Unterschied zu den Quads sind *null-Werte* nicht

[46] Die Darstellungsart für Quads wurde für dieses Beispiel aus [MK03] übernommen.

nötig. Trotzdem findet keine Festlegung auf semantischer Ebene statt. Zu bemerken ist, dass ein *Named Graph* mit nur einer Aussage innerhalb des Grafen als vergegenständlichte Aussage interpretiert werden kann. Weiterhin kann jede RDF/XML-Datei in einen Named Graph überführt werden, indem die *Base URI* oder, falls nicht vorhanden, die URL unter der die Datei zu finden ist, als Name genommen wird. In [CBHS04] wird dies als Abwärtskompatibilität von Named Graphs zu RDF bezeichnet. Die Umkehrung gilt nicht, da Named Graphs auch Literale als Subjekt und anonyme Ressourcen als Prädikate zulassen. Neben der TriG-Syntax gibt es noch TriX [CS04], um Named Graphs zu beschreiben.

Chris Bizer und Jeremy Carroll geben in [BC04] Beispiele an, wie Named Graphs für verschiedene Anwendungsfälle verwendet werden können.

Object-Centered Context – MacGregor und Ko beschreiben neben den Quads in [MK03] noch einen weiteren Ansatz, den sie *Object-Centered Context* nennen. Hierbei können über eine spezielle Eigenschaft (sie nennen diese *ex:theRealThing*) verschiedene Sichten (*snapshots*) eines Objektes erzeugt werden. Verwendung findet diese Methode z. B. zur Darstellung von Schiffspositionen mit Breiten- und Längengrad zu verschiedenen Zeiten. Die verschiedenen Sichten können dabei untereinander in Beziehung stehen. Am Beispiel der Schiffe zum Darstellen, welche Fracht zu einer bestimmten Zeit geladen ist.

Dieser Ansatz kommt ohne Erweiterung des RDF-Modells aus und verzichtet dabei auf eine massive Verwendung von vergegenständlichten Aussagen. Die Abbildung des Kontextes auf die Statements erfolgt indirekt über die Sichten. Für Autoren oder Quellinformationen würde dies im Vergleich zu Quads eine zusätzliche zwischengeschaltete Ressource bedeuten. MacGregor und Ko halten in [MK03] auch die Kombination mit Quads für sinnvoll.

Triple Plus Context Node – auch bei diesem Ansatz werden Quadrupel der Form: <*C, S, P, O*> verwendet. Den RDF-Aussagen (*S, P, O*) wird der Kontextknoten *C* hinzugefügt, der genutzt wird, um die Aussagen in Gruppen zusammenzufassen. Der Unterschied zu den Ansätzen Quads und Named Graphs, besteht in der Nutzung des Kontextknotens. Es ist bei diesem Ansatz nicht vorgesehen innerhalb einer RDF-Aussage durch Verwendung der Angaben, die unter *C* vorkommen, Aussagen über die Aussagen zu treffen, die mittels dieses Kontextknotens gruppiert wurden. Es findet also eine semantische Trennung des Kontextknotens vom RDF-Modell statt. Durch Ausblenden der Kontextknoten würde damit auf die reinen RDF-Daten reduziert werden. Als Beispiel seien wieder die beiden Aussagen „*Julia mag Ballett.*“ und „*Julia ist die Tochter von Astrid.*“ gegeben, die zum gleichen Kontext „*C1*“ gehören, also:

```
[C1 _:Julia ex:likes ex:Ballet]
[C1 _:Julia ex:daughterOf _:Astrid]
```

Wenn nun diese Aussagen von „Karsten“ am „20.10.2004“ in eine Datensenke gespeichert wurden, so müssen diese Metainformationen unter Verweis auf *C1* getrennt abgespeichert werden. Daher wird durch diesen Ansatz die Ausdrucksfähigkeit *innerhalb* des Modells nicht gesteigert. Eine Möglichkeit zur Nutzung des Kontextknotens *C* ist für Herkunftsinformationen der gespeicherten Daten. Es sind jedoch auch durchaus andere Möglichkeiten vorstellbar.

3.2.4 Vergleich der Ansätze

Welcher Ansatz zur Darstellung verwendet werden soll, hängt vom einzelnen Anwendungsfall ab. Ist nur interner Kontext wichtig, empfiehlt sich sicherlich der Named Graph-Ansatz, da er am weitesten entwickelt scheint. Sollen Momentaufnahmen von Objekten dargestellt werden oder existierende RDF-Werkzeuge benutzt werden, die keine Quadrupel unterstützen, ist der Object-Centered Context-Ansatz zu empfehlen. Ist jedoch eine sichere Trennung von internem und externem Kontext wichtig, kann nur der Triple Plus Context Nodes-Ansatz Verwendung finden. Da dieser jedoch nicht für die Darstellung von internem Kontext geeignet ist, wäre eine

Kombination mit dem Object-Centered Context-Ansatz eine mögliche Lösung. Eine Kombination des Named Graphs-Ansatzes mit dem Triple Plus Context Node-Ansatz wäre auch denkbar, scheint aber nicht ratsam, da hierdurch Quintruple entstehen würden, deren Handhabung noch komplexer erscheint. In der folgenden Tabelle 1 sind die zentralen Eigenschaften noch einmal übersichtlich dargestellt.

Tabelle 1 Vergleich der Ansätze zur Darstellung von Kontext im Semantischen Web bezüglich zentraler Eigenschaften. Positives ist dabei grün hinterlegt.

	Vergegenständlichte Aussagen und Container	Quadruple (Quads)	Named Graphs	Object-Centered Context	Triple Plus Context Node
Trennung interner – externer Kontext	nein	nein	nein	nein	ja
Erweiterung von RDF nötig	nein	ja	ja	nein	ja
einfache Handhabung	nein	(ja)	ja	(ja)	(ja)

In der Tabelle 2 werden die Ansätze bezogen auf ihre Einsetzbarkeit für die genannten Anwendungsfälle verglichen. Bei der Darstellung der Kontextgrenzen ist für den Triple Plus Context Node-Ansatz die Nutzung der Dokumentengrenze als mögliche Kontextgrenze nutzbar, jedoch entspricht dies nicht dem eigentlichen Grundgedanken dieser Anwendung.

Tabelle 2 Vergleich der Ansätze zur Darstellung von Kontext im Semantischen Web bezüglich der Anwendungsfälle.

	Vergegenständlichte Aussagen und Container	Quadruple (Quads)	Named Graphs	Object-Centered Context	Triple Plus Context Node
Darstellung der Kontextgrenzen	+	+	+	+	-
Behandlung von Ausnahmen	+	+	+	o	-
Verschiedene Ausprägungen	o	o	o	++	-
Hierarchischer Kontextaufbau	+	+	+	+	-
Herkunftsinformationen	+	+	+	-	++

3.3 Zusammenfassung

In diesem Kapitel wurde erläutert, warum der Umgang mit Kontext so schwierig ist. Auch wurden die Unterschiede existierender Darstellungsarten für Kontext im Semantischen Web erläutert. Bezogen auf die Glaubwürdigkeit der Daten wurde die Möglichkeit der Trennung der Daten von deren Metadaten als wichtig herausgestellt. Als einzige Darstellungsart bietet hierfür die Triple Plus Context Node-Ansatz eine sichere Lösung. Diese Darstellungsart ist jedoch in ihrer Ausdrucksstärke bezogen auf den internen Kontext begrenzt. Dies kann durch eine Kombination mit einer weiteren Darstellungsart, die das RDF-Modell nicht erweitert, kompensiert werden.

Im folgenden Kapitel 4 wird ein Speichersystem vorgestellt, welches dem Triple Plus Context Node-Ansatz folgt und eine saubere Trennung der externen und internen Daten vornimmt. Dort werden bereits die Vorteile dieses Vorgehens für die Wartbarkeit der Daten deutlich. Die

weiteren Möglichkeiten bezüglich der Interpretation und Verarbeitung werden in Kapitel 5 im Rahmen der Anfragesprache eRQL angesprochen.

4 Speicherung von RDF-Daten plus Kontext

Das Semantische Web stellt eine Erweiterung des Internets dar, wobei die Daten als RDF/XML-Dateien oder als Zusätze zu den existierenden Daten abgelegt werden. Für Anwendungen, die Ergebnisse in kurzer Zeit liefern sollen, wie z. B. bei Informationsportalen, wäre ein direkter Zugriff auf diese verteilten Daten zu langwierig. Auch ist das RDF/XML-Format nicht für einen schnellen Zugriff konzipiert, sondern für den Datenaustausch. Weiterhin ist für manche Anwendungen eine permanente Zugriffsmöglichkeit wichtig. Dies kann in einem verteilten System nicht immer gewährleistet werden. Insgesamt bleibt festzuhalten, dass die Möglichkeit zur Speicherung der RDF-Daten in Datensenken zwingend notwendig ist. Dabei spielt das verwendete Datenmodell, also wie die Daten in einer Datensenke abgebildet werden, eine entscheidende Rolle. Es hat Auswirkungen auf sämtliche Anwendungen, die auf der Datensenke aufbauen.

In diesem Kapitel wird auf die Speicherung der RDF-Daten eingegangen. Dazu werden in Abschnitt 4.1 existierende Datenmodelle vorgestellt. Anschließend wird ein neues relationales Datenmodell beschrieben, welches zwei existierende Ansätze kombiniert. Es wird daher *kombinierte Repräsentation* (*ComRepr*) genannt. Durch die Kombination können einzelne Nachteile der beiden Modelle, bezüglich unterschiedlicher Anfragetypen, ausgeglichen werden.

Zusätzlich werden in der ComRepr die Aussagen nicht als Tripel, sondern als Quadrupel gespeichert, sodass die Speicherung von Kontextinformationen unterstützt wird. Hierbei wird die Darstellung des Kontextes im *Triple Plus Context Node-Ansatz* genutzt, um eine klare Trennung des internen und externen Kontextes zu ermöglichen. Im Abschnitt 4.2 wird der Aufbau der ComRepr genauer beschrieben.

Im Abschnitt 4.3 wird das entwickelte *RDF-Source related Storage System* (*RDF-S3*) vorgestellt, welches die ComRepr implementiert. Dabei wird auf einzelne Aspekte der Implementierung sowie auf die Leistungsfähigkeit und Skalierbarkeit des Speicherns eingegangen. Die Leistungsfähigkeit bezüglich Anfragen von RDF-S3 wird im Kapitel 5 zusammen mit der Anfragesprache *eRQL* genauer behandelt.

4.1 Datenmodelle zur Speicherung von RDF-Daten

Die Datenmodelle werden maßgeblich von der Art der Datensenke und ihrer Funktionalität beeinflusst. So gibt es Ansätze und Implementierungen, die sich auf logische Regeln konzentrieren. Durch die Regeln können neue Aussagen aus den vorhandenen Aussagen abgeleitet werden. Solche Ableitungen (Inferenzen) werden z. B. durch deduktive Datenbanken bzw. Prologsysteme optimal unterstützt [KR03, WLS03, WSW03]. Als kommerzielles Produkt sei hier der OntoBroker von Ontoprise[47] erwähnt, wobei die Aussagen als Fakten und Regeln in F-Logik gespeichert werden. Andere nutzen die objektorientierten Eigenschaften der Daten des Semantischen Webs über objektorientierte Datenbanken [BHS03]. Gleiches gilt für die Nutzung der Datenbank Caché von InterSystems, die in [Kar04] bezüglich der Speicherung von RDF-Daten getestet wurde. Alle diese Ansätze haben ihre Berechtigung. So kann auf den entsprechend von der Datensenke unterstützten Bereichen meist eine Verbesserung der Leistungsfähigkeit erzielt werden.

Am weitesten verbreitet sind jedoch *relationale Datenbankmanagementsysteme (RDBMS)*. Um die in Kapitel 2 beschriebene fehlende Akzeptanz gegenüber dem Semantischen Web zu verringern, ist es erforderlich, Lösungen für RDBMS zu bieten. Insbesondere in der Wirtschaft gibt es triftige Gründe für die Wahl eines RDBMS als Datensenke, wie der Wegfall zusätzlicher Lizenzgebühren, jahrelange Bindungen oder Fachwissen der Mitarbeiter. Eine für die

[47] Die Ontoprise GmbH (http://www.ontoprise.de) 1999 gegründetes Unternehmen mit Sitz in Karlsruhe.

Unternehmen *„exotische“* Datenbank würde die Akzeptanz weiter hemmen. Daher wird der Fokus hier auf relationale bzw. objektrelationale Datenbanksysteme gelegt.

Für die Darstellung im relationalen Modell gibt es zwei Hauptansätze, die *generische Repräsentation* (*GenRepr*) und die *schema spezifische Repräsentation* (*SpecRepr*). Beide werden im Folgenden kurz vorgestellt und anschließend vergliechen. Es wird dabei nur auf die führenden Anwendungen im Java-Bereich, die diese Ansätze geprägt haben, verwiesen. Dies sind *Jena* [McB02] von Hewlett-Packard und die *ICS-FORTH RDFSuite* [Ale+01]. Beiden sind auf der RDF-Startseite des W3C für Java-Entwickler aufgeführt. Zusätzlich sind dort Sesame [BK02] und die Kowari Metastore™ Datenbank zu finden. Für einen breiteren Vergleich sei auf [Mar+02] und [HBEV04] verwiesen.

Die GenRepr wird auch als *tripelorientierte Repräsentation* oder *monolithischer Ansatz* bezeichnet. Vereinfacht gesagt, werden hierbei die einzelnen Aussagen in einer Tabelle mit Subjekt, Prädikat und Objekt Attributen gespeichert. Diese Darstellungsweise wird z. B. von Jena verwendet. Es wurden verschiedene Abwandlungen getestet, um die Leistungsfähigkeit zu steigern [WSKR03]. Einer der größten Vorteile dieser Repräsentation ist die Einfachheit, wodurch sie leicht verstanden, implementiert und erweitert werden kann.

Die SpecRepr wurde in [Ale+01] erstmals von uns vorgestellt und in der ICS-FORTH RDFSuite implementiert. Sie wird auch als *grafbasierte* oder *ontologieorientierte Repräsentation* bezeichnet. Die RDFSuite setzt auf dem objektrelationalen Datenbankmanagementsystem *PostgreSQL*[48] auf. Hierbei werden die objektorientierten Eigenschaften für die Instanzvererbung innerhalb der Klassenhierarchien ausgenutzt. Der Ansatz wird in [Gro03] daher *objektrelationaler Ansatz* genannt. Es werden vorhandene RDF-Schema-Informationen zur Speicherung genutzt. Vereinfacht kann man sich vorstellen, dass für jede Klasse und jede Eigenschaft eine eigene Tabelle angelegt wird. In die *Klassentabellen* werden die Instanzen der jeweiligen Klasse eingetragen. Die *Eigenschaftstabellen* speichern die Verbindungen von den Subjekten zu den Objekten, welche durch die Eigenschaft vorgenommen werden. Neben den Klassen und Eigenschaften werden auch die Container und vergegenständlichte Aussagen gesondert gespeichert. Je nach Anzahl der Klassen bzw. Eigenschaften kann die Anzahl der benötigten Tabellen stark steigen. Die RDFSuite bietet daher die Option, anstelle der Klassentabellen, eine große Instanztabelle zu erzeugen, in der alle Instanzen zusammen mit den dazugehörigen Klassen-IDs abgelegt werden.

Es wurde erwähnt, dass die GenRepr sehr einfach zu verstehen ist. Dies bezog sich auf den Aufbau der Darstellung. Eine andere Sichtweise entsteht, wenn ein Entwickler die intern gespeicherten Daten betrachten möchte. Hier sieht er bei der GenRepr nur die *„flache“* Tabelle, wohingegen die SpecRepr einer Vorsortierung der Informationen entspricht. Der Inhalt der SpecRepr kann so einfacher verstanden werden.

Die Unterschiede in den Datenmodellen haben auch Auswirkungen auf die Anfragegeschwindigkeit. Für Anfragen können Schemainformationen relevant sein. In diesem Fall spricht man von *Schemaanfragen*. Anfragen in denen Schemainformationen nicht relevant sind werden *Datenanfragen* genannt. Hier jeweils ein Beispiel:

- *„Nenne alle Ressourcen der Klasse A.“* – Bsp. einer Schemaanfrage.
- *„Nenne alle Aussagen mit der Ressource X als Subjekt.“* – Bsp. einer Datenanfrage.

Datenanfragen können auf der GenRepr schnell durchgeführt werden, jedoch ist die GenRepr für Schemaanfragen weniger gut geeignet. Bei dem obigen Beispiel einer Schemaanfrage gehören neben den direkten Instanzen der Klasse *A* auch die Instanzen der Unterklassen von *A* mit zum Ergebnis. Die Klassenhierarchie muss in der GenRepr dabei über Join-Operationen

[48] PostgresSQL, Open-Source-Datenbank, online unter: http://www.postgresql.org

erstellt werde, was sehr zeitintensiv ist. Bei der SpecRepr ergeben sich durch die Aufteilung in Schemainformationen und Daten deutlich schnellere Antwortzeiten für Schemaanfragen. Hier stellt jedoch die oben aufgeführte Datenanfrage ein Problem dar. Um alle Aussagen mit einem bestimmten Subjekt zu finden, müssen alle Eigenschaftstabellen durchsucht werden. In [ACKP01] sind diesbezüglich einige Tests mit Beispielanfragen zu finden, die dieses Problem verdeutlichen.

Im vorherigen Kapitel haben wir gesehen, wie wichtig Kontextinformationen sind. Ihre Bedeutung wird für die meisten Anwendungsszenarien weiter steigen. Daher werden alle Repräsentationsformen sich mit der Darstellung von Kontext auseinander setzen müssen. Wie bereits gesehen, erweitern viele Ansätze zur Kontextmodellierung das RDF-Modell von Tripel auf Quadrupel. Die GenRepr ist hierfür gut geeignet, da die Erweiterung der gegebenen Tabelle um ein weiteres Attribut einfach zu realisieren ist. Tatsächlich bietet Jena auch ein so genanntes Modell-Attribut an, welches gesetzt und verschieden interpretiert werden kann. Für die SpecRepr ist eine entsprechende Erweiterung umfangreicher. In der RDFSuite ist eine Erweiterung diesbezüglich nicht zu finden.

Die folgende Tabelle 3 fasst noch einmal die genannten Vor- und Nachteile dieser beiden Repräsentationsformen zusammen.

Tabelle 3 Übersicht über die Vor- und Nachteile der GenRepr bzw. SpecRepr.

	Performance Datenanfragen	Performance Schemaanfragen	Komplexität der Darstellung	Verständnis gespeicherter Daten	Speicherung von Quadrupel
GenRepr	+	-	gering	schlecht	einfach
SpecRepr	-	+	hoch	gut	aufwendig

Die Unterschiedlichkeit der Anwendungsszenarien für das Semantische Web rechtfertigt die Existenz verschiedener Repräsentationsformen. Die GenRepr und die SpecRepr stellen dabei Extrempositionen dar, die bereits durch ihre Implementierungen Jena und RDFSuite durch Variationen abgewandelt wurden. Sesame [BK02] liegt zwischen diesen beiden Polen. Es baut hauptsächlich auf der GenRepr auf, bildet jedoch für die Klassen- und Eigenschaftshierarchien separate Tabellen, in denen auch die vererbten Hierarchiebeziehungen explizit mit erfasst werden.

4.2 Kombinierte Repräsentation (ComRepr)

Mein Ziel war es, eine Repräsentationsform für RDF zu schaffen, welche alle Vorteile der GenRepr und der SpecRepr für Anfragen vereint und dabei die Speicherung von Kontextinformationen unterstützt. Um dies zu erreichen, wurde der SpecRepr Ansatz um einen Kontextknoten erweitert und mit der GenRepr kombiniert. Das Ergebnis ist die *kombinierte Repräsentation (ComRepr)*, die im Folgenden erläutert wird.

Ausgehend vom RDF-Modell wird jede RDF-Aussage um einen Kontextknoten erweitert. Es werden also Quadrupel der Form *<C, S, P, O>* gespeichert. Die drei Komponenten *S, P* und *O* entsprechen einer RDF-Aussage mit Subjekt, Prädikat und Objekt und stellen RDF-Ressourcen dar. Die weitere Komponente *C* bildet einen Kontextknoten. Diese werden dabei nicht als RDF-Ressourcen angesehen, womit die Darstellung dem *Tripel Plus Context Node*-Ansatz entspricht. Dies ermöglicht eine klare Trennung des internen und externen Kontextes. Durch geringfügige Anpassungen kann die ComRepr jedoch auch für den Named Graphs bzw. den Quadrupel Ansatz verwendet werden. Hierauf wird in 4.2.3 näher eingegangen.

Mit RDF-S3 wurde eine Beispielanwendung entwickelt, welche die ComRepr nutzt. Dabei sind die Kontextknoten fest definiert und enthalten die Information, von welcher Quelle aus eine Aussage geladen wurde und wann dies geschehen ist. Identifiziert wird der Kontextknoten dabei nach außen über die URL der Quelle. Je nach Anwendung können hier auch andere oder

zusätzliche Informationen relevant sein. Um den späteren Umgang mit RDF-S3 zu erleichtern, wird in der folgenden Beschreibung des relationalen Modells der ComRepr die Namensgebung der Tabellen und die entsprechende Interpretation der Kontextknoten verwendet.

4.2.1 Aufbau der kombinierten Repräsentation

In der ComRepr werden die Quadrupel sowohl in einer flachen Tabelle (GenRepr), als auch in einer schemaorientierten Darstellung (SpecRepr) gespeichert. Die SpecRepr wird dazu erweitert, um die Kontextknoten zu den Aussagen speichern zu können. Durch die Kombination der GenRepr und der SpecRepr findet eine redundante Speicherung der Quadrupel statt. Um hierbei den zusätzlichen Speicherbedarf zu minimieren und die Konsistenz einfacher bewahren zu können, macht es Sinn die Ressourcen und Kontextknoten auf interne IDs abzubilden. Diese können in den weiteren Tabellen verwendet werden. Die Abbildung der Kontextknoten auf die internen IDs wird durch die Tabelle *sources* verwirklicht. Für die Abbildung der Ressourcen wird die Tabelle *resources* verwendet. Durch deren Attribut *isLiteral* können Literale erkannt werden. Alle weiteren Informationen über die Literale, wie Sprachdefinition, Datentyp und die eigentlichen Werte (Value), werden in der Tabelle *literals* gespeichert.

Literale besitzen keine URI-Referenz. In 2.3 wurde bemerkt, dass es problematisch ist, zwei Literale gleichzusetzen. Daher wird jedes Literal unabhängig davon, ob bereits ein Literal mit gleichem Wert, Sprach- und Datentyp gespeichert sein sollte, erneut gespeichert und mit einer eigenen internen ID versehen. Der dazugehörige Kontextknoten kann daher direkt mit dem jeweiligen Literal in der Tabelle *literals* abgelegt werden.

Ressourcen hingegen, die über eine URI-Referenz verfügen, können durch diese identifiziert werden. Sie können in verschiedenen Quellen vorkommen und somit verschiedenen Kontextknoten zugeordnet werden. Um dieser Tatsache gerecht zu werden, wird die Tabelle *resources_source_usage* verwendet. In ihr werden zu den Ressourcen mit URI-Referenz die entsprechenden Kontextknoten vermerkt.

Die bisher genannten Tabellen mit ihren Attributen werden zusammenfassend in Abbildung 9 grafisch dargestellt. Durch die Fremdschlüsselbeziehungen werden dabei die logischen Beziehungen verdeutlicht. Die Tabellen *resources*, *literals* und *sources* werden sowohl für die GenRepr als auch für die SpecRepr benötigt. Sie werden daher auch als *Basistabellen* bezeichnet.

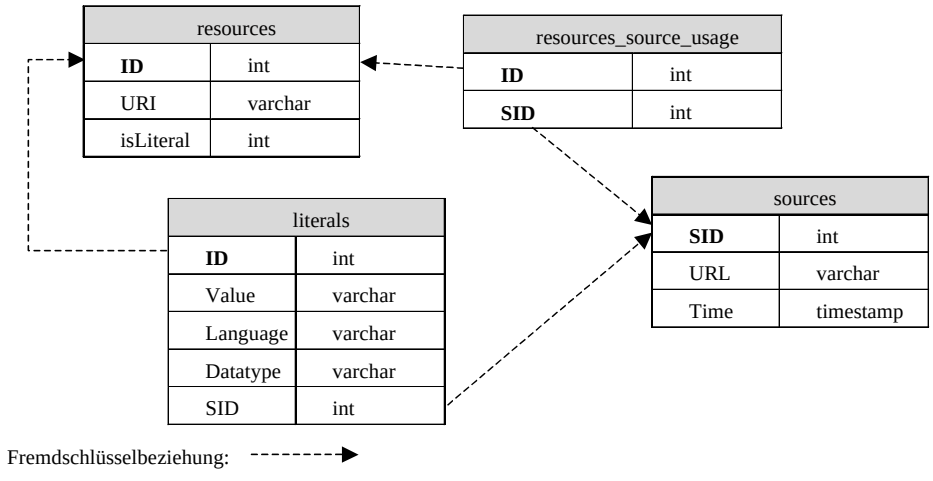


Abbildung 9 Darstellung der Basistabellen, die zur Abbildung auf interne ID benötigt werden. Schlüsselattribute werden durch fetten Schriftschnitt hervorgehoben.

Für die GenRepr wird eine Tabelle hinzugefügt, die den Namen *poi* (*Point Of Interest)* trägt. Dieser Begriff wird im Kapitel 5 über die Anfragesprachen näher erläutert. In der *poi*-Tabelle

werden die Quadrupel mithilfe der internen IDs dargestellt. Das Attribut *ID* steht dabei für das Subjekt, *PID* für das Prädikat, *TID* für das Objekt und *SID* für den Kontextknoten. Das Attribut *isLiteral* dient zur Kennzeichnung, ob es sich beim Objekt um ein Literal oder eine Ressource mit URI-Referenz handelt. Dies bedeutet, dass die Tabelle nicht der 2. Normalform entspricht, da das Attribut *isLiteral* nur von *TID* abhängt. Die Einführung dieses Attributes beschleunigt jedoch die Ausführung von Anfragen, da so nicht extra in der Tabelle *resources* geprüft werden muss, ob das Objekt ein Literal ist. In Abbildung 10 ist die *poi* Tabelle mit ihren Fremdschlüsselbeziehungen dargestellt.

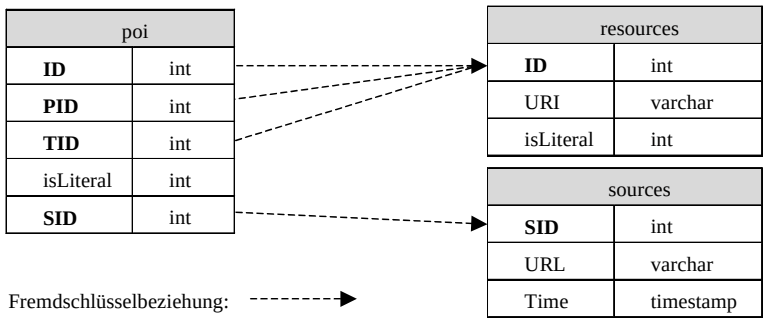


Abbildung 10 Darstellung der *poi* Tabelle. Aufbauend auf den Basistabellen bildet diese die GenRepr der Daten. Dabei stehen die Attribute *ID, PID* und *TID* für Subjekt, Prädikat und Objekt Elemente der RDF-Aussage entsprechend der Reihenfolge. Das Attribut *SID* verweist auf den Kontextknoten des Quadrupel.

Die Speicherstruktur für die SpecRepr ist komplexer und wird im Folgenden auf die einzelnen RDF-Konstrukte aufgeteilt. Zu bemerken ist, dass auch diese als Ressourcen in die Tabelle *resources* eingetragen werden. Um bei RDF-Konstrukten Anfragen zu erleichtern, die auf den Namen, z. B. der Klassen oder Eigenschaften, zugreifen, wurde wie in [Ale+01] eine Auftrennung der URI in den lokalen Namen und den Namensraum vorgenommen. Die URL des Namensraums wird dabei durch eine interne ID substituiert und in der separaten Tabelle *namespaces* gehalten. Auch wird hierdurch Speicherplatz eingespart, da in einem Namensraum üblicherweise mehrere RDF-Konstrukte definiert werden.

4.2.1.1 Klassen

Eine Klasse, welche gespeichert werden soll, wird in die Tabelle *classes* eingetragen, wobei die ID der Ressource, sowie deren Name und die Namensraum-ID gespeichert werden. Damit entspricht diese Tabelle der Klasse *rdfs:Classes*. Soll die Aussage, dass *ex:Person* eine Klasse ist ([ex:Person rdf:type rdfs:Class]) eingetragen werden, so wird deren interne ID, ihr Name und die ID des Namensraumes in die Tabelle *classes* eingetragen. Die Speicherung des Kontextknotens findet wie bei den Ressourcen in einer separaten Tabelle namens *classes_source_usage* statt. Zusätzlich zu einem Eintrag in der Tabelle *classes*, wird für jede Klasse genau eine Tabelle erzeugt, welche die Instanzen der Klasse aufnimmt. Der Name dieser Klassentabelle lautet dabei „*c_*“ gefolgt von der internen ID der Klasse. Jede dieser Klassentabellen hat die Attribute *ID* zum Speichern der internen ID der Instanzen und *SID* für die Angabe des Kontextknotens, der zu der entsprechenden Instanzierung gehört. Der Aufbau wir noch einmal in Abbildung 11 verdeutlicht.

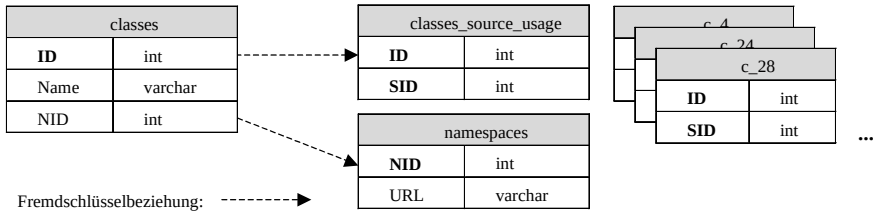


Abbildung 11 Tabellen zur Speicherung von Klassen für die SpecRepr mit Kontextknoten. Die Fremdschlüsselbeziehungen zu den Basistabellen sind hier nicht mit aufgezeigt.

Seien als Beispiel die folgenden vier Quadrupel gegeben, wobei die Quelle *Q1* der URL *http://www.ex.org/Q1* und *Q2* der URL *http://www.ex.org/Q2* entspricht. Steht weiterhin das Präfix *ex* für die URL *http://www.ex.org/Schema.rdf#*, so resultieren daraus die in Abbildung 12 dargestellten Tabellen und Tabelleneinträge.

```
1. [Q1 ex:Person rdf:type rdfs:Class]
2. [Q1 ex:Julia rdf:type ex:Person]
```

} Quelle *Q1*, geladen zum Zeitpunkt t_0.

```
3. [Q2 ex:Astrid rdf:type ex:Person]
4. [Q2 ex:Julia rdf:type ex:Person]
```

} Quelle *Q2*, geladen zum Zeitpunkt t_1.

Auf den ersten Blick scheint die Darstellung sehr komplex und aufwendig. Man beachte jedoch, dass für das vierte Quadrupel nur ein einziger Eintrag in der Klassentabelle *c_12* nötig war. Alle weiteren Informationen waren bereits vorhanden. Generell ist davon auszugehen, dass die Menge der genutzten Klassen und Eigenschaften begrenzt ist. Sind bereits die verwendeten Klassen und Eigenschaften integriert, reduziert sich auch der Aufwand des Speicherns.

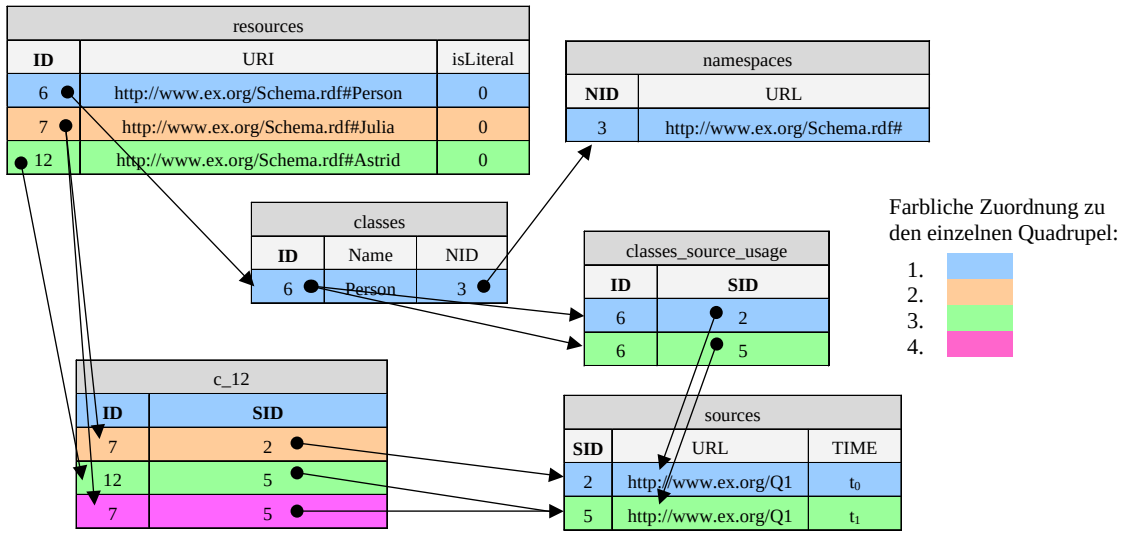


Abbildung 12 Darstellung der vier Quadrupel in der SpecRepr. Die Pfeile dienen dabei der besseren Verdeutlichung worauf durch die internen IDs verwiesen wird. Die Tabellen mit den weiß hinterlegten Attributnamen gehören zu den Basistabellen und sind bereits vorhanden. Die Tabelle *c_12* wird durch das 1. Quadrupel erzeugt.

4.2.1.2 Eigenschaften (Properties)

Eigenschaften werden in die Tabelle *properties* eingetragen. Diese entspricht damit der Klasse *rdf:Properties*. Auch hier werden die ID, der lokale Name und ein Verweis auf den Namensraum gespeichert. Die Verknüpfung mit dem Kontextknoten findet über die Tabelle *properties_source_usage* statt. War die Eigenschaft in der Tabelle *properties* noch nicht eingetragen, so wird eine neue Tabelle angelegt. Diese Eigenschaftstabelle hat, wie in

Abbildung 13 dargestellt, die Attribute *ID* (Subjekt-ID), *OID* (Objekt-ID), *isLiteral* und *SID* (ID des Kontextknotens). Der Name einer Eigenschaftstabelle lautet „*p_*“ gefolgt von der internen ID der Eigenschaft. In ihr werden die Verbindungen, die durch die Eigenschaft realisiert werden, gespeichert.

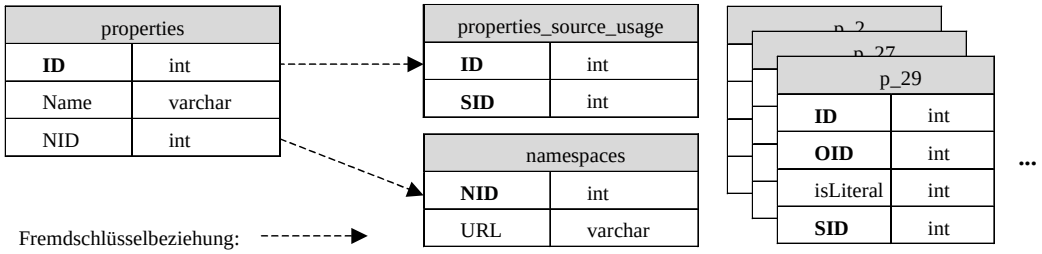


Abbildung 13 Tabellen zur Speicherung von Eigenschaften für die SpecRepr mit Kontextknoten. Die Fremdschlüsselbeziehungen zu den Basistabellen sind hier nicht mit aufgezeigt.

4.2.1.3 Klassen- und Eigenschaftshierarchien

Die Hierarchien für Klassen und Eigenschaften werden über die Eigenschaften *rdfs:subClassOf* und *rdfs:subPropertyOf* erzeugt. Da diese zur Bestimmung von Klassenzugehörigkeiten bzw. zum Erkennen von verwandten Eigenschaften sehr wichtig sind, ist es sinnvoll, diese separat zu speichern. Hierfür werden die Tabellen *subclasses* und *subproperties* verwendet. Der Inhalt dieser Tabellen unterscheidet sich zurzeit nur durch das Weglassen des *isLiteral* Attributes von den Tabellen anderer Eigenschaftstabellen. Dies ist möglich, da für beide Eigenschaften nur Klassen, also Ressourcen mit URI-Referenz, verwendet werden dürfen. Auf Alternativen zu dieser flachen Speicherung wird später auf Seite 62 eingegangen.

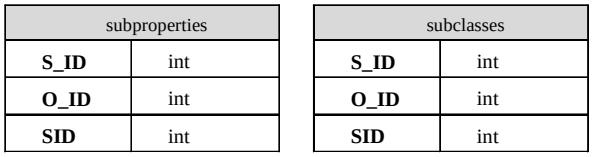


Abbildung 14 Tabellen zur Darstellung der Klassen- und Eigenschaftshierarchien.

4.2.1.4 Container

Container werden in der Tabelle *containers* eingetragen. Hier wird auch der Typ des Containers vermerkt, also ob es sich um eine Menge (*rdf:Bag*), eine Sequenz (*rdf:Seq*) oder eine Alternative (*rdf:Alt*) handelt. Die mit dem Container verbunden Kontextknoten werden in der Tabelle *container_source_usage* festgehalten. Beim ersten Eintrag wird eine *Containertabelle* mit dem Namen „*con_*“ gefolgt von der internen ID des Containers erzeugt. In dieser werden die Elemente des Containers vermerkt. Die Containertabellen haben die Attribute *ID*, *PID* und *SID*. Das Attribut *ID* steht für die IDs der Elemente des Containers. Das Attribut *PID* speichert die ID der Eigenschaft, mit der das Element in den Container eingetragen wurde. Dies ist insbesondere für die Sequenz wichtig, da hier die Reihenfolge semantisch relevant ist. Weiterhin wird über das Attribut *SID* vermerkt, zu welchem Kontextknoten der Eintrag gehört.

Die Abbildung 15 zeigt diesen Aufbau grafisch. Man beachte, dass hier im Gegensatz zu den Klassen und Eigenschaften die URI der Container nicht mit aufgenommen wurde. Für Anwendungen, die sehr viel mit Containern arbeiten, wäre eine entsprechende Erweiterung möglich, um gegebenenfalls die Zugriffszeit zu verkürzen.

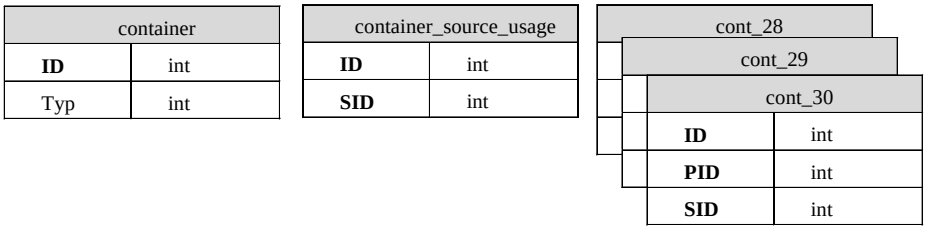


Abbildung 15 Tabellen zur Speicherung von Container in der SpecRepr mit Quellenangabe.

4.2.1.5 Vergegenständlichte Aussagen (reified statements)

Vergegenständlichte Aussagen werden in der Tabelle *statements* eingetragen. Es wird davon ausgegangen, dass eine vergegenständlichte Aussage genau ein Subjekt, Prädikat und Objekt besitzt, auf welches sie verweist. Weiterhin wird davon ausgegangen, dass die Aussagen, die eine vergegenständlichte Aussage repräsentieren, zu einem Kontextknoten gehören. Der Kontextknoten kann daher direkt mitgespeichert werden und muss nicht in einer separaten Tabelle abgelegt werden. Wird in einem Anwendungsfall von der gegebenen Annahme abgewichen, ist dies entsprechend zu modifizieren.

statements	
Statement_ID	int
Subject	int
Predicate	int
Object	int
SID	int

Abbildung 16 Aufbau der Tabelle *statements* zur Speicherung von vergegenständlichten Aussagen.

4.2.1.6 Listen (collections)

Listen werden in RDF-S3 nicht gesondert behandelt. D. h. wenn die Listen-Eigenschaften *rdf:first* und *rdf:rest* verwendet werden, werden diese, wie oben beschrieben, als *normale* Eigenschaften behandelt und gespeichert. Sollte eine Anwendung eine intensive Nutzung von Listen benötigen, ist eine Transformation in eine Abbildung ähnlich der Container denkbar. Als Container-Typ könnte *rdf:List* eingetragen werden. Das Hauptproblem bildet jedoch die ungenaue semantische Definition der Listen. Auch sind deren möglichen Entartungen nicht geklärt und sollten abgefangen werden. Siehe dazu auch Abschnitt 2.3.

4.2.2 Zusammenfassung der kombinierten Repräsentation

Die folgende Tabelle 4 beschreibt noch einmal alle genannten Tabellen, die für die hier beschriebene kombinierte Repräsentation verwendet werden. In der darauf folgenden Abbildung 17 ist eine zusammenfassende grafische Darstellung zu sehen, in der die Attribute der Tabellen und ihre Fremdbeziehungen dargestellt sind.

Man beachte, dass die hier beschriebene Darstellung in einigen Punkten einem Anwendungsfall angepasst werden kann. Abhängig von der gewünschten Interpretation und einer entsprechenden Validierung der zu speichernden Quellen (siehe auch 2.3), kann dies sogar notwendig sein.

An mehreren Stellen, z. B. der Tabelle *poi* wurde, bewusst eine Denormalisierung (nicht 3. Normalform) vorgenommen, um hierüber eine Anfragebeschleunigung zu erreichen. Durch Entfernen des *isLiteral* Attributes könnte die 3. Normalform erreicht werden. Andererseits könnten die Tabelle *statements* oder die Containertabellen (*cont_nnn*) um ein *isLiteral*-Attribut erweitert und damit ebenfalls denormalisiert werden. Bei einer starken Nutzung von Containern

oder vergegenständlichten Aussagen könnte dies ebenfalls eine Anfragebeschleunigung bewirken.

Tabelle 4 Kurzbeschreibung der Tabellen (alphabetisch sortiert), die für die *ComRepr* benötigt werden.

Database Tabellen Name	Beschreibung
c_*nnn*	Für jede Klasse wird eine Tabelle erzeugt. *nnn* wird durch die interne ID der Klasse substituiert. In die Tabelle werden die direkten Instanzen der Klasse mit Referenz auf den Kontextknoten (KK) eingetragen.
classes	Enthält alle gespeicherten Klassen, zuzüglich des Namens und einem Verweis auf den Namensraum.
classes_source_usage	Speichert welche Klassen mit welchen KK verbunden sind.
cont_*nnn*	Für jeden Container wird eine Tabelle erzeugt. *nnn* durch die interne ID des Containers substituiert. In die Tabelle werden die Elemente (members) des Containers zusammen mit der Eigenschaft und dem KK eingetragen.
container	Speichert die Container und ihren Typ, z. B. rdf:alt, rdf:bag, rdf:seq.
containers_source_usage	Speichert welche Container mit welchen KK verbunden sind.
literals	Enthält die Literale (Zeichenketten) inklusive den Länder- und Typangaben und dem KK, mit dem sie verwendet wurden.
namespaces	Dient dazu die Namensräume in den Klassen- und Eigenschafts-Tabellen abzukürzen.
p_*nnn*	Für jede Eigenschaft wird eine Tabelle erzeugt. *nnn* wird durch die interne ID der Eigenschaft substituiert. In die Tabelle werden die Ressourcen, die die Eigenschaft verbindet, der KK und ob das Objekt ein Literal ist eingetragen.
poi	Point Of Interest – entspricht der GenRepr und enthält Referenzen auf Subj., Präd., Obj., KK und gibt an ob das Obj. ein Literal ist.
properties	Enthält alle gespeicherten Eigenschaften, zuzüglich des Namens und einem Verweis auf den Namensraum.
properties_source_usage	Speichert welche Eigenschaften mit welchen KK verbunden sind.
resources	Enthält alle Ressourcen und ermöglicht die Abbildung dieser auf interne IDs, die in den weiteren Tabellen verwendet werden. Für Ressourcen mit URI-Referenz wird diese mit angegeben. Bei Literalen wird das Attribut *isLiteral* auf 1 gesetzt.
sources	Beschreibt die KK, zurzeit werden die Quellenangabe (URL) und der Zeitpunkt des Speicherns vermerkt.
statements	Dient zum Speichern von vergegenständlichten Aussagen.
subclasses	Wird zur Modellierung der Klassenhierarchie durch Speicherung der Subjekte, Objekte und dem KK benötigt.
subproperties	Zur Modellierung der Eigenschaftshierarchie durch Speicherung der Subjekte, Objekte und dem KK.

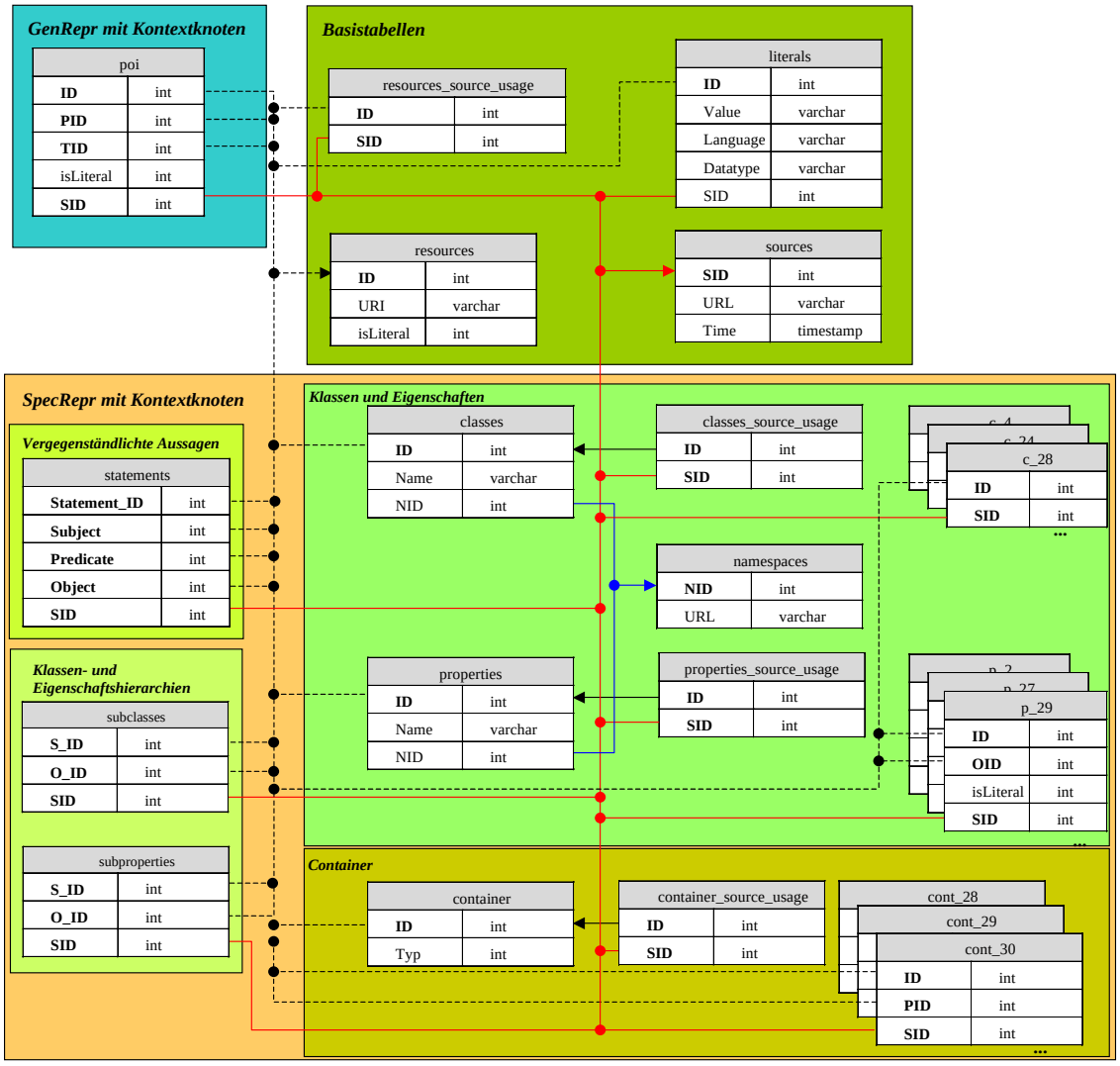


Abbildung 17 Gesamtübersicht über die Tabellen der kombinierten Repräsentation (ComRepr).

4.2.3 Nutzung der ComRepr für andere Kontext-Darstellungsarten

Die ComRepr wurde für den Triple Plus Context Node-Ansatz entworfen. Es stellt sich die Frage, ob die ComRepr auch für die weiteren Darstellungsarten für Kontext, welche im vorherigen Kapitel 3 beschrieben wurden, genutzt werden kann.

Da mit der ComRepr das RDF-Modell dargestellt werden kann, ist offensichtlich, dass sie auch für Darstellungsarten verwendet werden kann, die das RDF-Modell nicht erweitern. Soll keine Kombination einer dieser Darstellungsarten mit dem Triple Plus Context Node-Ansatz verwendet werden, werden die Erweiterungen für die Kontextknoten jedoch nicht benötigt. In diesem Fall würde Speicherplatz verschwendet und unnötiger Aufwand betrieben werden.

Die Ansätze Quads und Named Graphs können zusammen untersucht werden, da sie in ihrer Struktur und Wirkungsweise sehr ähnlich sind. Hierbei werden die Kontextknoten als RDF-Ressourcen angesehen. Es ist daher wichtig, dass die internen IDs für Kontextknoten und für Ressourcen zusammen verwaltet werden. Die Tabelle *sources* müsste daher mit der Tabelle *resources* verschmolzen werden.

Entscheidend bei diesen Varianten ist, dass Aussagen über Gruppen von Aussagen getroffen werden können. Hier noch einmal das Beispiel für Quads aus dem vorherigen Kapitel:

```
[_:cxt _:Julia ex:likes ex:Ballet]
[_:cxt _:Julia ex:daughterOf _:Astrid]
[null _:cxt ex:storedBy “Karsten”]
[null _:cxt ex:storedDate “20.10.2004”]
```

Die letzteren beiden Aussagen gelten hierbei für alle Aussagen, die mit dem Kontextknoten *_:cxt* zusammengefasst werden. Im Beispiel also die oberen beiden Aussagen. Es gibt generell zwei Möglichkeiten zu verfahren:

1. *Explizite Speichermethode* – es werden alle daraus resultierenden Aussagen gespeichert.
2. *Implizite Speichermethode* – es werden nur die gegebenen Quadrupel gespeichert.

Bei der expliziten Speichermethode ist die Frage, wie die resultierenden Aussagen beschrieben werden. Eine Möglichkeit wäre die Nutzung anonymer Ressourcen:

```
[_:1 _:Julia ex:likes ex:Ballet]
[_:2 _:Julia ex:daughterOf _:Astrid]
[null _:1 ex:storedBy “Karsten”]
[null _:2 ex:storedBy “Karsten”]
[null _:1 ex:storedDate “20.10.2004”]
[null _:2 ex:storedDate “20.10.2004”]
```

Möchte man die Information bewahren, welche Aussagen zu welchem Kontext gehören, so müssen folgende Aussagen hinzugefügt werden:

```
[null _:cxt ex:contains _:1]
[null _:cxt ex:contains _:2]
```

Durch die explizite Speichermethode wird die Anzahl der Aussagen in Abhängigkeit der Kontextnutzung stark erhöht. Wie sie hier skizziert ist, entspricht die explizite Speichermethode einer Darstellung mithilfe von vergegenständlichten Aussagen. Die Vorteile der Quadrupelnutzung würden sich nur auf eine verbesserte Handhabung vergegenständlichter Aussagen beschränken. Die ursprünglichen Vorteile würden verloren gehen. Der Vorteil der expliziten Speichermethode ist, dass keine Zeit zum Ableiten impliziter Aussagen benötigt wird. Es stellt sich jedoch die Frage, ob dieser Vorteil nicht durch die entstehende Datenflut und daraus resultierende Mehrarbeit kompensiert wird.

Bei der impliziten Speichermethode muss die Ableitung der weiteren Aussagen während der Anfragen erfolgen. Dies hat jedoch den Vorteil, dass die Quelle unverändert gespeichert wird. Sollten sich die Regeln für die Ableitung später ändern, muss bei der impliziten Speichermethode nur die Schnittstelle angepasst werden. Bei der expliziten Speicherung müssen die impliziten Aussagen wieder entfernt werden, falls dies dann noch möglich ist.

Zusammenfassend kann gesagt werden, dass die ComRepr auch für die Ansätze Quads und Named Graphs verwendet werden kann. Die ID-Verwaltung für die Kontextknoten und Ressourcen muss dabei zusammengelegt werden. Die Hauptarbeit würde hierbei in der Realisierung einer entsprechenden Anfrageschnittstelle liegen.

4.3 RDF-Source related Storage System (RDF-S3)

Das *RDF-Source related Storage System (RDF-S3)* [TW04a, TW04b] stellt eine Beispielimplementierung für die *ComRepr* dar. Es wurde in Java™ entwickelt und steht kostenlos und als Open-Source unter GNU-Lizenz zur Verfügung [49]. Zum Betrieb wird mindestens JRE 1.4 oder höher benötigt. Weiterhin muss ein relationales Datenbankmanagementsystem (RDBMS) installiert sein.

[49] RDF-S3 kann unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/RDFS3/ heruntergeladen werden.

Um eine möglichst breite Akzeptanz und Einsatzmöglichkeit für RDF-S3 zu erreichen, wurden keine besonderen Eigenschaften einer bestimmten Datenbank für die Speicherung verwendet. Insbesondere werden keine objektorientierten Eigenschaften, wie dies z. B. bei der RDFSuite der Fall ist, gefordert. Auch eine Verschiebung einzelner Aufgaben auf die Datenbankseite, z. B. als *Stored Procedures*, wurden unterlassen. Die Eigenschaften, die das RDBMS bieten muss, beschränken sich daher auf eine JDBC-Schnittstelle und die Unterstützung geschachtelter SQL-Anfragen. Empfohlen wird jedoch die Verwendung der DB2 Universal Database (DB2 UDB) von IBM in der Version 8.1, auf der das System entwickelt und stellenweise optimiert wurde. Die Lauffähigkeit wurde aber erfolgreich mit MySQL Version 4.1 zusammen mit MySQL Connector/J Version 3.1.6 getestet.

Im folgenden Unterabschnitt wird die funktionale Ebene betrachtet. Anschließend werden in 4.3.2 der interne Aufbau und das Zusammenspiel mit dem verwendeten RDF-Übersetzer betrachtet. In Unterabschnitt 4.3.3 wird auf Einzelheiten der Implementierung eingegangen. Abschließend folgt eine Leistungsanalyse in 4.3.4.

4.3.1 Funktionalitäten und Benutzungsoberflächen

Zum Speichern von Quellen verfügt RDF-S3 über eine einfach zu bedienende Benutzungsoberfläche, die in Abbildung 18 abgebildet ist. Über diese können alle relevanten Einstellungen hinsichtlich der Datenbankanbindung, der Auswahl der zu speichernden Quellen und den Einstellungen für das Übersetzten (Parsen) getroffen werden. Von ihrem Aussehen her ist die Benutzungsoberfläche der von RDFSuite angepasst, um einen Wechsel zwischen diesen Anwendungen zu erleichtern.

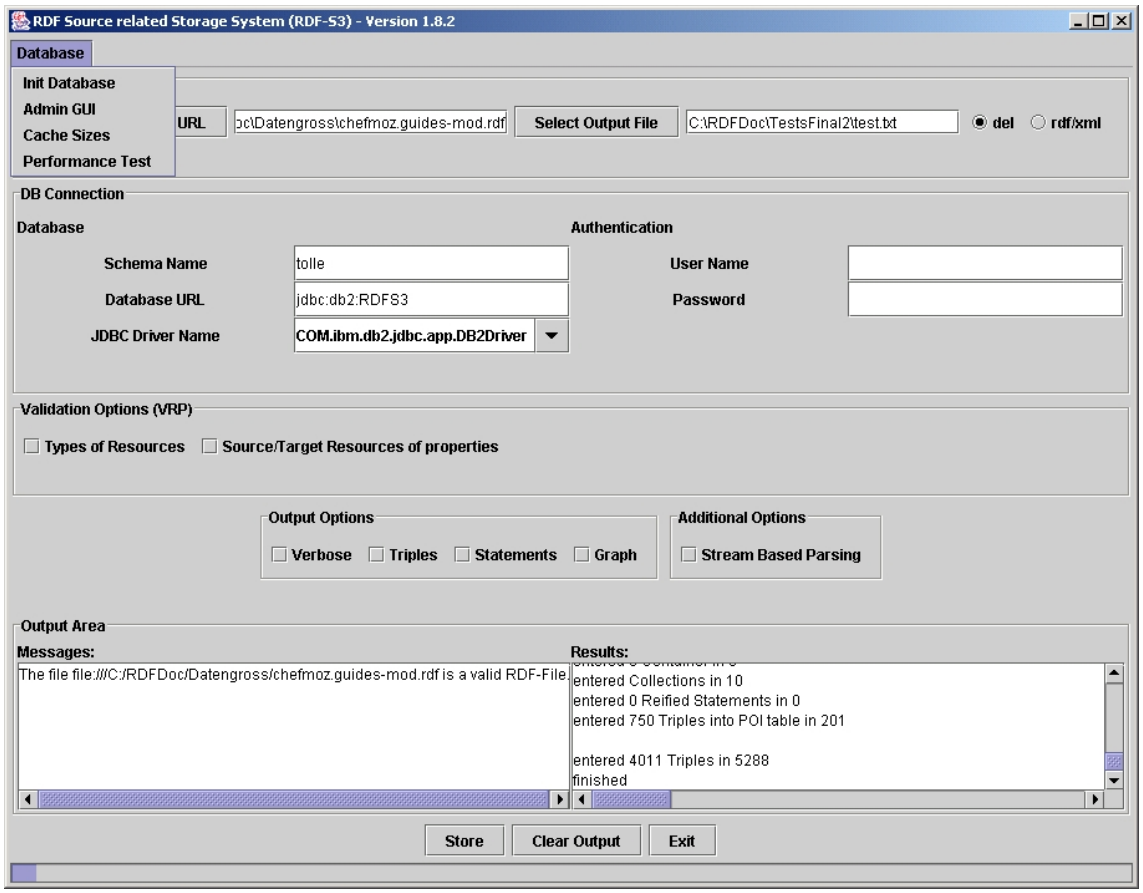


Abbildung 18 Bildschirmfoto der Benutzungsoberfläche von RDF-S3 zum Speichern von RDF-Dateien.

Bevor RDF/XML-Dateien durch RDF-S3 gespeichert werden können, muss die Datenbank initialisiert werden. Nach der Eingabe der nötigen Parameter für die Datenbankverbindung kann dieser Vorgang über die Benutzungsoberfläche von RDF-S3 im Menü *Database* → *init database* gestartet werden. Da die Namensräume von RDF und RDF-Schema immer benötigt werden, werden diese während der Initialisierung eingefügt. Dazu wird eine interne Version dieser Namensräume verwendet, um den Betrieb auch ohne Internetverbindung zu ermöglichen. Beide können jedoch wie jede andere RDF/XML-Datei behandelt und damit auch gelöscht oder aktualisiert werden.

Beim Speichern einer Quelle in RDF-S3 wird jede ihrer Aussagen um einen Kontextknoten erweitert. Der Kontextknoten ist dabei semantisch auf die URL der Quelle und den Zeitpunkt der Speicherung festgelegt. Diese Festlegung schränkt zwar die Nutzung ein, bietet jedoch Sicherheit im Umgang und zusätzliche Funktionalitäten. So können die Informationen der Kontextknoten von RDF-S3 und den darauf aufbauenden Werkzeugen gezielt genutzt werden. Dies ermöglicht RDF-S3 gespeicherte Quellen wieder zu löschen oder zu aktualisieren. Eine Verbesserung insbesondere gegenüber der ICS-FORTH RDFSuite, da in ihr keine Zuordnung einzelner Aussagen zu Quellen möglich ist. Dies ist jedoch sehr wichtig, da nur so der Betreiber, z. B. einer Internetsuchmaschine, unglaubwürdige Quellen löschen und veraltete Quellen aktualisieren kann. Durch diese Verwaltungsmöglichkeit kann er seinen Kunden Daten von höherer Verlässlichkeit anbieten. Auch die Datenmenge kann durch Entfernen veralteter oder unglaubwürdiger Daten reduziert werden.

Das Logo von RDF-S3, welches in Abbildung 19 zu sehen ist, symbolisiert die Beziehung eines Kontextknotens zu einer RDF-Aussage.

Abbildung 19 Das Logo des *RDF-Source related Storage System (RDF-S3)* stellt die Beziehung des Kontextknotens zu einer RDF-Aussage dar.

Für die Verwaltung der gespeicherten Quellen werden die Benutzer (z. B. Betreiber einer Internetsuchmaschine) durch eine Benutzungsoberfläche unterstützt. Diese ist in Abbildung 20 dargestellt und kann über das Menü *Database* → *Admin GUI* der Benutzungsoberfläche von RDF-S3 geöffnet werden. In Tabellenform werden hier alle gespeicherten Quellen mit ihrer URL, ihrer internen ID und dem Zeitstempel, wann sie gespeichert wurden, angezeigt. Durch Klicken auf die Tabellenüberschriften kann eine Sortierung nach der entsprechenden Spalte durchgeführt werden. Die Dateien, die gelöscht oder aktualisiert werden sollen, können entsprechend in den Spalten *Update* bzw. *Delete* markiert werden. Durch Drücken der *Start*-Taste werden die gewählten Aktionen ausgeführt.

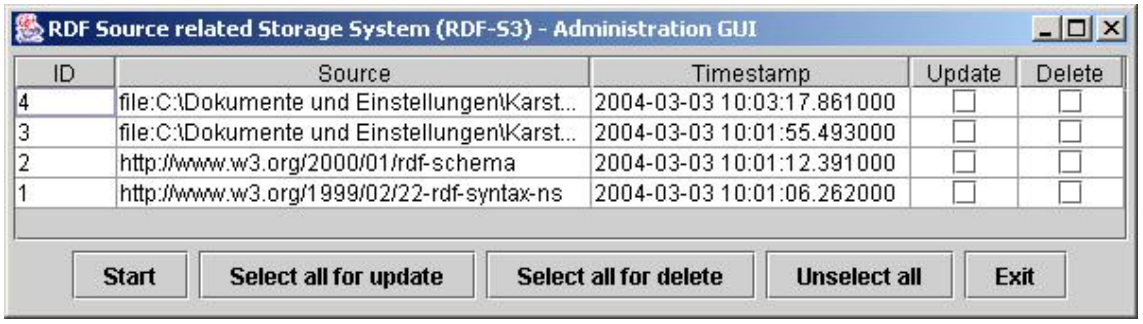


Abbildung 20 Bildschirmfoto der *RDF-S3 Administration GUI* zum Löschen und Aktualisieren einzelner Quellen.

Die semantische Festlegung des Kontextknotens ist weiter motiviert durch die bereits in Kapitel 3 erläuterte Bedeutung für die Glaubwürdigkeit der Daten. Durch die Speicherung der Quellinformation kann ein Benutzer:

a) an der Informationsquelle nach weiteren Informationen zu suchen,

b) die Aktualität der Informationen anhand des Speicherdatums oder an der Quelle überprüfen,

c) durch externe Ranking-Systeme über die Quellen die Glaubwürdigkeit der Daten bestimmen.

Man beachte, dass die Festlegung sich nur auf den externen Kontext bezieht, der über die Kontextknoten realisiert wird. Wird eine weitere Darstellungsart für den internen Kontext verwendet, die ohne Erweiterung auskommt, ist diese von der Festlegung nicht betroffen.

Trotz der Festlegung ist es jedoch möglich weitere Informationen im Kontextknoten zu speichern. Dies kann durch einfache Erweiterungen der Tabelle *sources* und entsprechender Anpassung während der Speicherung geschehen. Denkbar wäre es weiterhin zu vermerken, wer die Quelle gespeichert hat. Auch ein Kommentar, warum die Quelle gespeichert wurde, mag für einige Anwendungsfälle interessant sein. Eine Erweiterung zum Speichern verschiedener Versionen einer Quelle ist ebenfalls möglich. In diesem Fall müsste der Zeitstempel in den Schlüssel für die Tabelle *sources* mit aufgenommen werden.

RDF-S3 baut ebenso wie die RDFSuite auf VRP auf. Damit bietet RDF-S3 die Möglichkeit, die RDF/XML-Dateien vorab semantisch prüfen zu können und so eine höhere Datenqualität zu bieten. Dies umfasst unter anderem:

- Überprüfung der Klassen- und Eigenschaftshierarchien auf Zyklen.
- Überprüfung der Eigenschaften *rdfs:range* und *rdfs:domain* entsprechend der informellen Erweiterung in [W3C Semantic 04] als Beschränkungen (siehe auch Abschnitt 2.3 Punkt 5).
- Plausibilitätsüberprüfung der *rdfs:range* und *rdfs:domain* Definitionen von Untereigenschaften bezüglich ihrer Obereigenschaften.

Meine Erfahrungen als Entwickler von VRP haben gezeigt, dass die Überprüfungen eine deutliche Steigerung der Datenqualität mit sich bringen. Dies stellt eine Besonderheit der RDFSuite und von RDF-S3, gegenüber Jena und anderen RDF-Speicherwerkzeugen, dar. Die Einstellungen für VRP können teilweise über die grafische Benutzungsoberfläche von RDF-S3 beeinflusst werden. Die Überprüfungen auf Zyklen in Klassen- oder Eigenschaftshierarchien sind jedoch immer aktiviert. Werden Fehler entdeckt, so werden diese angezeigt. Der Benutzer kann dann entscheiden, ob er die Daten trotzdem einfügen möchte oder nicht.

4.3.2 Verbindung von VRP und RDF-S3

Der interne Aufbau von RDF-S3 ist in Abbildung 21 dargestellt. Das System baut, wie bereits erwähnt, auf dem *Validating RDF Parser (VRP)* [Tol00] auf. Die Speicherung kann über die Erstellung eines internen Modells oder über die eigens entwickelte *Stream Based API* von VRP erfolgen.

Soll VRP Überprüfungen vornehmen, muss hierfür ein internes Modell im Hauptspeicher erzeugt werden. Dies kann bei großen Dateien zu *Out of Memory*-Fehlern führen. Zwar kann vor dem Start des Programms die Größe des *Java Heaps* über die Parameter *Xms* und *Xmx* eingestellt werden, jedoch werden durch die vorhandene Hardware Grenzen gesetzt.

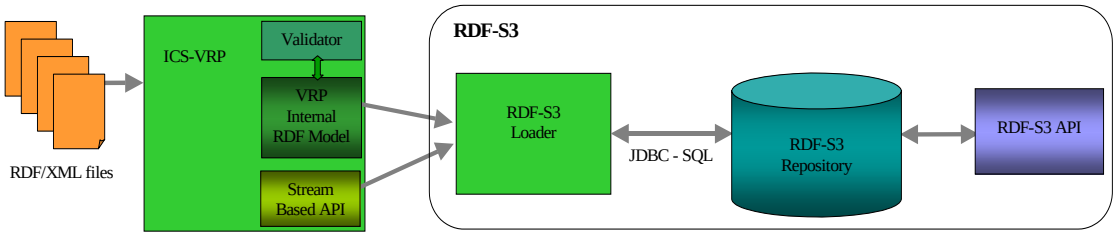


Abbildung 21 Interner Aufbau von RDF-S3. Durch die Pfeile wird der Datenfluss angezeigt.

Für sehr große RDF/XML-Dateien kann daher kein internes Modell aufgebaut werden. Diese können durch den Nutzer in kleinere Dateien aufgespaltet und anschließend einzeln geladen werden. Dieses Vorgehen bringt jedoch zwei Probleme mit sich:

- Beim Aufspalten können Fehler entstehen, z. B. durch Weglassen von Namensraumdefinitionen.
- Das System würde die Dateien als unterschiedliche Quellen ansehen, wodurch ein künstlicher Bruch in Bezug auf die Kontextknoten entstehen würde.

Durch die Stream Based API wurde von mir eine Alternative für große Dateien entwickelt. Bei deren Nutzung wird kein internes Modell aufgebaut, sondern die RDF-Aussagen direkt an RDF-S3 weitergeleitet. Überprüfungen können dabei nicht durchgeführt werden. Für die Stream Based API gelten folgende Regeln, um eine Ressource als entsprechendes RDF-Konstrukt zu speichern:

1. Eine Ressource *R* wird als **Klasse** gespeichert, wenn einer der folgenden Punkte für die zu speichernde RDF-Aussage gilt:
   - das Prädikat ist *rdf:type* und *R* ist das Objekt,
   - das Prädikat ist *rdf:type*, das Objekt ist *rdfs:Class* und *R* ist das Subjekt,
   - das Prädikat ist *rdfs:subClassOf* und *R* ist das Object,
   - das Prädikat ist *rdf:subClassOf* und *R* ist das Subjekt.
2. Eine Ressource *R* wird in RDF-S3 als **Eigenschaft** gespeichert, wenn einer der folgenden Punkte für die zu speichernde RDF-Aussage gilt:
   - *R* ist das Prädikat,
   - das Prädikat ist *rdf:type*, das Objekt ist *rdf:Property* und *R* ist das Subjekt,
   - das Prädikat ist *rdfs:subPropertyOf* und *R* ist das Objekt,
   - das Prädikat ist *rdfs:subPropertyOf* und *R* ist das Subjekt.
3. Eine Ressource *R* wird in RDF-S3 als **Container** gespeichert, wenn einer der folgenden Punkte für die zu speichernde RDF-Aussage gilt:
   - das Prädikat ist *rdf:type*, das Objekt ist *rdf:Bag*, *rdf:Seq* oder *rdf:Alt* und *R* ist das Subjekt,
   - das Prädikat ist *rdf:_nnn* und *R* ist das Subjekt.

4. Eine Ressource *R* wird in RDF-S3 als **vergegenständlichte Aussage** gespeichert, wenn einer der folgenden Punkte für die zu speichernde RDF-Aussage gilt:
   - das Prädikat ist *rdf:subject*, *rdf:predicate* oder *rdf:object* und *R* ist das Subjekt,
   - das Prädikat ist *rdf:type*, das Objekt ist *rdf:Statement* und *R* ist das Subjekt.

Bei kleineren Dateien sollte jedoch auf jeden Fall ein internes Modell erzeugt werden. Aus diesem können Informationen gezielt abgerufen und in die Datenbank eingefügt werden. So ist über das interne Modell bekannt, ob eine Ressource ein spezielles RDF-Konstrukt darstellt oder nicht. Man kann sich dies wie eine Vorsortierung vorstellen, wodurch der Ladevorgang beschleunigt wird. Dieser Gewinn ist abhängig vom Inhalt der Quelle und der Größe des resultierenden Modells. Weiterhin werden für das Laden ohne internes Modell mehrere Threads benötigt, die u. U. aufeinander warten müssen. Tests haben gezeigt, dass in vielen Fällen daher der Geschwindigkeitsgewinn beim Laden die Kosten für den Aufbau des internen Modells überstieg.

Aspekte der Implementierung für das Zusammenspiel von RDF-S3 und VRP werden unter 4.3.3.2 näher behandelt.

4.3.3 Implementierung von RDF-S3

In diesem Unterabschnitt wird ein Überblick über die Implementierung gegeben. Für ein tieferes Verständnis, insbesondere der hier nicht erwähnten Bereiche, sei ergänzend auf die Java-Dokumentation verwiesen. RDF-S3 wurde unter Eclipse Version 2.1 entwickelt. Für die Erzeugung der verschiedenen hier dargestellten UML-Diagramme wurde das Open-Source-Plugin von Omondo für Eclipse[50] genutzt.

In Abbildung 22 ist die Aufteilung des Quellcodes in Pakete abgebildet. Dort enthalten sind auch die Pakete der Anfragesprache *easy RDF Query Language (eRQL)*, die zur Distribution von RDF-S3 gehört. Die entsprechenden Packete für eRQL werden im Kapitel 5 erläutert. Alle Pakete, die hier beschrieben werden, beginnen mit dem Namen *de.jwg.dbis.rdf.rdfs3*. Es wird daher auf sie nur mithilfe des anschließenden Namens verwiesen. Die folgenden Pakete werden noch weiter beschrieben:

- *db* – dient zur Datenbankkommunikation.
- *storage.insert* – realisiert das Speichern der RDF-Daten einer Quelle.
- *storage.delete* – realisiert das Löschen der RDF-Daten einer Quelle.
- *access* – enthält die Klassen der Anfrageschnittstelle von RDF-S3.

Die für das Verständnis weniger wichtigen Pakete werden hier nur kurz aufgeführt. Weitere Informationen können der Java-Dokumentation entnommen werden. Dies sind:

- *gui* – enthält die Klassen für die Benutzungsoberflächen.
- *init* – dient zur Initialisierung der Datenbank, d. h. zur Erstellung der für die ComRepr benötigten Tabellen.
- *util* – enthält verschiedene Helferklassen, insbesondere die Klasse *IOutil*, welche die Datenausgaben auf der Benutzungsoberfläche oder in Dateien steuert.

[50] Die freie Eclipse UML-Version von Omondo ist online unter: http://www.omondo.com/download/free/eclipse_3x/index.html erhältlich.

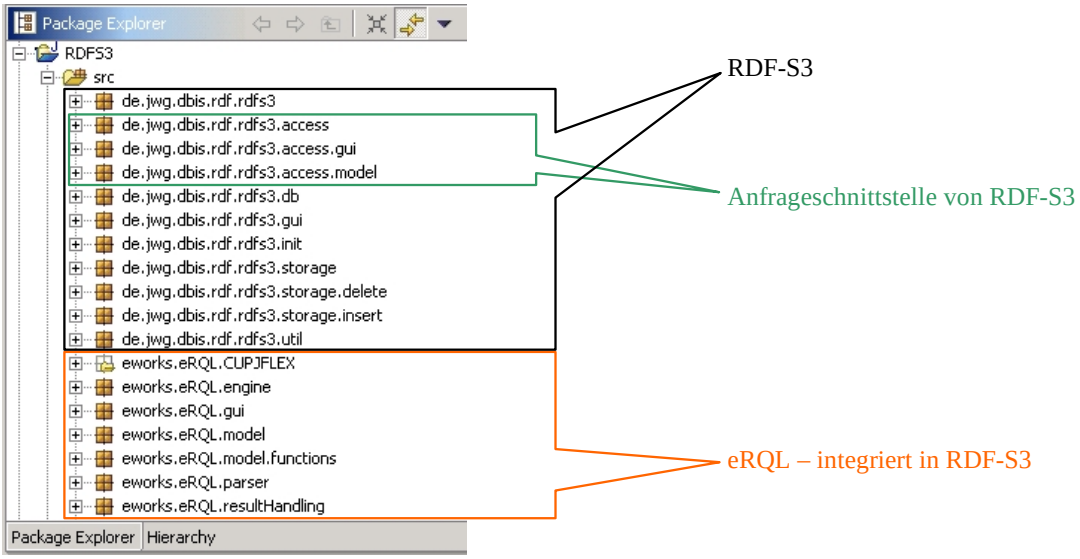


Abbildung 22 Übersicht über die Aufteilung des Quellcodes von RDF-S3 in Pakete (inklusive der Pakete von eRQL).

4.3.3.1 Datenbankzugriff – Paket db

Die Datenbankzugriffe werden im Paket *de.jwg.dbis.rdf.rdfs3.db* (siehe Abbildung 23) gekapselt. Wo es möglich war, wurden dabei SQL-Befehle als *Prepared Statements* vorbereitet. Hierdurch kann für diese bereits ein Zugriffsplan erstellt werden, wodurch die spätere Ausführung beschleunigt wird. Bei der DB2 UDB geschieht dies bei der ersten Anfrage auf die Datenbank, welche daher länger dauert[51].

Die SQL-Befehle, die nur für das Einfügen und Löschen, beziehungsweise für die Anfrageschnittstelle nötig sind, wurden in die Klassen *DBInsert* und *DBQuery* aufgeteilt. Beide erben dabei von der gemeinsamen Oberklasse *DBHelp*, welche grundlegende Funktionalitäten bereitstellt. In der Klasse *DBTableDef* sind die verwendeten Tabellen- und Attributnamen für die ComRepr als Variablen abgelegt. Sollten andere Namen verwendet werden, so kann dies durch Ändern der Variablen in dieser Klasse geschehen, ohne weitere Änderungen am Quellcode vornehmen zu müssen. Eine solche Anpassung ist vor dem Initialisieren und Einfügen von RDF-Daten vorzunehmen.

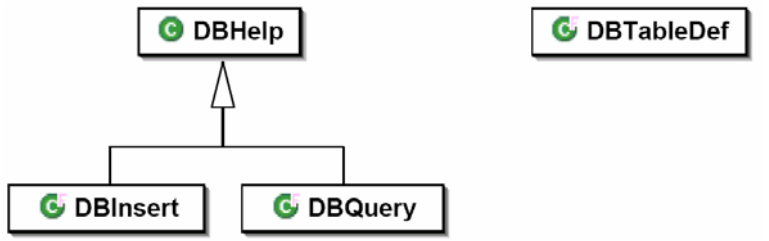


Abbildung 23 UML-Klassendiagramm der im Paket *de.jwg.dbis.rdf.rdfs3.db* enthaltenen Klassen.

4.3.3.2 Einfügen von RDF/XML-Dateien – Paket storage.insert

Das Paket *de.jwg.dbis.rdf.storage.insert* dient zum Speichern der RDF-Dateien. Die Klassen dieses Paketes und ihre Beziehungen sind in Abbildung 24 dargestellt. Das Speichern einer Quelle wird bei der Nutzung eines VRP-Modells über die Klasse *RDFInsert* initiiert. Falls die

[51] Im Gegensatz zur DB2 UDB wird bei MySQL Connector/J bereits beim Verbinden die Vorbereitung der *Prepared Statements* durchgeführt.

Stream Based API von VRP verwendet werden soll, geschieht dies über die von RDFInsert abgeleiteten Klasse *StreamBasedInsert*. RDFInsert enthält allgemeine Funktionen, die beim Speichern benötigt werden. In ihr werden auch Zwischenspeicher (Caches) gehalten, welche die Kommunikation zwischen RDF-S3 und der Datenbank entlasten. Hierauf wird später in 4.3.4.2 weiter eingegangen.

Bei der Nutzung eines internen VRP-Modells wird ein Objekt vom Typ *Model_p* erzeugt. Zu beachten ist, dass das VRP-Modell zwischen einfachen Ressourcen und sogenannte *RDF-Ressourcen*, die zusammen mit Eigenschaften des RDF-Kern-Vokabulars (*rdfs:comment, rdfs:seeAlso, rdf:type, rdf:value*, usw.) verwendet werden, unterscheidet. Die RDF-Ressourcen werden in einem Objekt vom Typ *RDF_Resource* gespeichert. *Einfache Ressourcen* besitzen hingegen nur eine URL und werden von Objekten des Typs *Resource* repräsentiert. Auch im Klassenmodell für RDF-S3 wird diese Aufteilung vorgenommen. Während des Speicherns der Eigenschaftsverbindungen, den Container Einträgen etc., werden die entsprechenden Knoten jeweils nur als Ressourcen gespeichert, unabhängig davon, ob es sich um einfache oder RDF-Ressourcen handelt. Nach dem Eintragen der RDF-Konstrukte werden die RDF-Ressourcen des VRP-Modells noch einmal extra durchlaufen und mit ihren RDF- bzw. RDFS-Eigenschaften gespeichert. Durch die Methode *store()* werden die einzelnen RDF-Konstrukte des VRP-Modells in der Reihenfolge:

- Klassen,
- Eigenschaften und ihre Verbindungen,
- vergegenständlichte Aussagen,
- Container,
- Listen und
- RDF-Ressourcen

gespeichert. Für das eigentliche Speichern existieren Klassen, die den einzelnen RDF-Konstrukten entsprechen. Diese Klassen verfügen alle über eine Methode *store()*, welche die Speicherung des jeweiligen Konstruktes in die ComRepr umsetzt. Das Klassenmodell hierfür baut auf dem von VRP auf. Um das Verständnis zu erleichtern, wurde die Namensgebung der abgeleiteten Klassen von VRP übernommen und jeweils um den Zusatz „*_p*“ für *persistent* erweitert. Neu eingefügt wurde die abstrakte Klasse *RDF_Construct_Abstract*, die gemeinsame Funktionalitäten zur Speicherung von Klassen, Eigenschaften und Container kapselt.

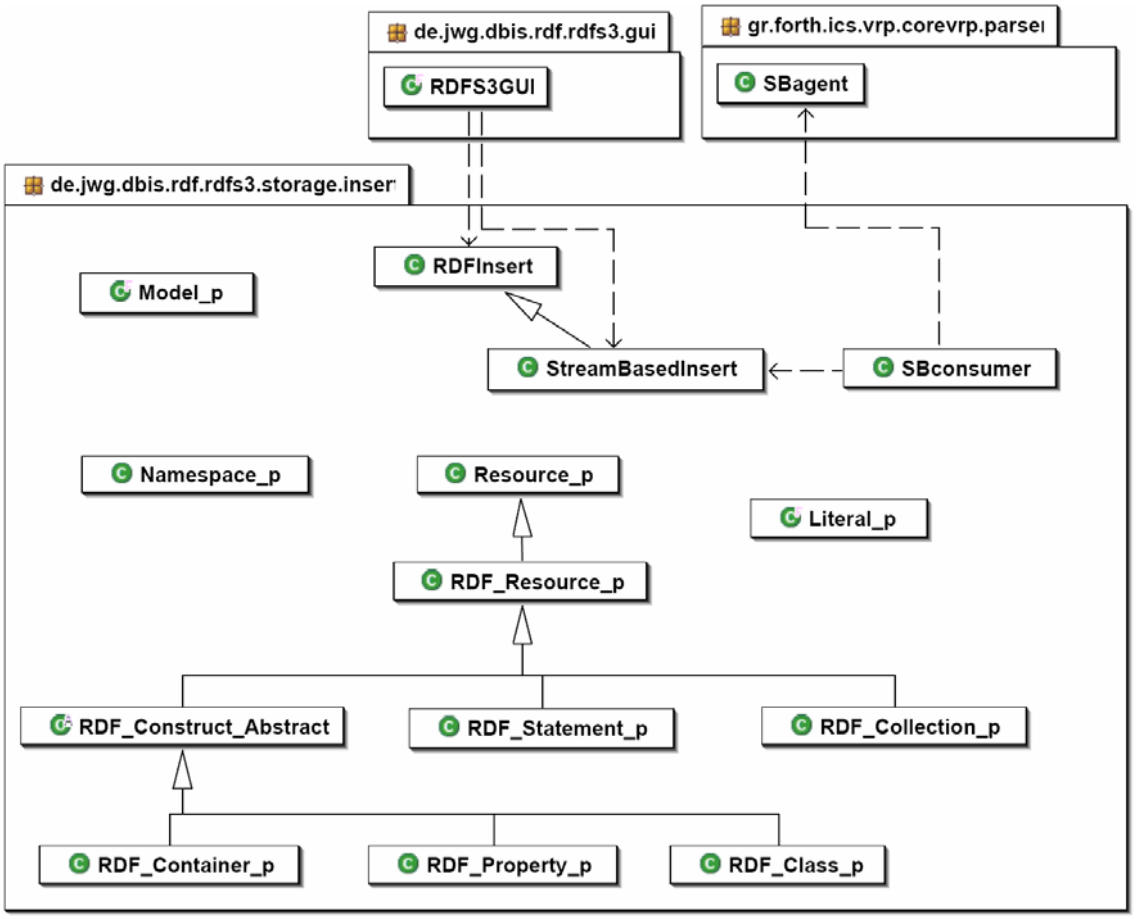


Abbildung 24 UML-Klassendiagramm der im Paket *de.jwg.dbis.rdf.rdfs3.storage.insert* enthaltenen Klassen. Die wichtigsten Abhängigkeiten sind in der Darstellung mit angegeben.

Eine Alternative zu einem weiteren Durchlauf der RDF-Ressourcen, wäre zu versuchen alle RDF-Ressourcen zu speichern, wenn man auf sie beim Speichern der Eigenschafts-Verbindungen trifft. Hierbei ist jedoch zu beachten, dass die RDF-Ressourcen selbst wieder auf RDF-Ressourcen über Eigenschaften des RDF-Kernvokabulars verweisen können. Dabei kann es zu Zyklen kommen. Diese müssten erkannt und aufgelöst werden. Ein Beispiel für einen solchen Zyklus würde durch die folgenden Aussagen gebildet werden:

[ex.ressource1 rdf:value ex:ressource2]

[ex:ressource2 rdf:value ex:ressource3]

[ex:ressource3 rdf:value ex:ressource1]

Weiterhin ist dadurch nicht sichergestellt, dass alle RDF-Ressourcen erreicht werden. Wird eine RDF-Ressource z. B. nur mit der Eigenschaft *rdfs:comment* verwendet, so ist sie nach dem Einfügen aller RDF-Konstrukte noch nicht in der Datenbank gespeichert. Ein weiterer Durchlauf ist also unvermeidbar.

Es wurden Tests durchgeführt diejenigen RDF-Ressourcen, die nur auf Literale, Klassen und einfache Ressourcen verweisen, die also keine Zyklen erzeugen können, früher und vollständig zu speichern. Die Idee dabei war den Aufwand für den Durchlauf der RDF-Ressourcen zu reduzieren. Die Testergebnisse zeigten jedoch keine Leistungssteigerungen, sondern eine Verlangsamung des Systems.

Für die Verwendung mit der Stream Based API von VRP werden die Klassen *StreamBasedInsert* und *SBconsumer* benötigt. Die Stream Based API beruht auf einem

Produzenten-Konsumenten-Prinzip. Hierbei kooperieren zwei Threads miteinander, wobei der eine Thread (Produzent) Ergebnisse produziert, die der andere Thread (Konsument) verarbeitet. Liegen zeitweilig keine Ergebnisse des Produzenten vor, so muss der Konsument warten. Zur Koordination gibt es einen Vermittler (Agent). Dieser arbeitet mit synchronisierten Methoden zur Steuerung der Konsumenten- und Produzenten-Threads. Gestartet wird der Vorgang aus der Klasse *de.jwg.dbis.rdf.rdfs3.storage.insert.StreamBasedInsert* heraus. Zugegriffen wird dabei auf die Klasse *SBconsumer* des gleichen Pakets und auf die Klassen *SBagent* und *SBproducer*, die dem VRP-Paket *gr.forth.ics.vrp.corevrp.parser* angehören. Die Zusammenhänge sind in Abbildung 25 grafisch dargestellt. Grundlegende Informationen und Beispiele zum Produzenten-Konsumenten-Prinzip und dessen Implementierung in Java können im Abschnitt 17.6 von [SS99] gefunden werden.

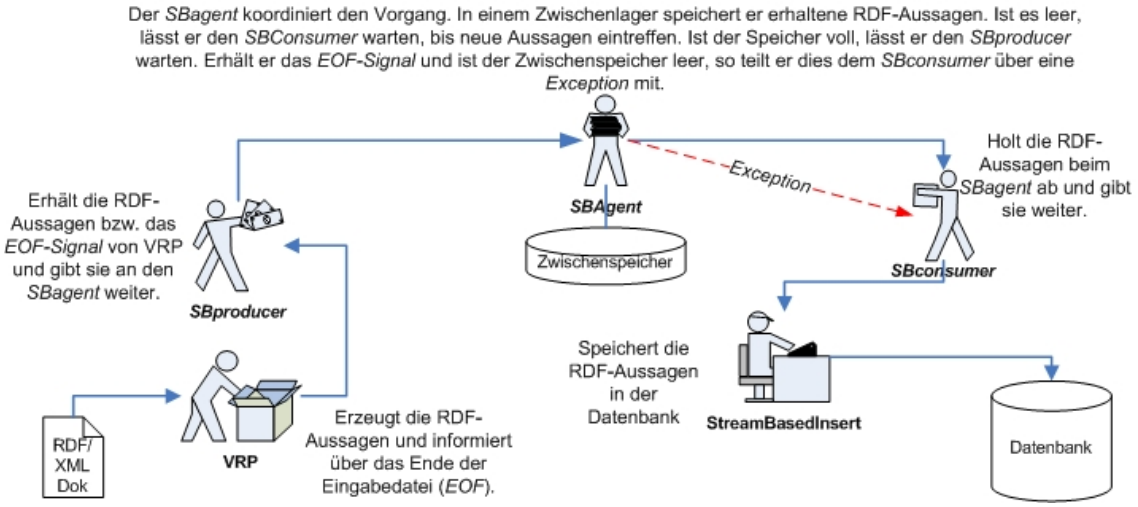


Abbildung 25 Vorgang bei der Nutzung der *Stream Based API*.

4.3.3.3 Löschen und Aktualisieren von RDF/XML-Dateien – Paket storage.delete

Eine Besonderheit von RDF-S3 stellt die Möglichkeit zum Löschen oder Aktualisieren von Quellen dar. Diese Funktionalität wird durch das Paket *de.jwg.dbis.rdf.rdfs3.storage.delete* zusammen mit Fremdschlüsseln innerhalb des relationalen Datenmodells der ComRepr realisiert. Der Vorgang des Aktualisierens entspricht dabei dem naiven Ansatz, dass die Quelle gelöscht und anschließend erneut gespeichert wird. Effektivere, jedoch nicht implementierte Ansätze, werden als zukünftige Arbeiten in 6.1.1 diskutiert.

Das Paket besteht zurzeit nur aus der Klasse *RDFDelete*. Beim Löschen einer Quelle wird in einem ersten Schritt deren Eintrag aus der Tabelle *sources* der ComRepr gelöscht. Die Quellen-ID ist über die Fremdschlüssel mit allen Tabelleneinträgen verbunden, die beim Speichern erstellt wurden (siehe Abbildung 17). Bei der Initialisierung wurden diese Fremdschlüssel mit dem Zusatz „`on delete cascade`“ versehen. Dies führt dazu, dass durch das Löschen des Quelleeintrages aus der Tabelle *sources* auch sämtliche referenzierenden Tabelleneinträge mitgelöscht werden.

Diese Nutzung der Fremdschlüssel zum Löschen minimiert den Kommunikationsaufwand zwischen RDF-S3 und der Datenbank. Auf der anderen Seite bedeuten die Fremdschlüssel einen Mehraufwand während des Speicherns. Dies wird später in 4.3.4 noch untersucht.

Um nicht mehr benötigte Eigenschafts-, Klassen- oder Containertabellen zu löschen gibt es einen Aufräum-Mechanismus (*Garbage Collection)*. Hierzu ein erläuterndes Beispiel: Ist z. B. Quelle *X* die einzige Quelle, in der eine bestimmte Eigenschaft *E* mit der internen ID von *12* verwendet oder definiert wird, so wird beim Speichern von *X* eine Tabelle für *E* mit dem Namen *p_12* angelegt. Nach dem Löschen der Quelle *X* ist diese Eigenschaftstabelle überflüssig.

Entsprechendes gilt für Klassen und Container. Daher werden abschließend nicht mehr benötigte Tabellen entfernt. Beispielhaft für die Entfernung dieser Tabellen sei der SQL-Befehl zum Auffinden der nicht mehr benötigten Eigenschaftstabellen angegeben:

```
select ID from PROPERTIES
     where ID not in
          (select PROPERTIES_SOURCE_USAGE from
               PROPERTIES_SOURCE_USAGE)
```

Des Weiteren müssen die Tabellen *resources*, *properties*, *classes* und *container* bereinigt werden. Auch dort können Eintragungen vorhanden sein, die nicht mehr in den entsprechenden *source_usage*-Tabellen referenziert werden. Hierfür reicht es jedoch die Tabelle *resources* zu säubern, da die Einträge der anderen Tabellen wieder über Fremdschlüssel mit dem Zusatz „`on delete cascade`“ automatisch aktualisiert werden. Für die Ressourcen ist die Quellenzugehörigkeit in der separaten Tabelle *resource_source_usage* abgelegt. Diese wurde durch die Fremdschlüsselbeziehung zur Tabelle *sources* bereits aktualisiert. Durch den folgenden SQL-Befehl können daher die nicht mehr benötigten Einträge in der Tabelle *resources* gelöscht werden. Hierdurch werden die Tabellen *properties*, *classes* und *container* automatisch aktualisiert.

```
delete from RESOURCES where
     isLieral <> 1
   and
     ID not in (select ID from RESOURCES_SOURCE_USAGE)
```

Man beachte, dass einige DBMS geschachtelte SQL-Befehle nicht unterstützen. Dies ist z. B. bei MySQL in der Version 4.0 der Fall. Mit der Version 4.1 von MySQL wurde dieser Nachteil jedoch behoben.

Neben der Benutzungsoberfläche zum Verwalten der Daten (siehe Abbildung 20), kann die Aktualisierung einzelner Quellen auch über ein erneutes Speichern erfolgen. Der Benutzer wird in diesem Fall darauf hingewiesen (siehe Abbildung 26), dass die Quelle bereits gespeichert wurde und kann wählen, ob er die alte Version vor dem Speichern löschen möchte oder nicht. Wird die alte Version nicht gelöscht (*Insert anyway*), bleiben alle bereits gespeicherten Aussagen erhalten und werden durch neue Aussagen ergänzt. Dabei wird für Literale nicht geprüft, ob diese bereits vorhanden sind, sondern sie werden zusätzlich eingefügt. Dieser Vorgang entspricht daher nicht einer Aktualisierung und sollte nur für Testzwecke verwendet werden.

Abbildung 26 Anfragefenster an den Nutzer, falls die gewählte Quelle bereits in der Datensenke gespeichert ist.

4.3.3.4 Anfrageschnittstelle – Paket access

Das Paket *de.jwg.dbis.rdf.rdfs3.access* bildet die Anfrageschnittstelle (API) von RDF-S3. Hierüber können Anwendungen auf die gespeicherten Daten zugreifen. Der Aufbau des Paketes ist in Abbildung 27 dargestellt. Die nötigen SQL-Anfragen werden wenn möglich als *Java-PreparedStatements* in der Klasse *DBQuery* aus dem Paket *db* vorgehalten. Ist dies nicht möglich, werden die Anfragen dynamisch erzeugt. Ein Beispiel hierfür ist der Zugriff auf Eigenschafts- oder Klassentabellen, deren Tabellenname erst während der Laufzeit ermittelt werden kann.

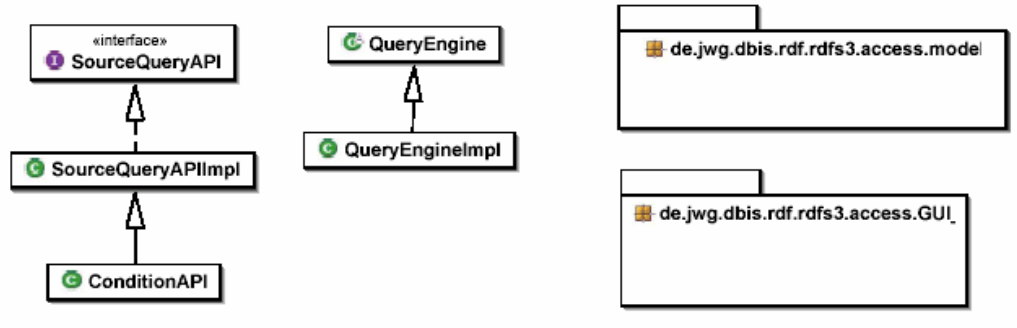


Abbildung 27 Inhalt des Paketes *de.jwg.dbis.rdf.rdfs3.access*, dargestellt als UML-Diagramm.

Die abstrakte Klasse *QueryEngine* und die Klasse *QueryEngineImpl* dienen in der derzeitigen Implementierung von eRQL, die in RDF-S3 integriert ist, als Zugriffspunkt. Die einzelnen Funktionen und Anfragemöglichkeiten, die eRQL benötigt, werden hier in Teilprozesse aufgespalten und auf die in der *ConditionAPI* und *SourceQueryAPIImpl* vorhandenen Methoden abgebildet.

Die Schnittstelle *SourceQueryAPI*, die von der Klasse *SourceQueryAPIImpl* implementiert wird, realisiert dabei vorwiegend schemaspezifische Anfragen und Funktionen. Hierbei wird auf die Tabellen der SpecRepr zugegriffen. Als Beispiel seien die Methode *classes()*, die alle gespeicherten Klassen, oder die Methode *instancesOfClass(String)*, die alle Instanzen einer bestimmten Klasse zurückgibt, genannt. Sämtliche Methoden werden dabei durch Methoden überladen, die den in RDF-S3 gespeicherten Kontextknoten ausnutzen. Als Beispiel existieren die Methoden *classes(ArrayList)* und *instancesOfClass(String, ArrayList)*. In den zusätzlichen *ArrayList*-Variablen kann eine Liste von Quell-URIs angegeben werden, die für die Anfrage berücksichtigt werden sollen.

Die Klasse *ConditionAPI*, welche die *SourceQueryAPIImpl* erweitert, ermöglicht bedingte Anfragen. Hierunter sind Anfragen zu verstehen, die Zeichenkettenvergleiche enthalten. Als Beispiel sei hier die Methode *executeCondition(String cond)* genannt. Durch sie werden alle Quadrupel zurückgegeben, welche die in der Variablen *cond* gegebenen Zeichenkette innerhalb des Subjektes, des Prädikats oder des Objektes enthalten. Weiterhin bietet die ConditionAPI Methoden, welche die Umgebungen von RDF-Aussagen mit bestimmter Entfernung berechnen. Eine solche Umgebung wird auch *Point Of Interest* (*POI*) genannt. Die entsprechende Berechnung erfolgt ausschließlich auf Datenebene, weshalb die GenRepr (also die *poi* Tabelle) genutzt wird. Die Verwendung der DB2 UDB ermöglicht für diese Berechnung eine rekursive SQL-Anfrage (siehe Abbildung 28), wodurch die Anfrage aufseiten der Datenbank optimiert und nur das Endergebnis übertragen werden muss. Wird ein anderes DBMS verwendet, wird auf den rekursiven SQL-Befehl verzichtet, wobei jedoch dann das Übertragen von Zwischenergebnissen nötig ist.

```
with path (ID, P_ID, T_ID, SOURCE_ID, LIT, deep) AS
      (select ID, P_ID, T_ID, SOURCE_ID, LIT, 0 from tolle.poi
            where ID = 12 and P_ID = 19 and T_ID = 20
      union all
            select poi.ID, poi.P_ID, poi.T_ID, poi.source_id, poi.lit,
            p.deep+1 from tolle.poi poi, path p
                  where (p.ID = poi.ID or p.ID = poi.T_ID or
                         p. T_ID = poi.ID or p.T_ID = poi.T_ID)
                  and not
                        (p.ID = poi.ID and p. T_ID = poi.T_ID and
                         p.P_ID = poi.P_ID)
                  and
                        p.deep < 3
      )
select * from path;
```

Abbildung 28 Beispiel für die rekursive SQL-Anfrage, welche die Umgebung mit einer Entfernung von drei um die Aussage mit den internen IDs 12, 19, 20 berechnet.

Ergebnisse auf Anfragen sind vom Typ *Result*. Für die verschiedenen Anfragen, die insbesondere eRQL bietet, wird dieser Ergebnis-Typ noch weiter spezialisiert. Die Klassen hierfür sind im Unterpaket *access.model* enthalten. Die Ergebnistypen stellen dabei Behälter dar, welche die Ergebniselemente aufnehmen. Ergebniselemente basieren ebenfalls auf dem VRP-Model. Da hier zusätzlich der Kontextknoten eine Rolle spielt, wurden einige Klassen des VRP-Modells um diesen erweitert. Dabei steht die Klasse *Context* für den Kontextknoten. Die weiteren Klassen sind *LiteralPlusContext*, *ResourcePlusContext*, *LinkPlusContext* und *RStatementPlusContext*. Die Klasse *RStatemtentPlusContext* ist dabei für vergegenständlichte Aussagen zuständig und enthält die dazugehörigen *rdf:subject*, *rdf:predicate* und *rdf:object* Beziehungen. Für die weiteren RDF-Konstrukte wie Klassen, Eigenschaften und Container werden zurzeit keine eigenen Klassen angeboten. Dies ist momentan für eRQL auch nicht nötig, könnte jedoch im Rahmen einer Erweiterung in Betracht gezogen werden.

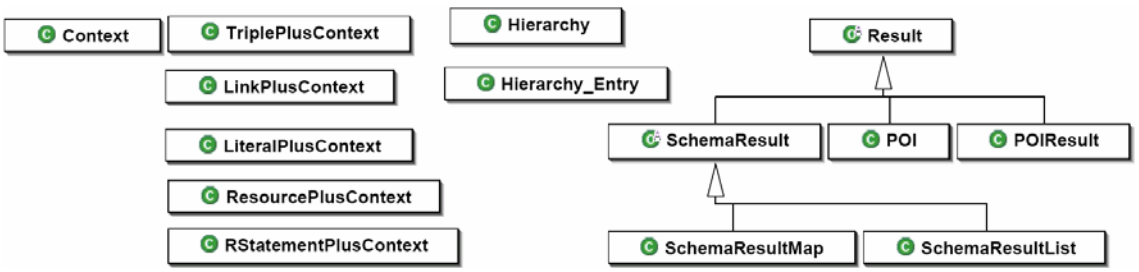


Abbildung 29 UML-Klassendiagramm des Pakets de.jwg.dbis.rdf.rdfs3.access.model.

Die Klassen- und Eigenschaftshierarchien werden in RDF-S3 nur implizit gespeichert. Vererbungen müssen daher während der Laufzeit aus den gewünschten Daten abgeleitet werden. Dazu werden die nötigen Informationen, aus der Tabelle *subclasses* bzw. *subproperties* extrahiert.

Man beachte, dass eine explizite Speicherung in diesem Fall problematisch wäre, da RDF-S3 auch hier die relevanten Informationen über den Kontextknoten begrenzen können sollte. Z. B. werden durch die Methode *subClassOf(String uri, int deep, ArrayList source_urls)* alle Unterklassen für die Klasse mit der in der Variablen *uri* gegebenen URI berechnet. Hierbei werden nur die Informationen aus den Quellen, die in der Liste *source_urls* angegeben sind, berücksichtigt. Verbesserungen durch die Nutzung von z. B. *Labeling Schemes* [CPST03, Sta04] sind daher nicht direkt übertragbar.

Über die Variable *deep* kann weiterhin die Tiefe festgelegt werden, wie weit der Klassenbaum untersucht werden soll. Bei negativen Werten wird der gesamte Baum betrachtet. In der

folgenden Abbildung 30 sind Klassenhierarchieeinträge aus den Quellen *Q1* und *Q2* gegeben und grafisch dargestellt. Weiterhin werden die Ergebnisse von beispielhaften Methodenaufrufen gezeigt.

Quadrupel:
Q1: [ex:Class3 rdfs:subClassOf ex:Class2]
Q1: [ex:Class2 rdfs:subClassOf ex:Class1]
Q1: [ex:Class5 rdfs:subClassOf ex:Class4]
Q2: [ex:Class4 rdfs:subClassOf ex:Class1]

Beispielanfragen mit Ergebnissen:

```
subClassOf(ex:Class1, 1, [Q1;Q2])
```
→ ex:Class2, ex:Class4

```
subClassOf(ex:Class1, -1, [Q1;Q2])
```
→ ex:Class2, ex:Class3, ex:Class4, ex:Class5

```
subClassOf(ex:Class1, -1, [Q1])
```
→ ex:Class2, ex:Class3

Klassenhierarchie:

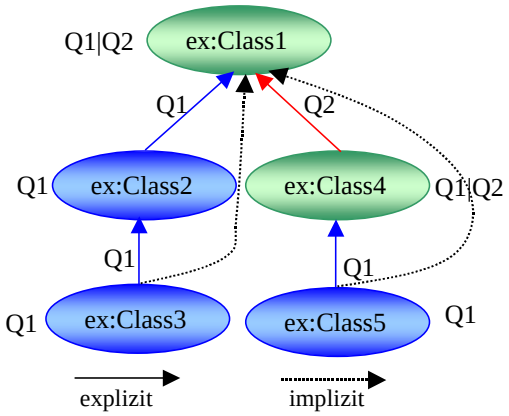


Abbildung 30 Ergebnisse von verschiedenen Aufrufen der Methode *subClassOf* für die gegebene Klassenhierarchie.

Für die Berechnung der impliziten Beziehungen dient die Klasse *Hierarchy*. Dort werden die Verknüpfungen der Ressourcen mit dem dazugehörigen Kontextknoten über die Methode *enter(int subject, int object, int context)* eingetragen. Verwendet werden hier nur interne IDs. In der Klasse *Hierarchy* wird durch die Eintragungen eine Adjazenzliste des Hierarchiebaumes aufgebaut. Jedes Subjekt erhält dabei eine Liste mit Eintragungen vom Typ *Hierarchy_Entry*. In einem Element vom Typ *Hierarchy_Entry* werden das verbundene Objekt und der dazugehörige Kontextknoten gespeichert. Über die Methode *getSuper(int resource, int deep)* können nach dem Aufbau der Hierarchie die Oberelemente zur Ressource *resource* bis zur Distanz *deep* berechnet werden. Hierfür wird rekursiv die Liste *L* der Ressource *resource* um die Listen (wenn vorhanden) der in *L* eingetragenen Ressourcen bis zur angegebenen Distanz erweitert.

4.3.4 Leistungsfähigkeit und Skalierbarkeit von RDF-S3

Die Leistungsfähigkeit (Performance) und Skalierbarkeit (Scalability) eines Systems spielt für viele Anwendungen, z. B. bei einem Informationsportal, eine entscheidende Rolle. Für einen effektiven Einsatz sollte RDF-S3 auf die gegebene Umgebung (Soft- und Hardware) und den Bedürfnissen des Anwendungsfalls angepasst werden. Für RDF-S3 steht in diesem Abschnitt die Speichergeschwindigkeit für die Leistungsfähigkeit im Vordergrund. Die Anfragegeschwindigkeit wird zusammen mit der Anfragesprache eRQL in Kapitel 5 behandelt. Unter Skalierbarkeit wird die Verwendung auch mit vielen Quellen und großen Datenmengen verstanden.

Das oberste Ziel der Entwicklung von RDF-S3 war nicht, eine fertige Anwendung für den Einsatz zu erstellen, vielmehr sollte eine Lösung aufgezeigt werden, welche durch die Kontextnutzung neue Funktionalitäten und erhöhte Glaubwürdigkeit der Daten mit sich bringt. Weiterhin stellt die ComRepr der Daten eine Möglichkeit dar, die Leistungsfähigkeit in Bezug auf Anfragen sowohl der Aussagen- als auch der Schema-Ebene gegenüber anderen Ansätzen zu verbessern.

Damit eine Lösung eingesetzt und akzeptiert wird, muss sie möglichst einfach zu installieren, einfach zu bedienen und hoch portabel sein. Aus diesem Grund wurden in der Entwicklung von RDF-S3 konzeptionelle Entscheidungen in Kauf genommen, welche die Leistungsfähigkeit insbesondere der Speicherung negativ beeinträchtigt haben. Hierzu zählen vorwiegend:

- Keine Nutzung von besonderen Eigenschaften eines RDBMS, welche die Portabilität einschränken würden, wie objektrelationale Funktionalitäten oder Stored Procedures.
- Verwendung von JDBC, um eine einfache Anwendbarkeit und Portabilität zu gewährleisten.
- Starke Nutzung von Fremdschlüsseln für eine hohe Datenkonsistenz und eine Vereinfachung des Löschvorganges einzelner Quellen.
- Redundantes Speichern der Daten (GenRepr und SpecRepr) für schnelle Antworten sowohl bei Daten- als auch bei Schemaanfragen.

Um die Leistungsfähigkeit des Systems und den Effekt von Veränderungen überprüfen zu können, wurde im Paket *de.jwg.dbis.rdf.rdfs3.storage* die Klasse *Statistic* eingefügt. Diese wird genutzt, um während des Einfügens bzw. Löschens die Dauer der Aktionen zu messen und zu sammeln. Zur Zeitdauerermittlung wird die Differenz der Systemzeiten zum Beginn und am Ende einer Aktion berechnet. Je nach Aktion stehen verschiedene Granularitäten zur Verfügung. Für einen Löschvorgang wird nur die Gesamtzeit berechnet. Beim Speichern von RDF-Dateien hängt es davon ab, ob durch VRP ein internes Modell erzeugt oder ob die Stream Based API verwendet wurde. Bei Verwendung der Stream Based API werden nur die benötigte Gesamtzeit und die Anzahl der in der Quelle enthaltenden Aussagen angegeben. Wird ein internes VRP-Modell erzeugt, so wird weiter aufgeschlüsselt, wie viele der jeweiligen RDF-Konstrukte in der Quelle enthalten sind und wie lange der jeweilige Speichervorgang für diese dauert. Die Angaben werden in der Benutzungsoberfläche angezeigt und können in einer Ausgabedatei zur weiteren Verarbeitung gespeichert werden. In der Ausgabedatei können die Informationen als RDF/XML oder als einfache Aufzählung mit Semikolons als Trennzeichen kodiert werden. Das für die RDF/XML-Ausgabe genutzte Vokabular kann in Anlage 8.5 gefunden werden.

Zur Leistungsmessung ist es sinnvoll nicht nur jeweils eine Datei, sondern mehrere hintereinander speichern zu können. Dies kann über die Nutzung von Wildcards bei der Angabe der Quell-URL erreicht werden. Ab der Version 1.7 wurde RDF-S3 zusätzlich mit einer *Leistungsmessfunktion* ausgestattet. Hierbei kann eine Datei mehrmals mit unterschiedlichen Quellnamen eingefügt werden, wobei der Vorgang des Erstellens eines VRP-Modells nur einmal durchgeführt wird[52]. Die Anzahl der Speicherungen kann dabei über die Benutzungsoberfläche (Menüeintrag *Database* → *Performance Test*) vorgenommen werden. Hierdurch kann die Nettozeit einzelner Testreihen erheblich verkürzt werden.

Die vorliegenden Testreihen wurden unter Verwendung der DB2 UDB von IBM v8.1 durchgeführt. Ein Vergleich mit anderen RDBMS wurde nicht vorgenommen[53]. In Unterabschnitt 4.3.4.1 wird die Optimierung der DB2 UDB für RDF-S3 besprochen. Anschließend werden in Unterabschnitt 4.3.4.2 einzelne Maßnahmen und ihr Effekt auf die Leistungsfähigkeit des Systems vorgestellt[54]. Im Unterabschnitt 4.3.4.3 werden mögliche Leistungssteigerungen durch veränderte Konzepte besprochen.

4.3.4.1 Optimierung der DB2 UDB für RDF-S3

Die Optimierung von IBMs DB2 UDB für RDF-S3 in der Version 1.7 war Gegenstand der Diplomarbeit von Jashar Rexhepi. Eine vollständige Beschreibung der Tests und ihrer Ergebnisse können in [Rex05] nachgelesen werden. Hier werden die wichtigsten Ergebnisse der

[52] Diese Funktionalität steht für die Nutzung der *Stream Based API* jedoch nicht zur Verfügung.

[53] Es existieren Benchmark-Tests für den Vergleich von Datenbanksystemen. Eigene Tests können z. B. mithilfe von *PolePosition* erstellt werden. PolePosition ist ein Open-Source Java-Framework. Mit ihm können verschiedene Arten von Datenbanksystemen verglichen werden. Für die Nutzung von JDBC wird dabei MySQL als RDBMS angenommen. Weitere Informationen können online unter: http://today.java.net/pub/a/today/2005/06/14/poleposition.html nachgelesen werden.

[54] Da die Testreihen nur mit DB2 von IBM durchgeführt wurden, lassen sich die Wirkungsweisen der Maßnahmen nicht zwingend auf alle existierenden RDBMS übertragen.

Arbeit vorgestellt. Die Optimierung und Testergebnisse beziehen sich dabei auf die Testumgebung, wie sie in Tabelle 5 dargestellt ist.

Tabelle 5 Testumgebung für die Optimierung der DB2 UDB-Parameter.

Rechner	Prozessor	Arbeitsspeicher	Betriebssystem	RDBMS
IBM ThinkPad R40	Intel Centrino M (1.4 GHz)	512 MB	MS Windows XP Professional (SP1)	IBM DB2 UDB v8.1 (Personal Edition)

Für die Optimierung wurden Testreihen mit vier verschiedenen Dateien durchgeführt. Von diesen enthielten drei nur Daten und keine Schemainformationen. Die vierte Datei (*culture.rdf*) enthielt im Gegensatz hierzu nur Schemainformationen und definiert 12 Klassen und 13 Eigenschaften. Die weiteren Kenngrößen der Dateien waren:

- *culture-data.rdf* (4 KB) mit 51 RDF-Aussagen,
- *chefmoz.guides-mod.rdf* (242 KB) mit 4011 RDF-Aussagen,
- *PerfTest.rdf* (1462 KB) mit 37500 RDF-Aussagen und
- *culture.rdf* (3 KB) mit 40 RDF-Aussagen (Schemainformationen).

Über die Leistungsmessfunktion von RDF-S3 wurden je nach Dateigröße pro Testreihe zwischen zehn und 1500 Speicherungen der einzelnen Dateien durchgeführt. Durch eine Optimierung der Konfigurationsparameter und Einstellungen der DB2 UDB, konnte eine Reduktion der Ausführungszeit von bis zu 75 % erreicht werden. Eine Übersicht der Ergebnisse ist in Tabelle 6 zu sehen. Wesentliche Verbesserungen wurden durch die Nutzung von *DMS-Tabellenbereichen (DMS – Database Managed Space)* auf einer unformatierten Partition und die Erhöhung der Pufferpools (von 250 Seiten auf 7500 Seiten) erreicht. Weitere Einstellungen, die sich gegenüber den Standardwerten positiv bemerkbar machten, sind in Tabelle 7 aufgezählt.

Tabelle 6 Durchschnittliche Reduktionen der Ausführungszeit in Prozent, welche durch die Modifikationen der Standardeinstellungen an DB2 erreicht wurden.

Testdateien	VRP-Modell-Methode	Stream Based Parsing-Methode
culture-data.rdf	75 %	63 %
chefmoz.guides-mod.rdf	71 %	48 %
PerfTest.rdf	71 %	51 %
culture.rdf	51 %	55 %

Tabelle 7 Optimierte Konfigurationsparamter für IBMs DB2 bei der Nutzung mit RDF-S3.

Parameter	Standardwert	optimierter Wert
buff_page	250 Seiten	7500 Seiten
logfil_siz	250 Seiten	2500 Seiten
logprimary	3	5
logsecond	2	3
dbheap	300 Seiten	1000 Seiten
applheapsz	256 Seiten	500 Seiten
pckcache_sz	320 Seiten = 8*maxappls	1000 Seiten

Beispielhaft seien hier die Ergebnisse für die Datei *chefmoz.guides-mod.rdf* erwähnt. Die Datei enthält Daten eines Restaurantführers und wurde jeweils 100-mal mit der Standardkonfiguration und der optimierten Konfiguration gespeichert. Durch das Ergebnisdiagramm in Abbildung 31 wird die gesteigerte Speichergeschwindigkeit deutlich. Zusätzlich zeigt sich, dass sich bei der optimierten Konfiguration auch nach mehreren Speicherungen die Speichergeschwindigkeit nicht verschlechterte. Durch die Optimierung wurde also eine deutliche Steigerung der Skalierbarkeit des Systems erreicht. Aus den ermittelten Werten der optimierten Konfiguration ergab sich ein Durchschnittswert für die Speicherzeit einer RDF-Aussage von 2,96 ms.

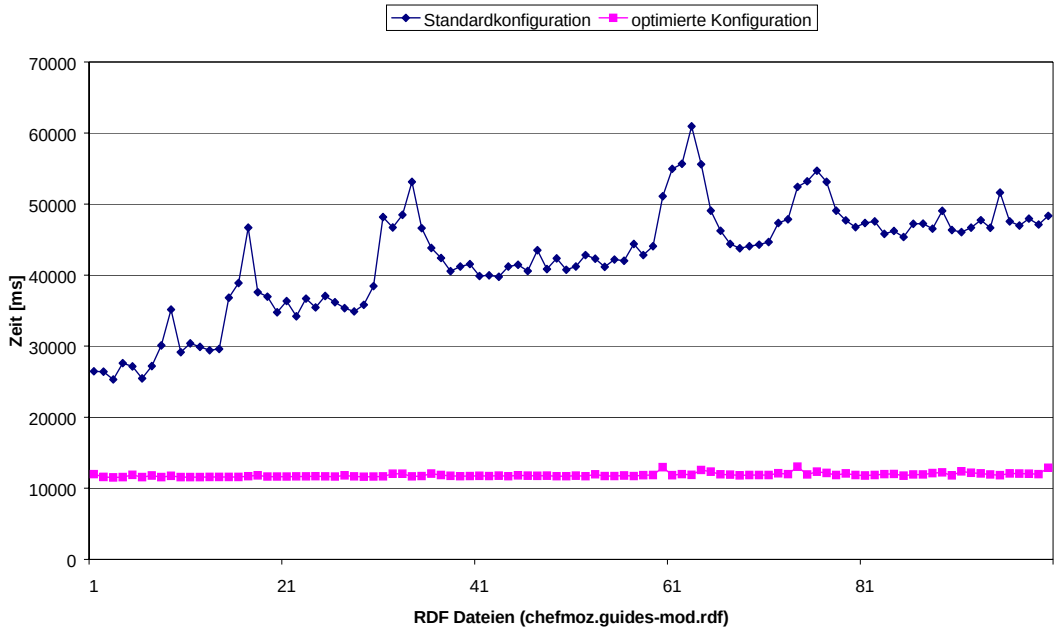


Abbildung 31 Vergleich der Standardkonfiguration gegenüber der optimierten Konfiguration bei 100-facher Speicherung der Datei *chefmoz.guides-mod.rdf*.

4.3.4.2 Einzelne Maßnahmen zur Optimierung von RDF-S3

In die Version 1.7 waren bereits Maßnahmen enthalten, welche die Leistungsfähigkeit steigern sollten. In diesem Unterabschnitt werden die einzelnen Maßnahmen vorgestellt und ihre Auswirkungen durch Testreihen demonstriert. Da ohne Optimierung des DBMS die Leistungsfähigkeit von *RDF-S3 Version 1.7* noch nicht befriedigend war, wurden weitere Maßnahmen untersucht diese zu steigern. Die gefundenen Verbesserungen werden anschließend unter *RDF-S3 Version 1.8* beschrieben.

Die Tests erfolgten auf einem anderen Rechner als die Optimierung im vorherigen Unterabschnitt. Die Testumgebung ist in Tabelle 8 genauer beschrieben. Auch stand in diesem Fall keine unformatierte Partition zur Verfügung.

Tabelle 8 Testumgebung zur Demonstration einzelner Optimierungsmaßnahmen in RDF-S3.

Rechner	Prozessor	Arbeitsspeicher	Betriebssystem	RDBMS
Dell Latitude C610	PIII Mobile Prozessor (1.0 GHz)	512 MB	MS Windows 2000 (SP4)	IBM DB2 UDB v8.1 (Enterprise Edition)

Ziel der Maßnahmen sollte eine Optimierung auch ohne intensiver Anpassung des verwendeten DBMS sein. Aus diesem Grund wurde die Standardkonfiguration mit nur kleinen Veränderungen als Basis für die Tests verwendet. Verändert wurde die Größe des Bufferpools von 250 auf 7500 Seiten und die Größe des Parameters *applheapsz* von 256 auf 512 Seiten. Auch wurden die standardmäßigen *SMS-Tabellenbereiche (SMS – System Managed Space)* verwendet.

RDF-S3 Version 1.7

Im Folgenden werden die einzelnen Maßnahmen für Version 1.7 vorgestellt. In der Abbildung 32 sind die Auswirkungen dieser Maßnahmen verdeutlicht, wobei wie im vorherigen Test, die Datei *chefmoz.guides-mod.rdf* 100-mal gespeichert wurde. Die beste Leistung (unterste Kurve) wird erreicht, wenn alle Maßnahmen aktiviert sind. Bei den Oberen wurde jeweils eine der folgenden Maßnahmen nicht durchgeführt. Die einzelnen Maßnahmen sind:

- **Unterbinden eines automatischen Commit** – bei der Verbindung zwischen RDF-S3 und DBMS wird die Methode `setAutoCommit(false)` verwendet. Der Effekt dieser Maßnahme ist abhängig vom verwendeten DBMS und dessen Recovery-Strategie. Im Fall von DB2, wird vor einem *Commit* nur in Log-Dateien und nicht in die Datenbank selbst geschrieben (*No Undo-Strategie*). Ein Commit führt bei DB2 zum *Flushen* des Cache. Im gegebenen Fall führte ein Weglassen dieser Maßnahme im Durchschnitt zu 63 % längeren Speicherzeiten.
- **Nutzung eines internen Zwischenspeichers für die ID-Abbildungen** – einen Engpass stellt die Kommunikation zwischen der Java-Anwendung und dem DBMS dar. Bei der Speicherung in den verschiedenen Tabellen werden interne IDs verwendet. Diese müssen vor dem Speichern für die entsprechenden Ressourcen und Namensräume bestimmt werden. Um hierfür nicht jedes Mal einen Datenbankzugriff durchführen zu müssen, wurde in RDF-S3 ein Zwischenspeicher (Cache) eingebaut, der die Abbildung von Ressourcen URIs bzw. Namensraum URLs auf die internen IDs ermöglicht.

  Bei der ersten Nutzung einer Ressource wird die ID bestimmt, wofür ein Datenbankzugriff nötig ist. Anschließend wird das URI-ID-Wertepaar in eine Hash-Tabelle abgelegt, wobei die URI den Schlüssel bildet. Bei der weiteren Nutzung der Ressource kann so ein weiterer Datenbankzugriff vermieden werden. Entsprechendes gilt für Namensräume.

  Da der Hauptspeicher endlich ist, ist auch der Zwischenspeicher begrenzt. Standardmäßig ist der Zwischenspeicher für 10.000 Ressourcen und 100 Namensräume ausgelegt. Die daraus resultierende Größe in Byte hängt von den einzutragenden URIs bzw. URLs ab. Die maximale Anzahl von Ressourcen bzw. Namensräumen des Zwischenspeichers kann über das Menü der Benutzeroberfläche verändert werden. Durch Eintragen einer Null kann der entsprechende Zwischenspeicher ganz ausgestellt werden.

  Ist die Kapazität der Zwischenspeicher erschöpft, werden die Einträge gelöscht und entsprechend neu aufgebaut. Durch eine Nachricht wird der Nutzer hierüber informiert, sodass er die Größe des Zwischenspeichers vor dem nächsten Aufruf anpassen kann. Hier

wäre auch eine Ersetzungsstrategie, wie sie sonst für Caches verwendet werden, denkbar. Tests mit dem LRU-Verfahren[55] haben jedoch keine Verbesserungen ergeben. Insbesondere bei kleinen RDF-Dateien bedeutet eine Ersetzungsstrategie nur zusätzliche Verwaltung. Werden nur große RDF-Dateien gespeichert und ist der Arbeitsspeicher klein, könnte die Implementierung einer Ersetzungsstrategie vorteilhaft sein.

- **Batch-Verarbeitung der SQL-Befehle für die Tabelle *poi*** – bei der Nutzung von Batch werden gleichartige SQL-Anweisungen gesammelt und am Block zum DBMS gesendet und ausgeführt. Dies führt zu einer Reduktion des Kommunikationsaufwandes zwischen Anwendung und DBMS (siehe auch [Goo04]). In Version 1.7 wurde dies für die besonders frequentierte Tabelle *poi* erfolgreich eingesetzt. Der Verlauf der roten Kurve in Abbildung 32 zeigt, dass diese Maßnahme sich insbesondere bei einer gefüllten Datenbank positiv auswirkt und so die Skalierbarkeit verbessert. Für den Test ergab sich durchschnittlich eine Verbesserung der Speicherzeit von 9 %.
- ***First 1 Rows only*** – da die URL einer Ressource eindeutig ist, kann dies bei der Anfrage an die Tabelle *resources* zum Bestimmen der internen IDs ausgenutzt werden. Hierzu kann für die DB2 UDB die Anfrage mit dem Zusatz: „FETCH FIRST 1 ROWS ONLY“ versehen werden, wodurch diese nach dem ersten Auffinden eines passenden Eintrages abbricht und diesen zurück gibt. Da dies von anderen DBMS wie MySQL nicht unterstützt wird, ist diese Maßnahme nur für die DB2 UDB aktiv.

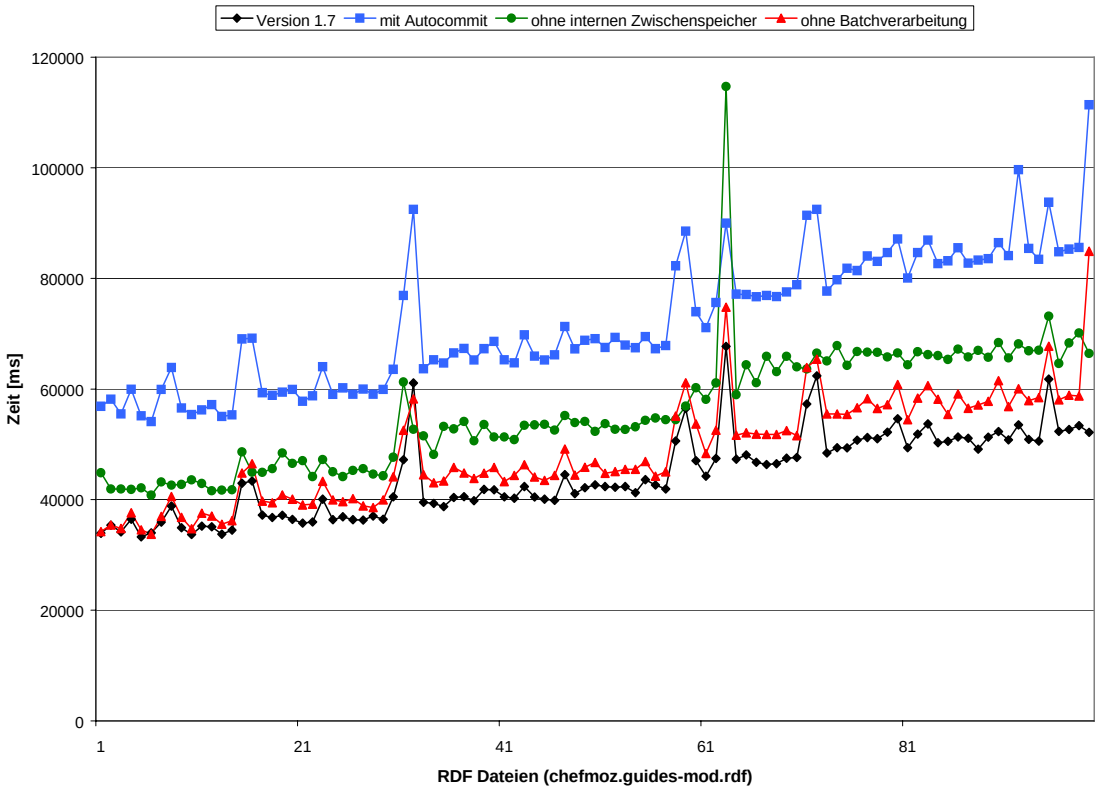


Abbildung 32 Leistungsunterschiede beim Weglassen einzelner Optimierungsstrategien der Version 1.7.

[55] LRU-Verfahren (Least Recently Used), online unter: http://de.wikipedia.org/wiki/Least_recently_used

RDF-S3 Version 1.8

Trotz der Maßnahmen in Version 1.7 bleibt festzuhalten, dass die Speicherdauer sich in den Tests verschlechterte, je mehr Daten in der Datenbank gespeichert waren. Ohne aufwendige Optimierung der Datenbankeinstellungen ist damit die Nutzung des Systems mit größeren Datenmengen nicht akzeptabel. Es wurde daher nach weiteren Möglichkeiten gesucht, die Leistungsfähigkeit und insbesondere die Skalierbarkeit des Systems zu verbessern. Die folgenden beiden Erweiterungen werden ab Version 1.8 von RDF-S3 unterstützt.

- **Reihenfolge in der Primärschlüsseldefinition** – ein positiver Effekt konnte durch eine Änderung der Reihenfolge in der Definition des Primärschlüssels der Tabelle *poi(ID, PID, TID, isLiteral, SID)* erzielt werden. Die Tabelle entspricht der GenRepr, wobei die ersten drei Schlüsselelemente den Subjekt-, Prädikat- und Objektelementen einer RDF-Aussage entsprechen. Das Attribut SID repräsentiert den zugehörigen Kontextknoten. Der zusammengesetzte Primärschlüssel wurde in Version 1.7 der Reihenfolge nach definiert. Beim Einfügen neuer Einträge muss geprüft werden, ob der einzutragende Schlüssel bereitsw existiert. Für diese Prüfung wird scheinbar ein Suchbaum entsprechend der definierten Reihenfolge des Schlüssels aufgebaut. Bei zusammengesetzten Schlüsseln entscheidet die Reihenfolge und Varianz der einzelnen Schlüssel-Attribute über die Breite bzw. die Tiefe des Suchbaumes. Werden die Schlüssel-Attribute entsprechend ihrer Varianz sortiert (höchste zuerst), nimmt die Breite des Suchbaumes zu und die Tiefe ab.

  Für eine Optimierung sollte daher das Attribut mit der höchsten Varianz in der Primärschlüsseldefinition zuerst auftreten. Die größte Varianz weisen in der ComRepr die Objekte auf, dort sind die Literale enthalten, die auch bei syntaktischer Gleichheit durch unterschiedliche IDs repräsentiert werden. Eigenschaften werden hingegen wiederholt und in verschiedenen Quellen verwendet. Die Reihenfolge wurde daher geändert auf *TID, SID, ID* und *PID*. Die Entscheidung, *SID* an die zweite Stelle zu setzen, war dabei willkürlich. Eine entsprechende Optimierung ist von der Information über die Größe und Anzahl der zu speichernden Quellen abhängig.

  Entsprechend wurde für die Erstellung der Eigenschaftstabellen die Reihenfolge des Primärschlüssels angepasst, sodass die ID des Objektes zuerst geprüft wird. Der Effekt wird durch die rote Kurve in **Abbildung 33** verdeutlicht. Diese weist eine deutliche Abflachung und damit eine Steigerung der Skalierbarkeit auf.

- **Intensivierung der Nutzung von Zwischenspeicher** – die Nutzung interner Zwischenspeicher wurde erweitert (blaue Kurve in **Abbildung 33**). Hierbei wurde die Kontextnutzung ebenfalls mit in einen Zwischenspeicher integriert und führte zu einer weiteren Reduktion der Speicherzeit von durchschnittlich fast 9 %.

Indexstrukturen können Anfragen an eine Datenbank verbessern. Insbesondere bei der Speicherung ist hier jedoch Vorsicht geboten, da Änderungen der Daten auch eine Anpassung der Indizes verlangen. Dieser Verwaltungsaufwand kann den Nutzen eines Indexes übersteigen. Ein Index über das Attribut *URL* der Tabelle *resources*, um schneller auf die interne ID zugreifen zu können, führte zu einer Verschlechterung des Speicherverhaltens.

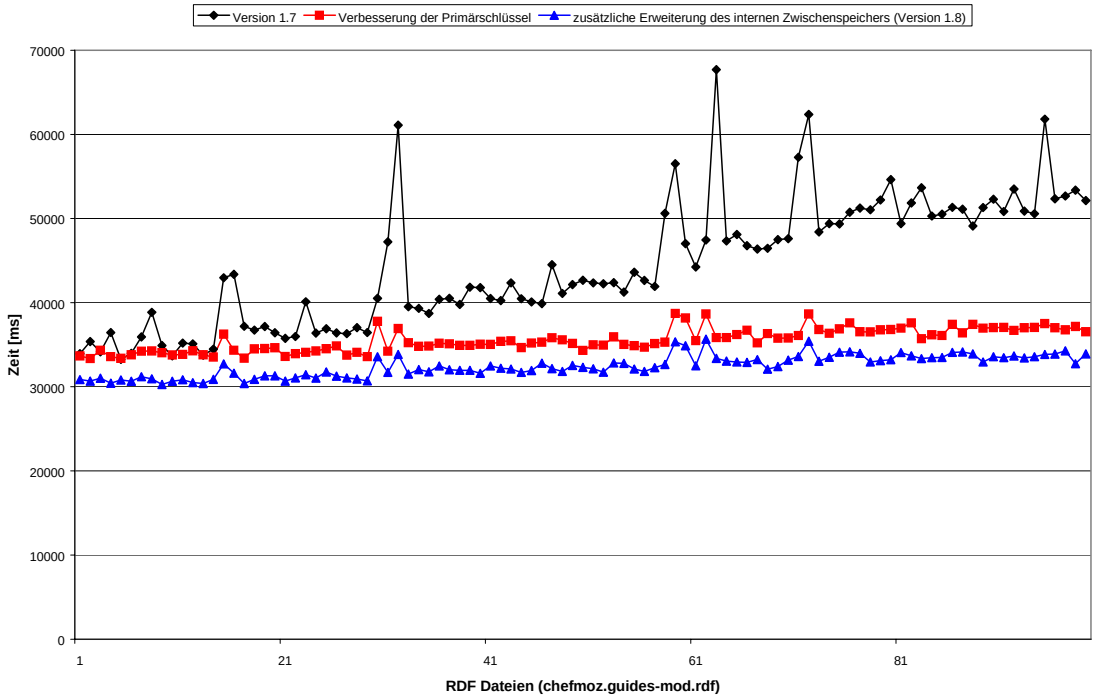


Abbildung 33 Leistungsverbesserung der Version 1.7 durch Verbesserungen der Primärschlüssel und die zusätzliche Erweiterung des internen Zwischenspeichers. Beides wurde in Version 1.8 integriert.

4.3.4.3 Mögliche Erweiterungen zur Leistungssteigerung

Sollte die gegebene Leistungsfähigkeit und Skalierbarkeit von RDF-S3 nicht ausreichen, gibt es verschiedene Möglichkeit diese weiter zu steigern. Wie oben gezeigt, zählt hierzu die Optimierung des verwendeten DBMS. Des weiteren sind folgende Maßnahmen denkbar, die auch getestet wurden:

- **Intensivierung der Batch-Verarbeitung** – bereits in Version 1.7 wurde gezeigt, dass die Verarbeitung der SQL-Befehle als Batch Leistungsvorteile bringt. Bisher werden nur die Eintragungen in die Tabelle *poi* entsprechend behandelt. Dies kann auf weitere Tabellen ausgedehnt werden.
- **Wechsel auf statisches SQL** – durch die Nutzung von JDBC erreicht RDF-S3 einen sehr hohen Grad an Portabilität. Hierbei werden jedoch die Anfragen dynamisch während der Laufzeit gebunden. Durch einen Wechsel auf statisches SQL (z. B. SQLJ) ist mit einer Leistungssteigerung zu rechnen, da der Vorgang des Bindens bereits früher erfolgen kann. Ist die Datenbank gar über mehrere Partitionen verteilt, kann durch die Nutzung von *Buffered Inserts* eine deutliche Leistungssteigerung erwartet werden, siehe hierzu [Poo02].
- **Verzicht auf Fremdschlüssel** – als weitere sehr effektive Maßnahme kann auf die intensive Nutzung von Fremdschlüsseln verzichtet werden. Diese Maßnahme ist in RDF-S3 vorbereitet und kann über der Variable `foreign_keys` innerhalb der Klasse *de.jwg.dbis.rdf.Main.java* vor der Initialisierung vorgenommen werden. Bei der momentanen Implementierung wird hierdurch jedoch das Löschen einzelner Quellen nicht mehr vollständig unterstützt. Auch muss man sich im Klaren über die höheren Gefahren bezüglich der Datenintegrität sein. Da für die Umsetzung der ComRepr zurzeit viele Fremdschlüssel verwendet werden, ist jedoch auch der Leistungsgewinn beträchtlich. Mit den Beispieldaten würde eine Reduktion der Speicherzeit von 33 % erreicht (rosa Kurve in

Abbildung 34). Zwar werden die Fremdschlüssel für das Löschen genutzt, jedoch kann dies auch ohne Fremdschlüssel realisiert werden.

- **Verzicht auf Primärschlüssel** – auch auf einzelne Primärschlüssel könnte verzichtet werden und würde eine weitere Verbesserung mit sich bringen. Beispielhaft wurde der Primärschlüssel der Tabelle *poi* entfernt. Das Ergebnis war eine weitere Reduktion der Speichergeschwindigkeit von fast 8 % (grüne Kurve in Abbildung 34). Dies muss jedoch von Anwendungen, die auf RDF-S3 aufbauen entsprechend berücksichtigt werden, sodass mit eventuellen Duplikaten umgegangen werden kann.

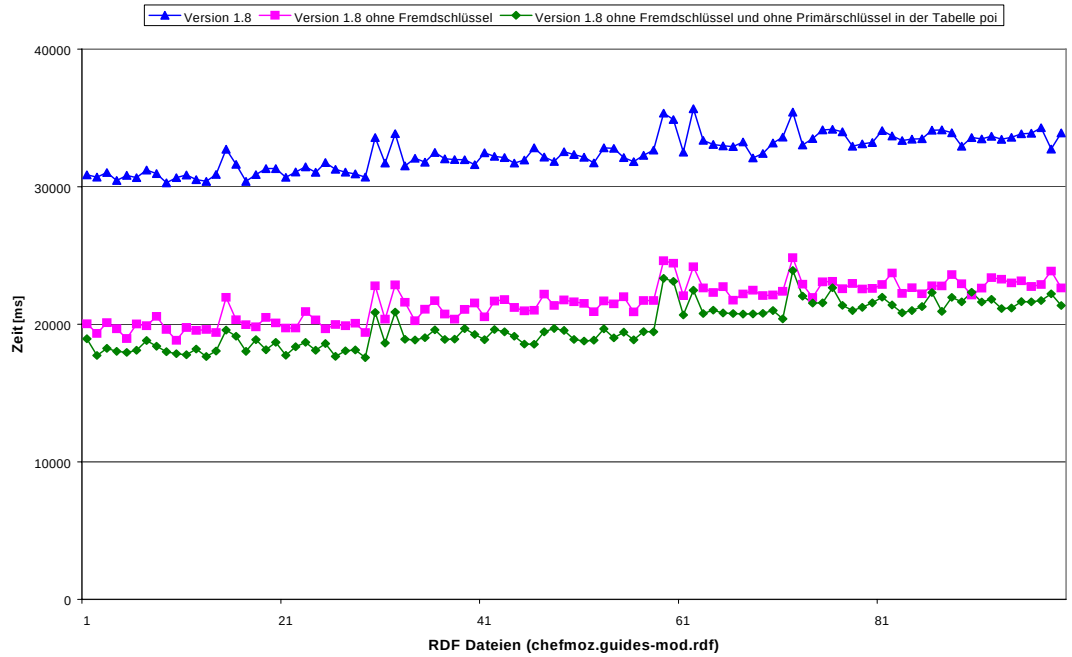


Abbildung 34 Leistungspotenzial durch Weglassen der Fremdschlüssel und des Primärschlüssels der Tabelle *poi*.

4.4 Zusammenfassung

In diesem Kapitel wurde die ComRepr vorgestellt, welche die SpecRepr und die GenRepr kombiniert. Zusätzlich wird in der ComRepr zu jeder Aussage ein Kontextknoten gespeichert, der Metainformationen bereithalten kann. In der vorgestellten Anwendung RDF-S3 wird die URL der Quelle und der Zeitpunkt des Speicherns im Kontextknoten referenziert. Auch wird auf eine klare Trennung der Herkunft dieser Metainformationen geachtet, indem der Triple Plus Context Node-Ansatz verfolgt wird. Dieser Ansatz ermöglicht:

1. An der Informationsquelle kann nach weiteren Informationen gesucht werden.
2. Die Aktualität der Informationen kann anhand des Speicherdatums überprüft werden.
3. Durch externe Ranking-Systeme über die Quellen oder Präferenzen des Nutzers kann die Glaubwürdigkeit der Daten bestimmt werden.
4. Durch die Nutzung von eRQL können diese Informationen bereits bei der Anfrage berücksichtigt werden (siehe Kapitel 5 über eRQL).
5. Eine Quelle kann gelöscht oder aktualisiert werden. Dies ermöglicht eine bessere Wartbarkeit der Daten.

In einem verteilten System, wie dem Semantischen Web, sind diese Punkte unverzichtbar. Ohne sie werden erfolgreiche Anwendungen in Zukunft nicht auskommen können.

Es wurde genauer auf die Implementierung von RDF-S3 eingegangen und Maßnahmen erläutert, mit deren Hilfe eine hohe Leistungsfähigkeit und Skalierbarkeit erreicht wurde. Trotzdem stellt RDF-S3 nur eine Beispielimplementierung dar und es wurden Möglichkeiten zur Leitungssteigerung für einen produktiven Einsatz genannt. Am Beispiel von DB2 UDB v8.1 wurde diesbezüglich gezeigt, wie effektiv eine Optimierung der Datenbankparameter sein kann. Dies lohnt sich daher bereits für kleinere Anwendungen.

Erweiterungen für RDF-S3 werden später in Abschnitt 6.1 besprochen. Dabei wird eine intelligentere Lösch- und Aktualisierungsstrategien theoretisch durchdacht. Auch wird bezogen auf Schemaänderungen untersucht, was bei der Aufhebung des Paradigmas, dass die Dokumente unabhängig voneinander sind, zu beachten ist.

5 easy RQL – Anfragen mit Kontextunterstützung

Ein Informationssystem, welches auf RDF-Daten aufbaut, kann den Nutzern nicht nur die zurzeit gängige Volltextsuche bieten, sondern auch semantische Beziehungen berücksichtigen. Um semantische Anfragen an ein Informationsportal oder eine Internetsuchmaschine stellen zu können, ist eine entsprechende Benutzungsschnittstelle notwendig – beispielsweise auf einer Anfragesprache basierend. Die Anfragesprache muss dabei auch für technisch unversierte Benutzer ähnlich intuitiv einsetzbar sein, wie z. B. die Anfragesprache der Internetsuchmaschine *Google*. Weiterhin sollte es mithilfe dieser Anfragesprache möglich sein, ohne Kenntnisse der im Informationssystem zugrunde liegenden Datenstruktur operieren zu können. Für erfahrene Nutzer ist dabei eine erweiterte Anfragemöglichkeit durchaus wünschenswert, die Zusatzwissen über die RDF-Schemastruktur der Daten einbezieht.

Neben der einfachen Nutzbarkeit der Anfragesprache spielt die Einbeziehung von Kontextinformationen, eine entscheidende Rolle. In Kapitel 3 wurden die Idee und die Nützlichkeit des Kontextes vorgestellt und die Unterscheidung in den *externen* bzw. *internen Kontext* erläutert.

Durch den *externen Kontext* können die Glaubwürdigkeit und der Weiterverwendungswert der Daten erhöht werden. Es sollte daher möglich sein, für eine konkrete Anfrage unglaubwürdige Quellen gezielt auszuschalten oder eine Anfrage von vornherein auf bestimmte, vertrauenswürdige Quellen zu begrenzen.

Die Einbeziehung des *internen Kontextes* in das Ergebnis ist wichtig, um die gesuchten Informationen in mehreren Fundstellen besser finden zu können. In ähnlicher Weise werden z. B. bei Google neben den Fundstellen auch deren umgebende Textpassagen mit angezeigt. Durch die Anzeige des internen Kontextes wird aber nicht nur die relevante Information schneller in den Fundstellen lokalisiert, auch das Verständnis der Fundstellen wird hierdurch erhöht. Nehmen wir als Beispiel die Anfrage: „*Wie breit ist die Golden Gate Brücke?*“ Auf diese Frage reicht „*90*“ als Antwort nicht aus. Ohne eine Angabe, um welche Einheit (Meter, Zoll, Fuß, ...) es sich dabei handelt, kann der Suchende mit dem Ergebnis wenig anfangen. Letztlich darf nicht vergessen werden, dass Nutzer eines Informationsportals oft suchen, ohne genau zu wissen, was sie suchen, wie etwas geschrieben wird (Synonyme/Homonyme) und wie eine optimale Suchanfrage danach aussehen könnte. Es sollte daher ein gewisser Spielraum für die Anfragen eingeräumt werden.

Die wichtigen Fragen für einen Benutzer einer Anfragesprache sind daher:

- *Wie stelle ich eine Anfrage, sodass ich die gewünschten Ergebnisse bekomme?*
- *Kann ich den Ergebnissen vertrauen?*
- *Wo kann ich weiterführende Informationen bekommen?*
- *Kann ich die Ergebnisse einfach und richtig interpretieren?*
- *Wie schnell bekomme ich die Ergebnisse?*

Existierende Anfragesprachen für RDF haben jedoch einen anderen Fokus. Bei ihnen steht die Mächtigkeit im Vordergrund, also die Möglichkeit möglichst viele Anfragen wie möglich in der jeweiligen Sprache ausdrücken zu können. Auch in veröffentlichten Vergleichen von RDF-Anfragesprachen steht oftmals alleine deren Mächtigkeit im Vordergrund, wie z.B. in [HBEV04]. Für eine Internetsuchmaschine ist die Mächtigkeit ihrer Anfragesprache jedoch nur bedingt von Interesse. Der Erfolg von Google liegt sicherlich nicht primär in der

Ausdrucksfähigkeit seiner Anfragesprache begründet[56]. Nein, im Gegenteil – es muss einen guten Grund haben, warum Google allenfalls hinter den Kulissen auf das wesentlich ausdrucksstärkere SQL, und im Vordergrund auf seine eigene, minimalistische Anfragesprache setzt. Um analog zu Google auch in der RDF-Welt entsprechend stärker auf die Bedürfnisse der Benutzer einer RDF-Internetsuchmaschine einzugehen, wurde die Anfragesprache *easy RQL (eRQL)* entwickelt.

Für einen Vergleich mit existierenden Anfragesprachen werden in Abschnitt 5.1 die zwei relativ bekannten Anfragesprachen *RDF Query Language (RQL)* und *RDF Data Query Language (RDQL)* vorgestellt. eRQL wird anschließend in Abschnitt 5.2 zusammen mit seiner Implementierung beschrieben, die auf RDF-S3 aufbaut. Abschließend werden die wichtigsten Erkenntnisse dieses Kapitels in 5.3 zusammengefasst.

5.1 Die Anfragesprachen RDQL und RQL

Die schon in Kapitel 4 genanten Systeme Jena von HP und RDFSuite des ICS-FORTH bieten jeweils auf ihr spezielles Speichersystem zugeschnittene Anfragesprachen an. Für Jena ist dies die *RDF Data Query Language* (RDQL), für RDFSuite die *RDF Query Language* (RQL). Das dritte System *Sesame* unterstützt beide Anfragesprachen und bietet für jede eine eigene Benutzungsoberfläche an. RDQL und RQL werden im Folgenden kurz vorgestellt.

5.1.1 RDF Data Query Language (RDQL)

Die *RDF Data Query Language* (RDQL) basiert auf SquishQL [MSR02]. Sie ist Teil von Jena, wird aber auch von den Systemen Sesame, RDFStore[57], PHP XML Classes[58], 3Store[59] und RAP[60] unterstützt [Sea04]. RDQL ist damit wohl die am weitesten verbreitete Anfragesprache für RDF [Pow03]. Eine übersichtliche Beschreibung von RDQL ist in [Sea04] oder im *RDQL Tutorial* [Sea02] zu finden.

RDQL arbeitet auf dem Graf-Modell von RDF. Die Syntax für RDQL-Anfragen ist an SQL angelehnt. Eine einfache Anfrage hat die Form: *select RÜCKGABE where BEDINGUNG*. Hierbei werden in der Bedingungs-Klausel RDF-Teilgrafen formuliert, die Variablen enthalten. Variablen werden durch vorangestellte Fragezeichen kenntlich gemacht. In der Rückgabe-Klausel wird definiert, welche der verwendeten Variablen zurückgegeben werden sollen. Die Anfrage nach dem Vornamen von *Picasso* könnte folgendermaßen lauten, wobei die beiden verwendeten Eigenschaften bekannt sein müssten:

select ?name

where (?x, <http://www.culture.net/lname>, “Picasso”) ,

(?x, <http://www. culture.net/fname>, ?name)

In dieser RDQL-Anfrage werden als Bedingung zwei so genannten *Graph Pattern* durch Kommata verbundenen. Die Graph Pattern stellen Beispielaussagen dar mit den Variablen als Platzhalter. In den gespeicherten Grafen werden passende Fragmente gesucht und die in den Fragmenten vorhandenen Werte für die Ausgabevariablen dem Ergebnis hinzugefügt. Zusätzlich können Filter über den Operator *AND* angehängt werden. Hierrüber lassen sich die Variablen durch arithmetische Vergleiche, Vergleiche mit Zeichenketten oder andere

[56] Obwohl auch insbesondere Google immer wieder versucht, die Ausdrucksfähigkeit seiner Anfragesprache von Zeit zu Zeit zu erweitern, siehe z. B. online unter: http://www.google.de/intl/de_de/features.html

[57] RDF Store, online unter: http://rdfstore.sourceforge.net/

[58] PHP XML Classes, online unter: http://phpxmlclasses.sourceforge.net/

[59] 3Store, online unter: http://sourceforge.net/projects/threestore/

[60] RAP – RDF API for PHP, online unter: http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/

Bedingungen einschränken. Möchte man nach Vornamen von *Picasso* suchen, die nicht *Pablo* lauten, würde man an die obige Anfrage folgendes anhängen:

AND ?name NE "Pablo"

Hierbei wird die Notationen der Programmiersprache *Perl* verwendet. Das *NE* steht für *„not equal“*.

Am Ende einer RDQL-Anfrage kann eine *USING*-Klausel angefügt werden. In ihr können Präfixe für Namensräume definiert werden, um URIs innerhalb der Anfrage abzukürzen und so komplexe Anfragen kompakter darstellen zu können. Weiterführende Vereinfachungen oder Kurzschreibweisen sind nicht vorgesehen. Als Ergebnis einer RDQL-Anfrage erhält man eine Menge von Werten für die Rückgabevariablen, die den gegebenen Bedingungen entsprechen.

Mithilfe der ebenfalls optionalen *FROM*-Klausel zwischen RÜCKGABE- und USING-Klausel kann durch Angabe der URI gezielt auf ein einzelnes Modell eingeschränkt werden. Dies ist nur dann möglich, wenn z. B. wie bei dem Jena-System die Aussagen in einzelnen Modellen abgespeichert werden. Im gewissen Umfang bietet daher auch RDQL die Möglichkeit, auf Quadrupeln zu arbeiten. Die Bedeutung eines Modells ist RDQL nicht festgelegt. Dies ist zwar sehr flexibel, verhindert jedoch, dass Werkzeuge es eindeutig interpretieren und entsprechende Funktionen mitgeliefert werden können.

5.1.2 RDF Query Language (RQL)

Die *RDF Query Language (RQL)* ist eine typisierte Anfragesprache, die auf einem funktionalen Ansatz aufbaut (ähnlich OQL [Cat+00]). Das formale Datenmodell und Typsystem, auf dem RQL aufbaut, unterscheidet dabei nach einfachen Daten, Schemadaten und Metaschemadaten. Eine vollständige Beschreibung des Datenmodells und Typsystems kann in [Kar+04] gefunden werden. Für den Umgang mit RQL ist das *RQL User Manual* [KC02] zu empfehlen.

RQL bietet eine Reihe von Basisanfragen und Funktionen, wobei der Fokus auf der Abfrage von so genannten *Schemainformationen* liegt. Unter *Schemainformationen* werden die gezielte Suche nach RDF-Klassen, Mitglieder von RDF-Klassen, Ableitungen von RDF-Klassen usw. verstanden. Beispielhaft seien hier nur die Funktionen *subClassOf* und *subPropertyOf* zum Anzeigen von Klassen- und Eigenschaftshierarchien genannt. Die Anfrage *subClassOf(Artist)* würde alle Unterklassen der RDF-Klasse mit dem Namen *Artist* zurückgeben. Als großer Unterschied gegenüber RDQL werden bei RQL-Anfragen generell auch die transitiven Beziehungen berücksichtigt, d. h., bei der Verwendung von RDF-Klassen werden generell auch die davon abgeleiteten Klassen berücksichtigt. Dies kann jedoch durch das ^-Zeichen unterbunden werden. So gibt die Anfrage *subClassOf^(Artist)* nur die direkten Unterklassen der Klasse *Artist* wieder. Gleiches gilt auch für Eigenschaften. Werden alle Verbindungen der Eigenschaft *creates* gesucht, kann dies in RQL über die Kurzschreibweise *creates* erfolgen, also durch einfaches Angeben des Namens der Eigenschaft. Hierbei werden neben den eigentlichen Verbindungen auch die Verbindungen, die durch Untereigenschaften von *creates* gebildet werden, zurückgegeben. Die direkten Verbindungen können durch *^creates* ermittelt werden.

Des Weiteren nutzt RQL ebenfalls die Anfragestruktur: *select ... from ... where ... ,* wie von SQL bekannt und auch von RDQL verwendet. Hierbei werden in der *FROM*-Klausel *Pfadausdrücke* wie z.B. *{X}title{Y}* eingetragen. Pfadausdrücke beschreiben Beziehungen zwischen Ressourcen und binden diese Beschreibungen an Variablen. Ein Beispiel wäre die Anfrage: *select X from {X}title{Y},* die alle Ressourcen zurückgibt, die als Subjekt mit der Eigenschaft *title* verwendet werden. Einfache Pfadausdrücke können durch einen Punkt verbunden werden. Die Anfrage nach Ressourcen vom Typ *Museum* und ihren Änderungsdaten, wenn sie nach 2005 geändert wurden, lautet:

select X, Y from Museum{X}.last_modified{Y} where Y >= 2005-01-01

Als Alternative zur Verbindung von Pfadausdrücken können diese auch einzeln aufgeführt und in der *WHERE*-Bedingung verknüpft werden. Die obige Anfrage ist damit äquivalent zu:

select X, Y from Museum{X}, {Z}last_modified{Y} where Y >= 2005-01-01 and X = Z

Die RQL-Anfragen können über Mengenoperatoren (Vereinigung, Durchschnitt, Differenz) verknüpft werden. Auch die Bildung geschachtelter Anfragen ist möglich. Hierdurch wird RQL zu einer sehr mächtigen Anfragesprache. Durch die Basisanfragen, Funktionen und einige Kurzschreibweisen bleibt sie dabei einfacher zu bedienen, als RDQL [Pow03].

Ergebnisse werden in RQL als RDF-Container zurückgegeben. Hierbei wird zu jedem Rückgabewert auch dessen Typ (Literal, Ressource, Klasse oder Eigenschaft) mit angegeben.

Da RQL auf dem reinen RDF-Datenmodell aufbaut, welches keine Unterstützung eines Kontextknotens bietet, wird auch in RQL kein Kontext unterstützt. Es gibt daher keine Möglichkeit, sich auf bestimmte Modelle oder Quellen bei der Anfrage zu begrenzen oder diese auszuschließen.

5.2 easy RQL (eRQL)

RDQL und RQL lehnen sich von ihrer Syntax, wie die meisten existierenden Anfragesprachen, an SQL an. Damit sind sie jedoch ohne Vorwissen nur schwer bedienbar[61]. RQL bietet zwar Kurzschreibweisen, diese sind aber nur dann wirklich nutzbar, wenn Schemavorwissen vorhanden ist. In vielen Fällen weiß der Nutzer eines Informationsportals oder einer Internetsuchmaschine aber nicht einmal genau, wonach er sucht, geschweige denn, dass er über Schemakenntnisse verfügt. Ein *exploratives* Hangeln durch umfangreiche Klassenhierarchien ist dabei ein gangbarer Weg, aber nur begrenzt zumutbar. Auch sind sowohl RQL als auch RDQL sehr präzise (ähnlich SQL) und finden bereits bei abweichender Eingabe in puncto Groß- und Kleinschreibung die entscheidenden Fundstellen nicht. Zusätzlich enthalten ihre Ergebnisse ausschließlich diejenigen Werte, auf welche die Anfrage zutrifft. Eine Einbeziehung des internen Kontextes wird nicht unterstützt, wodurch der Zusammenhang der Werte verloren geht. Letztlich wird der externe Kontext von RDQL in nur sehr begrenztem Umfang und von RQL gar nicht unterstützt. Eine Reihe von Einschränkungen, die meiner Meinung nach dem Einsatz von RQL und RDQL in Google-artiger Weise im Wege stehen.

Um den Anforderungen einer Anfragesprache aus Sicht der Benutzer eines Informations- oder Suchportals gerecht zu werden, wurde die Anfragesprache *easy RQL (eRQL)* entwickelt. Eine erste Version entstand 2003 unter dem Namen *eRqlEngine* in der Diplomarbeit von Fabian Wleklinski [Wle03]. Hierbei wurden Anfragen ähnlich existierender Internetsuchmaschinen à la Google ermöglicht. Diese wurden von dem eRQL-Prozessor intern in eine oder mehrere äquivalente RQL-Anfragen übersetzt, was sich in der Namensgebung von eRQL widerspiegelt. Auch kann direkt in der eRQL-Anfrage spezifiziert werden, ob und in wieweit umgebende Aussagen zu den Ergebnissen zurückgeliefert werden sollen (*interner Kontext*). Implementiert wurde dieser Ansatz auf dem Validating RDF Parser (VRP) [Tol00] und dessen internem Modell. Damit war es möglich, gezielt in einzelnen Dokumenten oder Quellen zu suchen. Diese stand-alone Version von *eRqlEngine* ist auch heute noch separat erhältlich[62], wurde jedoch zwischenzeitlich weiterentwickelt, wie im Folgenden erläutert wird.

[61] Ironischerweise werden SQL-artige Syntaxen oftmals gerade mit dem Verweis auf intuitive Verwendbarkeit gerechtfertigt. Bezogen auf Datenbankadministratoren und –Entwickler trifft das sicherlich auch zu. Bezogen auf die Besucher eines Informationsportals oder einer Internetsuchmaschine wäre die Voraussetzung von SQL-Kenntnissen sicherlich aber eine zu starke Einschränkung der Zielgruppe.

[62] Die *eRQLEngine*, die ohne Nutzung eines DBMS auskommt, ist weiterhin frei verfügbar unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/eRQL/.

Die Informationen eines Informations- oder Suchportals werden aus verschiedenen Quellen gesammelt. In solchen Fällen ist, wie im Kapitel 3 gezeigt, der externe Kontext wichtig. Insbesondere spielt die Quellinformation der Daten eine entscheidende Rolle für deren Glaubwürdigkeit. In einer Weiterentwicklung wurde eRQL auf RDF-S3 aufgesetzt und um Funktionalitäten erweitert, welche Quellinformationen unterstützen. Da RQL auf dem RDF-Datenmodell aufbaut, welches keine Quellinformationen beinhaltet, bietet auch RQL keine Funktionalitäten hierfür an. Die jetzige Implementierung von eRQL verzichtet daher auf eine Übersetzung nach RQL [TW04a, TW04b, Tol04], anders als der Vorgänger *eRqlEngine* (siehe oben). Einzelne RQL-Funktionen werden jedoch direkt als eRQL-Schemafunktionen (siehe 5.2.1.2) übernommen.

Im folgenden Unterabschnitt 5.2.1 wird auf die Ziele und die Nutzung von eRQL eingegangen. Der Leser soll dadurch die Möglichkeiten von eRQL kennen lernen. Eine etwas ausführlichere Beschreibung mit mehr Beispielen ist im eRQL-Tutorial [Tol05] zu finden. Anschließend wird in Unterabschnitt 5.2.2 eine Übersicht über die Implementierung gegeben und genauer auf die Interpretation und interne Handhabung von eRQL-Anfragen eingegangen. Die Leistungsfähigkeit von eRQL aufbauend auf RDF-S3 wird im Unterabschnitt 5.2.3 diskutiert.

Auf mögliche Erweiterungen von eRQL wird später in Abschnitt 6.2 eingegangen.

5.2.1 eRQL Eigenschaften, Syntax und Nutzung

Für die Entwicklung von eRQL standen die folgenden Hauptziele im Vordergrund:

1. So einfach wie möglich – eRQL soll ohne Vorwissen über die Anfragesprache oder die Struktur der gespeicherten Daten bedienbar sein. Nicht wie SQL, eher wie Google.
2. Unterstützung des internen Kontextes[63] – durch Bereitstellen des internen Kontextes, also der benachbarten Aussagen der Fundstellen (*Umgebung*), sollen gefundene Daten besser verstanden werden können.
3. Unterstützung des externen Kontextes – über den externen Kontext sollen Ergebnisqualität und Glaubwürdigkeit der gefundenen Daten gesteigert werden.

Das erste Ziel wird dabei durch die Anlehnung an existierende Internetanfragesprachen erreicht. So bilden bereits einzelne Suchbegriffe, so genannte *Ein-Wort-Anfragen*, vollständige eRQL-Anfragen. Werden z. B. Informationen über den Künstler Namens Picasso gesucht, wäre die Anfrage *Picasso* denkbar. Hierbei wird geprüft, ob die Zeichenkette des Suchbegriffs in der URI-Referenz einer Ressource oder dem Wert eines Literals enthalten ist. Ist dies der Fall, wird die entsprechende Aussage zusammen mit ihrer Quellinformation der Ergebnismenge hinzugefügt. Je nach Grundeinstellung kann hierbei nach Groß- und Kleinschreibung unterschieden werden oder nicht. Die Ein-Wort-Anfragen von eRQL sind daher nahezu identisch mit den Anfragen gängiger Internetsuchmaschinen: uneingeschränkte Volltextsuche ohne Berücksichtigung von Groß-/ Kleinschreibung.

Welche Vereinfachung durch eRQL erreicht werden kann, wird durch die folgende Abbildung 35 verdeutlicht. Es wird dabei die Anfrage nach allen RDF-Aussagen, die das Wort „bridge“ enthalten in eRQL, RQL und RDQL dargestellt. Die deutliche Vereinfachung der eRQL-Anfrage wird dabei mit erreicht durch die grundsätzliche Ausführung auf allen Teilen einer RDF-Aussage:

[63] Da eine Einigung auf eine bestimmte Darstellungsart für internen Kontext nicht existiert, bezieht sich der interne Kontext hier auf die umgebenen RDF-Aussagen im Graphen.

eRQL-Anfrage:
```
bridge
```

RQL-Anfrage:
```
select * from {X}@P{Y} where
        @P like "*[bB][rR][iI][dD][gG][eE]*"
        or X like "*[bB][rR][iI][dD][gG][eE]*"
        or Y like "*[bB][rR][iI][dD][gG][eE]*"
```

RDQL-Anfrage:
```
select ?s, ?p, ?o where
      (?s, ?p, ?o)
    AND
      ?s =~ "*[bB][rR][iI][dD][gG][eE]*"
      || ?p =~ "*[bB][rR][iI][dD][gG][eE]*"
      || ?o =~ "*[bB][rR][iI][dD][gG][eE]*"
```

Abbildung 35 Die Anfrage nach dem englischen Begriff *bridge* für Brücken in den drei Anfragesprachen eRQL, RQL und RDQL. Der standardmäßige POI-Modus (siehe später) der eRQL-Anfrage wurde hierbei nicht mit in die RQL- bzw. RDQL-Anfrage übersetzt.

Es ist deutlich zu sehen, dass Anforderung 1 („Bedienung durch Laien“) durch eRQL einigermaßen erfüllt wird, durch RQL und RDQL jedoch relativ sicher nicht.

Anmerkung: Wenn das RDF-Schema nicht bekannt ist, nimmt der Vorteil von eRQL gegenüber RQL und RDQL weiter zu. Ohne Schemakenntnisse ist nicht klar, in „welcher *Richtung“* eine Eigenschaft definiert wurde. Möchte man als Beispiel ausdrücken von welchem Autor ein Buch geschrieben wurde, könnte man die Eigenschaften „wurde_geschrieben_von“ oder „hat_geschrieben“ verwenden. Bei der Eigenschaft „wurde_geschrieben_von“ würde man das Buch als Subjekt und den Autor als Objekt eintragen. Bei der Eigenschaft „hat_geschrieben“ wäre dies genau umgekehrt. Obige eRQL-Anfrage würde bei jeder der beiden Richtungen zum Ziel führen, die RQL- und RDQL-Anfrage hingegen müsste streng genommen jeweils zweimal formuliert und durchgeführt werden.

Das Ergebnis einer Ein-Wort-Anfrage besteht aus einer Menge von direkten Treffern. Ein *direkter Treffer* ist dabei eine RDF-Aussage zusammen mit ihrem Kontextknoten (Quadrupel). Mindestens eine Komponente der RDF-Aussage – Subjekt, Prädikat oder Objekt – muss dabei die Zeichenkette der Ein-Wort-Anfrage enthalten. Bei Ressourcen mit URI-Referenz bezieht sich dies auf die URI, bei Literalen auf deren Wert.

Wie in 5.2.1.1 über die eRQL-Modi noch genauer beschrieben wird, enthalten die Ergebnisse von Ein-Wort-Anfragen standardmäßig die so genannte *Umgebung* (*interner Kontext)* der direkten Treffer. Als *Umgebung einer RDF-Aussage X* werden dabei die RDF-Aussagen verstanden, die mit dem Subjekt oder dem Objekt von *X* über eine Kante (Eigenschaft) verbunden sind. Hierbei spielt die Richtung der Kante aus oben genanntem Grund keine Rolle. Diese Standardeinstellung von eRQL ermöglicht es, die Anfrageergebnisse mit ihrer Umgebung und daher in ihrem internen Kontext zu sehen und sie dadurch besser verstehen zu können. Hierdurch wird weiter deutlich, dass das Beispiel oben in Abbildung 35 noch geschönt ist. Die RQL- und RDQL-Anfragen müssten noch weiter aufgebläht werden, wenn man die RDF-Aussagen der Umgebung mit in das Anfrageergebnis einzubeziehen möchte.

Die Ein-Wort-Anfragen können durch die Verwendung von Wildcards * (für beliebige Zeichenketten) und *?* (für genau ein Zeichen) erweitert werden. Dadurch werden z. B. die eRQL-Anfragen *b*dge* und *bri??e* möglich.

Ein-Wort-Anfragen können durch Funktionen begrenzt werden. Eine Begrenzung auf Ressourcen mit URI-Referenz ist durch *res(Ein-Wort-Anfragen)* möglich. Durch die Funktionen*: subj(Ein-Wort-Anfragen)*, *pred(Ein-Wort-Anfragen)* und *obj(Ein-Wort-Anfragen)* kann auf die entsprechenden Komponenten einer Aussage begrenzt werden. Die genannten Funktionen werden daher als *Begrenzungsfunktionen* bezeichnet.

Eine weitere Begrenzungsfunktion wird durch Hochkommata aktiv. Hierdurch wird auf die Menge der Literale begrenzt, wobei ein Vergleich mit dem Wert des Literals vorgenommen wird. Dadurch werden auch Leerzeichen in Suchbegriffen zugelassen. Als Beispiel liefert „*Pablo Picasso*“ die Aussagen, die ein Literal als Objekt besitzen, dessen Wert die gegebene Zeichenkette enthält. Man beachte, dass eine Anfrage der Form „*Ein-Wort-Anfrage*“ immer eine Teilmenge der Anfrage *obj(Ein-Wort-Anfrage)* darstellt, was wiederum eine Teilmenge der Anfrage *Ein-Wort-Anfrage* ist. Dies gilt, da Literale in RDF nur als Objekte auftreten können[64].

Einzelne eRQL-Anfragen können durch die Booleschen Operatoren *AND* und *OR* zu neuen eRQL-Anfragen miteinander verknüpft werden, wobei die Ausführungsreihenfolge durch runde Klammern „(“ und „)“ festgelegt werden kann. Wird dies nicht getan, hat die UND-Verknüpfung Vorrang. Die Interpretation der UND-Verknüpfung ist dabei mit abhängig von der Anfrage und ihrem Ergebnistypen. Bei der Interpretation wird zugunsten interpretierbarer Ergebnisse nicht immer die Dijunktive Normalform erzwungen. In 5.2.2.2 wird die Interpretation der UND-Verknüpfung weiter vertieft und mit Beispielen beschrieben.

Durch die Begrenzungsfunktionen, die Booleschen Operatoren und die Klammerung wird die Mächtigkeit der Ein-Wort-Anfragen in eRQL deutlich erhöht. Weitere Funktionalitäten kommen durch die eRQL-Modi in Abschnitt 5.2.1.1 und die eRQL-Schemafunktionen in Abschnitt 5.2.1.2 hinzu.

Würde eRQL in einer Internetsuchmaschine eingesetzt werden, sollte ähnlich wie bei Google eine erweiterte Suchanfrage, die über die Nutzung von Ein-Wort-Anfragen hinausgeht, durch entsprechende Benutzungsoberflächen unterstützt werden. Die Benutzungsoberfläche wie sie zurzeit existiert und in 5.2.1.3 beschrieben wird, reicht hierfür nicht aus. Sie dient nur zum Testen von eRQL an sich. Der Grundansatz von eRQL ist und bleibt aber, möglichst einfach Anfragen zu unterstützten, erfahrenen Anwendern aber Spielraum für erweiterte und komplexere Anfragen zur Verfügung zu stellen.

5.2.1.1 eRQL-Modi

Für die Unterstützung des internen bzw. externen Kontextes werden unterschiedliche Modi in eRQL angeboten. Als Beispiel für die unterschiedliche Wirkung seien die in der folgenden Abbildung 36 gegeben Daten aus zwei Quellen gegeben.

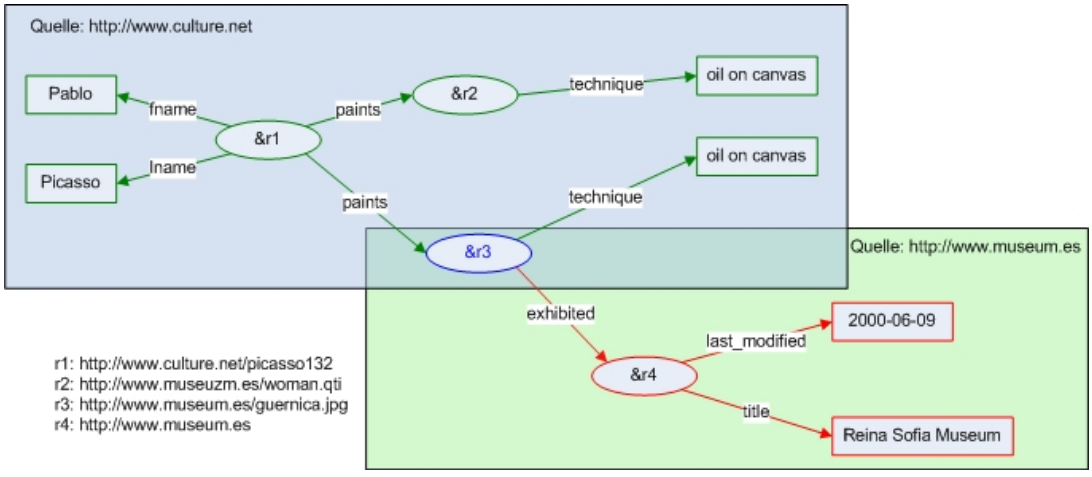


Abbildung 36 Beispieldaten aus zwei Quellen zur Demonstration der Wirkungsweise unterschiedlicher Modi von eRQL.

[64] Es gibt Versuche diese Einschränkung aufzuheben. TriX z. B. erlaubt auch die Nutzung von Literale als Subjekt.

1. Im ***Statement-Modus*** werden nur die direkten Treffer ohne jegliche Umgebung zurückgegeben. Der Statement-Modus wird durch das Einschließen einer Anfrage in eckige Klammern *[Anfrage]* aktiviert. Der *Statement-Modus* entspricht am ehesten der Rückgabe von Anfragesprachen wie RQL oder RDQL.

2. Beim ***PointOfInterest-Modus* (*POI-Modus*)** werden zu den direkten Treffern auch deren Umgebungen zurückgegeben. Das Ergebnis besteht somit aus mehreren zusammenhängenden Teilgrafen. Der POI-Modus ist die Grundeinstellung von eRQL. Der Radius der Umgebung kann dabei vergrößert oder auch auf Null gesetzt werden (*Statement-Modus*). Per Vorgabe verwendet eRQL den *POI-Modus* mit Radius eins. Dadurch werden zusätzlich zu den gefundenen RDF-Aussagen alle unmittelbar angrenzenden RDF-Aussagen zurückgegeben. Der *POI-Modus* kann auch explizit aktiviert werden, indem die eRQL-Anfrage in geschweifte Klammern *{Anfrage}* oder um ein vorangestelltes Schlangenzeichen zu *~Anfrage* erweitert wird. Bei jeder Erweiterung des Radius um eins wird die Umgebung um ihre Umgebung erweitert. Die Anfrage *{{{Anfrage}}}* fügt daher zu allen gefundenen RDF-Aussagen die Aussagen der Umgebung bis zu einer Distanz von drei dem Ergebnis hinzu. Beim Schlangenzeichen wird die Entfernung der Grundeinstellung dazugerechnet. Damit ergibt sich für *~Anfrage* eine Entfernung von zwei. Das Beispiel in der folgenden Abbildung 37 macht dies deutlich:

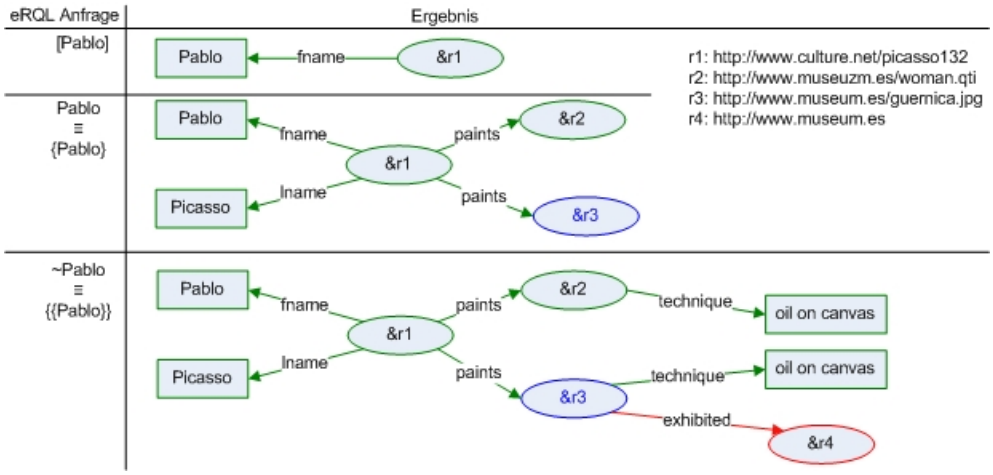


Abbildung 37 Ergebnisse für die Ein-Wort-Anfrage *Pablo* im Statement- und POI-Modus mit unterschiedlichen Entfernungen. Als Datenbasis gelten die Daten aus Abbildung 36.

3. Beim ***Dokumenten-Modus*** kann eine Anfrage auf eine Menge von Dokumenten beschränkt oder aber umgekehrt eine Menge von Dokumenten explizit ausgeklammert werden. Die Anfrage besteht dabei aus drei Komponenten, die durch Semikolons getrennt und in spitze Klammern gesetzt werden. Die Syntax lautet: *<Anfrage; QM; EXIN>*. Nach der *Anfrage* an sich wird in *QM* eine Menge von Quellen definiert. Einzelne Quellen werden innerhalb von *QM* durch mindestens ein Leerzeichen und einem optionalen Komma getrennt. Als Quellangabe kann entweder die URL der Quelle oder deren interne ID angegeben werden. Bei URL Angaben können ebenfalls die Wildcards * und ? verwendet werden[65], sodass ganze Gruppen von URLs gewählt werden können. Als Beispiel würde die Anfrage *<Anfrage; http://www.uni-frankfurt.de*; 1>* auf alle eingetragenen Quellen der Domäne der Universität Frankfurt zugreifen. Die Nutzung von Wildcards kann jedoch auch deaktiviert werden, da die Zeichen * und ? innerhalb von URLs vorkommen können und dann nicht als Wildcards interpretiert werden sollten. Die internen IDs können vorherigen Ergebnissen (explorativ Suchen) oder der Datenbank direkt entnommen werden. Hierzu bildet die

[65] Die Verwendung der Wildcard innerhalb der Protokollangabe der URL ist dabei nicht zulässig.

Benutzungsoberfläche von eRQL Gelegenheit (siehe Abbildung 39). Die dritte Komponente *EXIN* kann die Werte 1 oder 0 annehmen. Sie spezifiziert, ob die Anfrage nur auf die in *QM* angegebenen Quellen (1) oder aber auf allen außer diesen (0), ausgeführt wird. In der folgenden Abbildung 38 werden zwei Beispiele für Anfragen im Dokumenten-Modus gegeben:

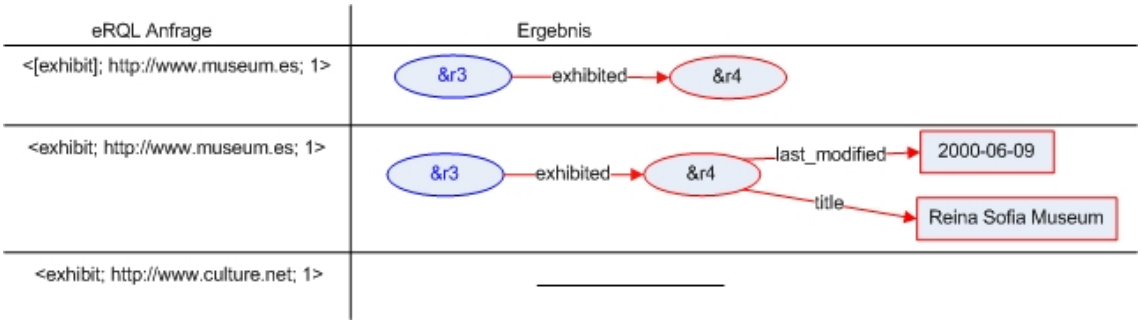


Abbildung 38 Ergebnisse für Beispielanfragen im Dokumenten-Modus. Als Datenbasis gelten die Daten aus Abbildung 36. Man beachte, dass die Eigenschaft *exhibited* auch gefunden wird, obwohl nach *exhibit* gefragt wurde.

Die drei eRQL-Modi können ineinander verschachtelt werden. In einigen Fällen kann man die Verschachtelung der Modi vereinfachen, z. B. kann *[[query]]* in *[query]* umgeformt werden, da ein mehrfach angewendeter *Dokumenten-Modus* das Ergebnis der Anfrage nicht weiter verändert. Auch können die verschiedenen Modi durch die Booleschen Operatoren *AND* und *OR* verknüpft bzw. mit Klammern deren Abarbeitungsreihenfolge beeinflusst werden. Hierauf wird in Unterabschnitt 5.2.2 weiter eingegangen.

5.2.1.2 eRQL-Schemafunktionen

Weiterhin unterstützt eRQL Schemafunktionen, die sich an die Funktionen von RQL anlehnen. Die folgende Auflistung ist aufgeteilt nach *generellen Schemafunktionen*, die keine Eingabeparameter benötigen und *Schemafunktionen auf Ressourcen*, die als Eingabeparameter eine Ein-Wort-Anfrage oder eine URI erwarten. Die Schemafunktionen auf Ressourcen können weiter aufgeteilt werden in *Schemafunktionen auf Klassen* und *Schemafunktionen auf Eigenschaften*. Für die einzelnen Funktionen werden jeweils auch Kurzschreibweisen angeboten, die bei den Auflistungen in eckigen Klammern angeführt werden. Schemafunktionen werden generell im Statement-Modus ausgeführt, es wird also keine Umgebung hinzugefügt – auch dann nicht, wenn dies in der Anfrage spezifiziert sein sollte. Sie können jedoch im Dokumenten-Modus ausgeführt werden, wodurch man die Funktionen auf ausgewählte Quellen begrenzen oder bestimmte Quellen ausschließen kann.

Die Ergebnisstruktur für Schemafunktionen besteht nicht aus Teilgrafen sondern entspricht einer Auflistung. Dies entspricht den natürlichen Erwartungen eines Anfragenden, der z. B. die Instanzen einer Klasse anfragt. Genauer wird auf die unterschiedlichen Ergebnisstrukturen in 5.2.2 eingegangen. Abweichend hiervon gehen einige Funktionen über die einfache Auflistung hinaus. So listet die Funktion *container()* nicht nur alle RDF-Container auf, sondern gibt gleichzeitig auch deren Elemente mit an. Dies mag anfangs ein wenig irritierend sein, jedoch ist die URI eines Containers wenig aussagekräftig. Die Angabe der Elemente eines Containers jedoch schon eher. Mit dieser Entscheidung soll das Verständnis für die Ergebnisse erhöht werden.

Aus dem gleichen Grund werden bei einigen eRQL-Schemafunktion Kontextinformationen nicht automatisch mit im Ergebnis abgebildet. Die Angabe der Kontextinformationen würde zur Folge haben, dass Ressourcen mehrfach dargestellt werden müssten, jeweils mit anderen Kontextknoten. Wird bei einer solchen Anfrage dennoch die Kontextinformation benötigt, so kann dies erzwungen werden, indem die Funktion in den Dokumenten-Modus eingebettet wird.

Um dabei aber dennoch auf allen Quellen suchen zu können, kann die Null als Quellmenge eingetragen werden. Die Anfrage hat damit die Form: *<Anfrage;0;0>*. Ein Beispiel hierfür ist die Funktion *classes()*. Sie zählt alle gespeicherten Klassen auf. Die Anfrage *<classes();0;0>* zählt ebenfalls alle Klassen auf, jedoch zusammen mit ihren Kontextknoten (wodurch es zu Mehrfachnennungen von Klassen kommen kann).

Generelle Schemafunktionen:

- *classes(), c(), C()* – listet alle Klassen auf. Durch Einbettung in den Dokumenten-Modus kann die Angabe der Kontextknoten erzwungen werden.
- *container(), con(), CON()* – listet alle Container auf und gibt deren Elemente jeweils zusammen mit ihren Kontextknoten an.
- *literals(), l(), L()* – listet alle Literale auf. Da Literale bezogen auf ihren Kontextknoten eindeutig sind, wird bei dieser Funktion auch dann der Kontextknoten mit angegeben, wenn die Funktion nicht in den Dokumenten-Modus eingebettet ist.
- *properties(), p(), P()* – listet alle Eigenschaften auf. Durch Einbettung in den Dokumenten-Modus kann die Angabe der Kontextknoten erzwungen werden.
- *reifiedStatements(), rs(), RS()* – listet alle vergegenständlichten Aussagen auf. Hierbei wird nicht nur die URI der vergegenständlichten Aussage gegeben, sondern die entsprechende Aussage selbst auch, wobei der Kontextknoten immer mit angegeben wird.
- *triples(), t(), T()* – listet alle Aussagen zusammen mit ihren Kontextknoten auf. Das Ergebnis stellt einen Grafen dar, der jedoch, anders als die Ergebnisse im POI-Modus, nicht zusammenhängend sein muss.

Bei den Schemafunktionen auf Klassen bzw. Eigenschaften kann eine URI oder eine Ein-Wort-Anfrage als Parameter übergeben werden. Die Ein-Wort-Anfrage ist dabei nicht immer eindeutig. Das Ergebnis zu diesen Anfragen beinhaltet daher eine Liste der Klassen bzw. Eigenschaften, die zur Ein-Wort-Anfrage passen, mit den jeweils dazugehörigen Ergebnissen der Funktion. Bei diesen werden die Kontextknoten im Ergebnis mit angegeben. Für Klassen und Eigenschaften kann weiterhin deren URI immer in einen Namen und in den Namensraum aufgeteilt werden, was allgemein für Ressourcen nicht gilt. Daher wird hier, anders als z. B. bei den Begrenzungsfunktionen, nur mit dem Namen verglichen. Wird eine URI als Parameter übergeben, wird diese am Aufbau, genauer der Schemadefinition, erkannt.

Schemafunktionen auf Klassen:

- *directInstancesOf(URI | Ein-Wort-Anfrage), di(URI | Ein-Wort-Anfrage), dI(URI | Ein-Wort-Anfrage)* – gibt alle direkten Elemente der Klasse bzw. Klassen, deren Namen dem gegebenen Parameter entsprechen wieder. Implizite Instanzen von Unterklassen werden nicht mit ausgegeben.
- *instancesOf(URI | Ein-Wort-Anfrage), i(URI | Ein-Wort-Anfrage), I(URI | Ein-Wort-Anfrage)* – gibt alle Elemente der Klasse bzw. Klassen, deren Namen dem gegebenen Parameter entsprechen wieder, inklusive impliziter Instanzen von Unterklassen.
- *subClassOf(URI | Ein-Wort-Anfrage), subc(URI | Ein-Wort-Anfrage), subC(URI | Ein-Wort-Anfrage)* – gibt alle Unterklassen der Klasse bzw. Klassen, deren Namen dem gegebenen Parameter entsprechen, wieder.
- *superClassOf(URI | Ein-Wort-Anfrage), superc(URI | Ein-Wort-Anfrage), superC(URI | Ein-Wort-Anfrage)* – gibt alle Oberklassen der Klasse bzw. Klassen, deren Namen dem gegebenen Parameter entsprechen.

Schemafunktionen auf Eigenschaften:

- *domain(URI | Ein-Wort-Anfrage), d(URI | Ein-Wort-Anfrage), D(URI | Ein-Wort-Anfrage)* – gibt die Klassen des Definitionsbereichs der Eigenschaft bzw. Eigenschaften, deren Namen dem gegebenen Parameter entsprechen, wieder.
- *range(URI | Ein-Wort-Anfrage), r(URI | Ein-Wort-Anfrage), R(URI | Ein-Wort-Anfrage)* – gibt die Klassen des Wertebereichs der Eigenschaft bzw. Eigenschaften, deren Namen dem gegebenen Parameter entsprechen, wieder.
- *subPropertyOf(URI | Ein-Wort-Anfrage), subp(URI | Ein-Wort-Anfrage), subP(URI | Ein-Wort-Anfrage)* – gibt alle Untereigenschaften der Eigenschaft bzw. Eigenschaften, deren Namen dem gegebenen Parameter entsprechen, wieder.
- *superPropertyOf(URI | Ein-Wort-Anfrage), superp(URI | Ein-Wort-Anfrage), superP(URI | Ein-Wort-Anfrage)* – gibt alle Obereigenschaften der Eigenschaft bzw. Eigenschaften, deren Namen dem gegebenen Parameter entsprechen, wieder.

Es sind weitere Funktionen denkbar, um eRQL zu erweitern. Für Container wären Funktionen denkbar, die auf die unterschiedlichen Containertypen beschränken könnten. Auch ist eine Auflistung der Container ohne Anzeige ihrer Elemente denkbar. Welche Erweiterungen sinnvoll sind, hängt von der jeweiligen Anwendung ab. Die hier aufgeführten Funktionen decken jedoch sicherlich die wichtigsten Bereiche ab.

5.2.1.3 Die Benutzungsoberfläche von eRQL

Die in RDF-S3 integrierte Benutzungsoberfläche für eRQL ist in Abbildung 39 dargestellt. Sie dient zum ersten Kennenlernen und Testen der Anfragesprache eRQL.

Die Einstellungen für die Datenbankverbindung können im oberen Bereich getätigt werden. Dieser hat den gleichen Aufbau wie in der Benutzungsoberfläche zum Speichern von RDF-Dateien unter RDF-S3 (siehe Abbildung 18). Im mittleren Bereich kann auf der linken Seite die gewünschte eRQL-Anfrage eingegeben werden. Ausgeführte Befehle werden in der darunter liegenden Historie eingetragen und können so einfach wiederholt werden. Durch Drücken der *Refresh*-Taste weiter rechts, kann eine aktuelle Liste der zurzeit in der Datenbank gespeicherten Quellen angezeigt werden. Neben den Adressen (URLs) der Quellen werden dort die internen IDs angegeben, die für Anfragen im Dokumenten-Modus genutzt werden können.

In den darunter liegenden Optionsschaltfeldern kann ausgewählt werden, ob für die gesamte Anfrage Groß- und Kleinschreibung berücksichtigt werden soll (*Case Sensitive*). Auch kann spezifiziert werden, ob die Zeichen * und ? als Wildcards zu behandeln sind oder nicht (z. B. falls diese innerhalb von Quell-Listen des Dokumenten-Modus auftauchen).

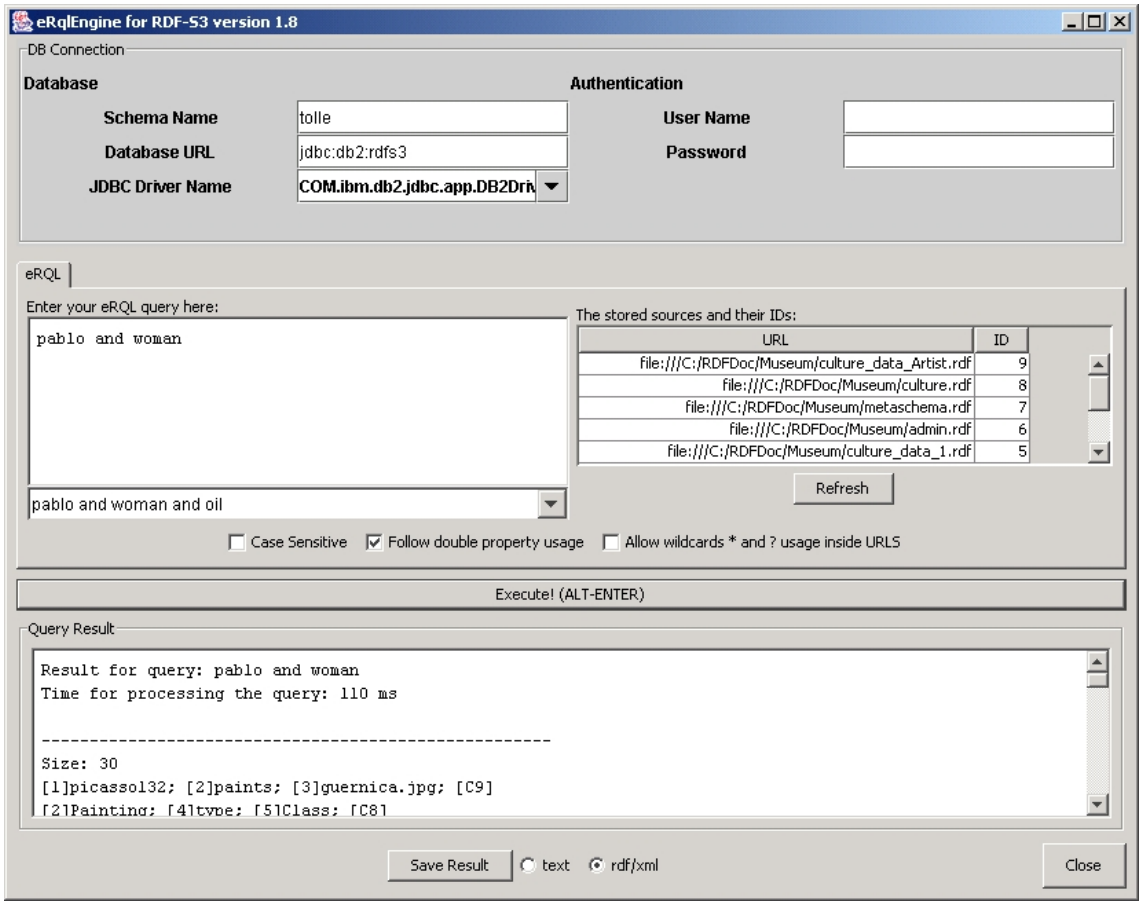


Abbildung 39 Bildschirmfoto der in RDF-S3 integrierten Benutzungsoberfläche für eRQL.

Die weitere Option *Follow double property usage* bewirkt, dass im POI-Modus eine Aussage nur dann in die Ergebnismenge integriert wird, wenn die enthaltene Eigenschaft nicht der Eigenschaft entspricht, welche um eins näher an den entsprechenden Treffer liegt. Diese Option bewirkt eine Reduktion der Ergebnismenge im POI-Modus. Hilfreich ist dies, falls eine Klasse im Radius des Treffermenge liegt. In diesem Fall würden alle direkten Instanzen der Klasse in die Ergebnismenge integriert werden, wodurch die Ergebnismenge sehr groß und damit unübersichtlich werden kann. Ein Beispiel hierfür ist in Abbildung 40 dargestellt. Die rot umrandeten Aussagen würden je nach Einstellung für diese Option für die Anfrage *~Pablo* zum Ergebnis gehören oder nicht. Man beachte, dass sich die Auswirkung nicht auf bestimmte Eigenschaften, wie *rdf:type*, beschränkt.

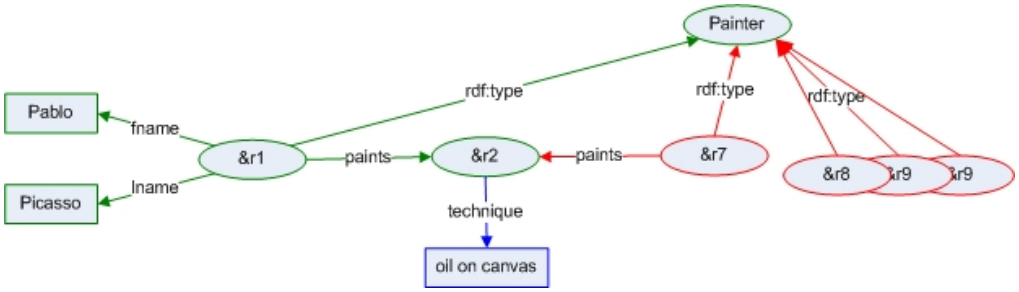


Abbildung 40 Beispielgraf für die Wirkungsweise der Option *Follow double property usage*. Die Anfrage *Pablo* gibt die grün gezeichneten Aussagen wieder. Erweitert man die POI-Umgebung über die Anfrage *~Pablo*, wird die blau gezeichnete Aussage hinzugefügt. Ist die *Follow double property usage* aktiviert, werden auch die rot gezeichneten Aussagen mit zurückgegeben.

Die Ausgabe der Ergebnisse erfolgt im unteren *Query Result* Textfeld. Dort wird nach der Wiederholung der Anfrage die benötigte Antwortzeit aufgeführt. Anschließend folgt die Anzahl der Teilergebnisse, z. B. Teilgrafen, die durch die POI-Bildung entstanden sind. Diese werden getrennt durch gestrichelte Linien aufgelistet. Um die eigentlichen Treffer innerhalb von Teilgrafen zu erkennen, werden sie mit dem Zusatz *„direct Hit“* gekennzeichnet. Bei der Darstellung werden die Ressourcen URIs wie auch die Kontextknoten abgekürzt. Bei den Kontextknoten stimmt dabei die angegebene Nummer mit der internen ID überein. Die entsprechenden Referenzen sind am Ende des Ergebnisses zu finden. Die Abbildung 41 zeigt beispielhaft das Ergebnis für die Anfrage *<pablo;6;1>*. Hierbei handelt es sich um eine Anfrage im Dokumenten-Modus, wobei nur die Quelle mit der internen ID 6 berücksichtigt und eine Umgebung der Entfernung eins zu den direkten Treffern ausgegeben wird.

```
Result for query: <pablo;6;1>

Time for processing the query: 10 ms
1 Hits found.
-----------------------------------------------------
Size: 5
[1]www.culture.net/picasso132; [2]paints; [3]woman.qti; [C6]
[1]www.culture.net/picasso132; [2]first_name; "Pablo"@en; [C6] - direct Hit
[1]www.culture.net/picasso132; [2]last_name; "Picasso"@en; [C6]
[1]www.culture.net/picasso132; [2]paints; [3]guernica.jpg; [C6]
[1]www.culture.net/picasso132; [4]type; [2]Cubist; [C6]

Namespaces:
[1] file:///C:/RDFDoc/Daten/culture_data1.rdf#
[2] file:///C:/RDFDoc/culture.rdf#
[3] http://www.museum.es/
[4] http://www.w3.org/1999/02/22-rdf-syntax-ns#
Sources:
[C6] file:///C:/RDFDoc/Daten/culture_data1.rdf
entered or updated at: 2005-01-08 17:15:00.251
```

Abbildung 41 Ergebnis zur Anfrage <pablo;6;1>. Hierbei handelt es sich um eine POI-Modus Anfrage, die im Dokumenten-Modus integriert ist. Es wurde nur ein passendes Quadrupel gefunden, welches mit dem Zusatz *„direct Hit“* gekennzeichnet ist. Dieses Quadrupel wurde um seine Umgebung der Entfernung eins erweitert, die vier Quadrupel umfasst.

Schemafunktionen weisen eine andere Ergebnisstruktur auf. Dies wird auch in der textuellen Darstellung widergespiegelt. In Abbildung 42 wird das Ergebnis für die *instancesOf*-Funktion dargestellt. Auch hier werden die verwendeten Namensräume und Quellen am Ende aufgelistet, was jedoch in der Abbildung weggelassen wurde. Nach den allgemeinen Informationen, wie Anfragetext, Ausführungszeit und Ergebnisgröße, werden die eigentlichen Ergebnisse aufgelistet. Dort wird noch einmal die Funktion wiederholt. Zu sehen ist dort die Klasse *[1]Sculpture*, welche die Anfragezeichenkette *sculp* enthält. Für diese werden anschließend die Instanzen aufgelistet.

```
Result for query: <i(sculp);6;1>

Time for processing the query: 30 ms
2 Hits found.
-----------------------------------------------------
instances of [1]Sculpture
--------------- START RESULT LIST -----------------------------
[2]descent.jpg; [C6]
[3]crucifixion.jpg; [C6]
[2]theslave.jpg; [C6]
--------------- END RESULT LIST -------------------------------
….
```

Abbildung 42 Anfang des Ergebnisses zur Anfrage <i(sculp);6;1>. Die Anfrage gibt alle Instanzen von Klassen wieder, welche die Zeichenkette „sculp“ im Namen enthalten. Hierbei wird nur die Quelle mit der internen ID von 6 berücksichtigt.

Die Textdarstellung des Ergebnisses kann über die Taste *Save Result* in einer Datei gespeichert werden. Alternativ hierzu können Ergebnisse in einem RDF/XML-Format gespeichert werden, wofür ein eigenes Vokabular, welches auch im Anhang unter 8.5 zu finden ist, verwendet wird. Das RDF/XML-Format ermöglicht Anfrageergebnisse, ohne auf die Programmierung Einfluss nehmen zu müssen, in anderen Anwendungen weiterverwenden zu können.

5.2.2 Implementierung, Ergebnistypen und Interpretation des Anfragebaums

In diesem Unterabschnitt wird genauer auf die Interpretation der Anfragen und auf die Implementierung eingegangen. Die Komplexität, die dahinter steckt, sollte den einfachen Benutzern von eRQL verwahrt bleiben. Zusätzliche Informationen können der Java-Dokumentation entnommen werden. Nach einer Übersicht über die Implementierung von eRQL wird genauer auf die Interpretation der einzelnen Modi, der Booleschen Verknüpfungen und die Ergebnistypen eingegangen. Eine zentrale Rolle nimmt die interne Darstellung der Anfrage als *Anfragebaum* ein. Für diesen werden Regeln beschrieben, aufgrund derer er umgeformt werden kann und so kompakter und leichter ausführbar wird.

5.2.2.1 Übersicht

Die Funktionalitäten, die eRQL bietet, werden bereits durch die Anfrageschnittstelle von RDF-S3 optimal unterstützt (siehe Seite 59). Dies entlastet die Implementierung von eRQL, in der die Ergebnistypen und Klassen aus RDF-S3 weiterverwendet werden. Die verbleibenden Aufgaben der Implementierung von eRQL sind:

- Bereitstellung einer grafischen Benutzungsoberfläche für eine einfache Nutzung.
- Prüfung der syntaktischen Korrektheit der eingegebenen Anfrage.
- Umwandlung der eingegebenen Anfrage in eine interne Darstellung.
- Zusammenfügen der einzelnen Zwischenergebnisse entsprechend der Booleschen Verknüpfungen.
- Darstellung und Speicherung der Ergebnisse.

Die Aufgabenstellungen werden in der Paketstruktur der Implementierung widergespiegelt (Abbildung 43).

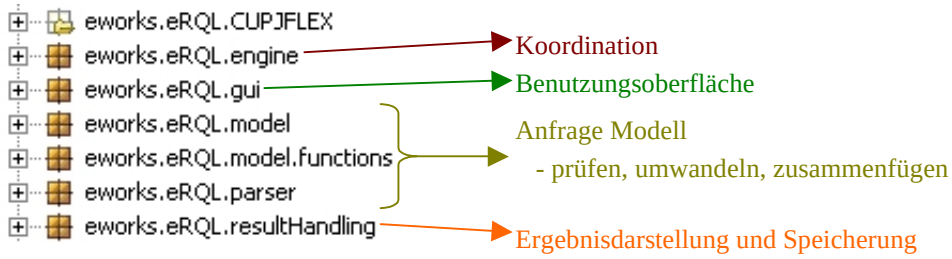


Abbildung 43 Die Pakete der eRQL-Implementierung.

Hierbei enthält das Paket *eworks.eRQL.engine* nur die Klasse *eRQLEngine*, welche die Rolle eines Koordinators übernimmt. Sie erhält die eRQL-Anfrage in Textform von der Benutzungsoberfläche. Die Anfrage wird syntaktisch geprüft und ein entsprechendes internes Modell aufgebaut, auf welches später näher eingegangen wird. Für die Syntax-Überprüfung baut eRQL auf einen lexikalischen Scanner und einen Übersetzer auf, die mittels der Generatoren JFLEX[66] und CUP[67] erzeugt wurden. Im Paket *eworks.eRQL.CUPJFLEX* sind die

[66] JFLEX ist ein lexikalischem Scannergenerator und ist online unter http://www.jflex.de erhältlich.

[67] Constructor of Useful Parsers (CUP) ist ein Übersetzergenerator für Java. Er generiert Lookahead-LR-Parser (LALR-Parser) und kann online unter http://www.cs.princeton.edu/~appel/modern/java/CUP/ bezogen werden.

Dateien enthalten, welche als Eingabe für die Generatoren dienten. Der daraus resultierende Scanner sowie der Übersetzer sind im Paket *eworks.eRQL.parser* zu finden.

Nach dem Aufbau des internen Anfragebaums stößt der Koordinator die Anfrageumformung und deren Ausführung an. Das Ergebnis wird schließlich vom Koordinator an eine entsprechende Klasse aus dem Paket *eworks.eRQL.resultHandling* zur Ausgabe weitergeleitet. Der Ablauf ist in Abbildung 44 noch einmal grafisch dargestellt.

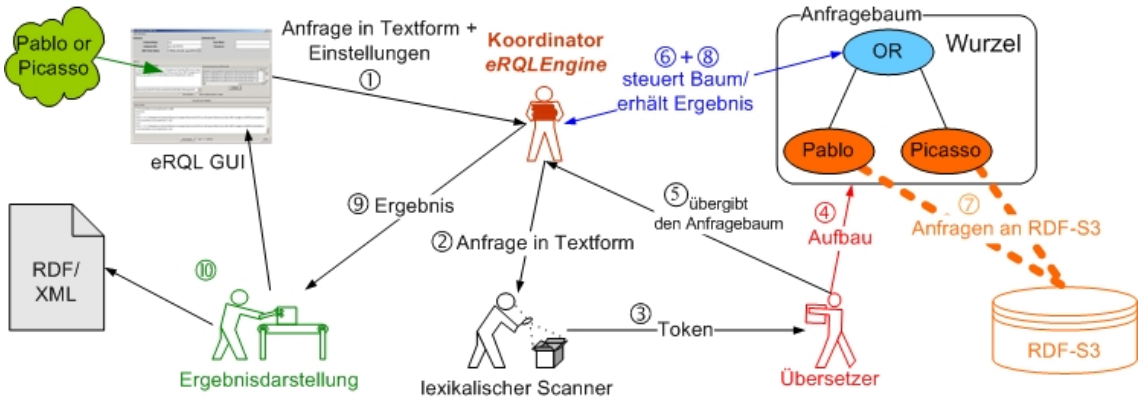


Abbildung 44 Grafische Darstellung des Ablaufs und der Koordinierung der Anfragebearbeitung in eRQL. Die eingekreisten Zahlen 1-10 geben dabei die Reihenfolge an. Die wichtigeren Komponenten und Aufgaben sind farbig hervorgehoben.

5.2.2.2 Ergebnistypen und Boolesche Verknüpfungen

eRQL bietet verschiedene Anfragemöglichkeiten, die auch Auswirkungen auf die Struktur des Ergebnisses haben. So ist die Struktur des Ergebnisses von Ein-Wort-Anfragen anders als von Schema Anfragen. eRQL nutzt die Anfrageschnittstelle von RDF-S3 und verwendet auch dessen Ergebnistypen, die in Abbildung 45 rechts als Klassenbaum dargestellt sind. Dabei bildet die abstakte Klasse *Result* die Wurzel, von der die eigentlichen Ergebnistypen *POI, POIResult, SchemaResultList* und *SchemaResultMap* abgeleitet sind. Die weitere abstrakte Klasse *SchemaResult* dient dabei zur besseren Strukturierung.

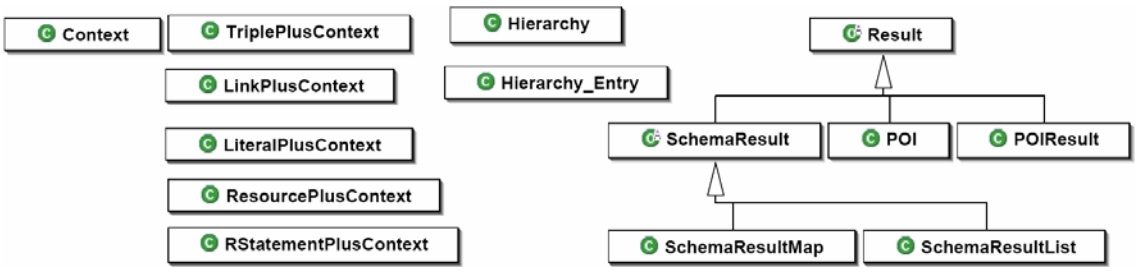


Abbildung 45 UML-Klassendiagramm des Pakets de.jwg.dbis.rdf.rdfs3.access.model.

Eine Instanz der Klasse *POI* repräsentiert einen Grafen. Vornehmlich dient dies zum Darstellen der im POI-Modus erzeugten Umgebungen um die direkten Treffer von Ein-Wort-Anfragen. Der Graf besteht aus Instanzen vom Typ *TriplePlusContext*, also aus Quadrupel. Da es für eine Anfrage mehrere direkte Treffer geben kann, werden von der *QueryEngine* in RDF-S3 die Ergebnisse vom Typ POI in einem *POIResult* zusammengefasst zurückgegeben. Beim Einfügen eines POIs in ein POIResult wird geprüft, ob dieser Graf bereits im POIResult vorhanden, von anderen überdeckt wird oder er selbst andere überdeckt. Ist dies der Fall, werden die Duplikate oder überdeckten POIs aus dem POIResult entfernt.

Ergebnisse vom Typ *SchemaResultList* werden für generelle Schemafunktionen verwendet. Man kann sie sich als Mengen von Ressourcen vorgestellen. Die Anfrage nach allen gespeicherten Klassen *classes()* wäre ein Beispiel hierfür. Die Klasse *SchemaResultMap* ist für

Schemafunktionen auf Klassen bzw. Eigenschaften zuständig. Sie entspricht einer Menge von Klassen bzw. Eigenschaften, die zum Parameter der Anfrage passen. Jedem dieser so genannten Schlüsseleinträge wird ein Ergebnis vom Typ SchemaResultList zugeordnet, welches das entsprechende Ergebnis der Funktion auf den Schlüsseleintrag hat. Die Anfrage *instancesOf(sculptor)* würde die Elemente aller Klassen aufzählen, welche die Zeichenkette ‚sculptor' im Namen enthalten.

Innerhalb von eRQL wird zusätzlich mit Ergebnismengen (hier vom Typ *java.util.ArrayList*) gearbeitet. Die Ergebnismengen werden genutzt, um die für die Booleschen Verknüpfungen nötige Weiterverarbeitung zu ermöglichen. Die ODER-Verknüpfung (OR) wird dabei durch einfaches Vereinigen der Teilergebnismengen realisiert. Dies kann dazu führen, dass innerhalb einer Ergebnismenge unterschiedliche Ergebnistypen auftreten. Ein Beispiel für eine Anfrage, bei der dies der Fall ist, ist: *Pablo OR classes()*. Die Ergebnismenge würde sowohl das POIResult der Anfrage *Pablo*, als auch die SchemaResultList aller Klassen enthalten.

Für die UND-Verknüpfung (AND), muss definiert werden, wie diese bei der Kombination der Ergebnistypen zu interpretieren ist und wie das Ergebnis einer solchen Kombination aussehen soll. In der folgenden Auflistung werden die einzelnen Fälle behandelt, wobei jeweils ein Beispiel dienen soll, die gewählte Entscheidung zu begründen. Hierbei gilt generell die Kommutativität der UND-Verknüpfung.

- **SchemaResultList AND SchemaResultList** – ein Beispiel hierfür ist die Anfrage *<classes();1;1> AND <classes();2;1>*. Diese würde interpretiert werden als: *Die Klassen, die sowohl in der Quelle mit der ID 1 als auch in der Quelle mit der ID 2 genutzt werden.* Das Ergebnis sollte daher der Durchschnitt beider Mengen sein. Hierbei ist zu beachten, dass unterschiedliche Kontextknoten zu ignorieren sind. Für unsere Beispielanfrage wäre sonst der Durchschnitt immer leer! Allgemein ist das Ergebnis in diesem Fall vom Typ SchemaResultList, die den Durchschnitt ohne Berücksichtigung des Kontextknotens repräsentiert.
- **SchemaResultMap AND SchemaResultList** – die Anfrage *superClassOf(Sculp) AND <classes();6;1>* würde interpretiert werden als: *Nenne alle Oberklassen der Klassen, welche die Zeichenkette ‚Sculp' im Namen besitzen, wobei die Oberklassen in der Quelle 6 enthalten sein müssen*. Es ist aber auch eine wichtige Information, wenn zu einer bestimmten Klasse keine Oberklasse mit der gegebenen Bedingung zu finden ist. Das Ergebnis sollte daher vom Typ SchemaResultMap sein und die gleichen Schlüsseleinträge enthalten, wie die gegebene SchemaResultMap. Die jeweils dazugehörigen Mengen werden durch die UND-Verknüpfung der SchemaResultListe des Schlüsseleintrages mit der eingegebenen SchemaResultListe gebildet. Die folgende Abbildung 46 visualisiert das gegebene Beispiel.

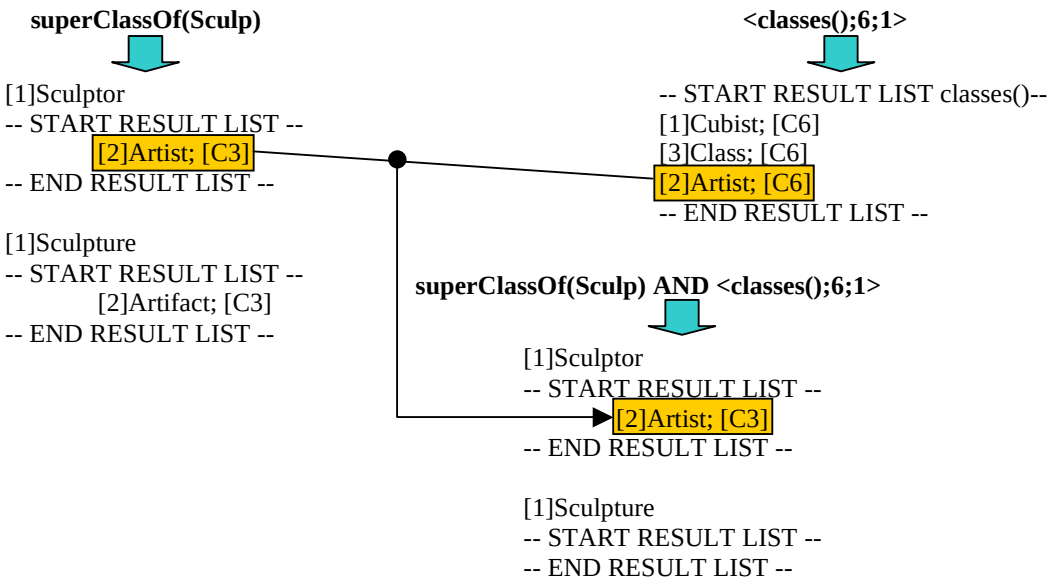


Abbildung 46 Beispiel für die UND-Verknüpfung einer SchemaResultMap mit einer SchemaResultList.

Am Beispiel in Abbildung 46 ist zu erkennen, dass im Ergebnis nur der Kontextknoten *C3* für *[2]Artist* aufgeführt ist. Hier wurde entschieden auf die doppelte Aufführung der gleichen Ressource bei unterschiedlichen Kontextknoten zu verzichten, um die Ergebnisse übersichtlicher zu halten. Als Alternative eine Liste von Kontextknoten anzuhängen wäre denkbar, würde jedoch weitere Verknüpfungen des Ergebnisses mit anderen erschweren.

- **SchemaResultMap AND SchemaResultMap** – ein Beispiel für diesen Fall bildet die Anfrage *instancesOf(Artist) AND instancesOf(Kuenstler).* In diesem Beispiel werden die Instanzenmengen der entsprechenden Klassen verglichen. Wichtig ist zu beachten, dass hierbei jeweils Mengen von Klassen mit ihren Instanzen verglichen werden. Die Zeichenketten ‚Artist' bzw. ‚Kuenstler' können schließlich in mehreren Namen enthalten sein. Als Ergebnis wird eine SchemaResultMap erstellt, die als Schlüsseleinträge alle Kombinationen der Schlüsseleinträge der eingegebenen SchemaResultMap enthält. Die jeweiligen Werte für die Eintragungen entsprechen der UND-Verknüpfung der dazugehörigen SchemaResultListen. In Abbildung 47 wird dies für das gegebene Beispiel grafisch dargestellt.

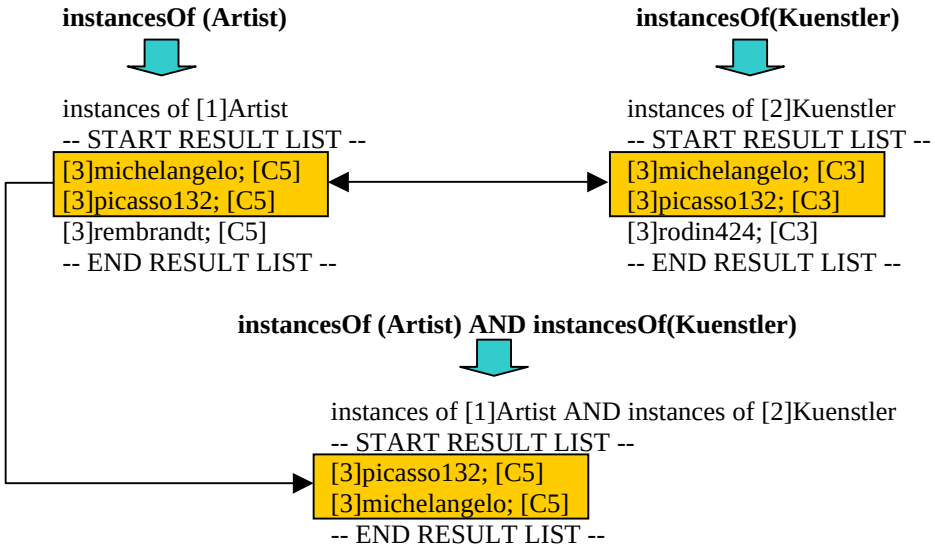


Abbildung 47 Beispiel für die UND-Verknüpfung einer SchemaResultMap mit einer SchemaResultMap.

- **SchemaResultList AND POIResult** – ein Beispiel hierfür ist die Anfrage: *properties() AND ~~Pablo*. Interpretiert werden würde diese Anfrage als: *Nenne die Eigenschaften, die im Grafen um die Aussagen mit der Zeichenkette ‚Pablo' in einer maximalen Entfernung von drei vorkommen.* Dieses wird realisiert durch ein Ergebnis vom Typ SchemaResultMap. In diesem wird für jeden Grafen des POIResults ein Schlüsseleintrag in der SchemaResultMap erzeugt. Als Wert wird diesem eine SchemaResultListe zugeordnet. Sie besteht aus den Elementen, die sowohl in der SchemaResultList der Eingabe, als auch als Subjekt, Prädikat oder Objekt des entsprechenden Grafen enthalten sind. Dies entspricht der Ausführung der Anfrage auf den einzelnen Grafen des POIResult. Die einzelnen Ergebnisse werden dabei in der SchemaResultMap als Gesamtergebnis eingetragen.

- **SchemaResultMap AND POIResult** – als Beispiel sei die Anfrage: *superClassOf(Sculp) AND Pablo* genannt. Interpretiert werden würde diese Anfrage als: *Suche die Oberklassen der Klassen, welche im Namen die Zeichenkette ‚Sculp' aufweisen und in der direkten Umgebung zu Aussagen mit der Zeichenkette ‚Pablo' enthalten sind.* Das Ergebnis ist vom Typ SchemaResultMap. Die Schlüsseleinträge des Ergebnisses werden durch den Durchschnitt der Schlüsseleinträge der SchemaResultMap und den einzelnen POIs des POIResults gebildet. Unter Durchschnitt ist hier zu verstehen, dass die Schlüsseleinträge als Subjekt, Prädikat oder Objekt im POI enthalten sein müssen. Die SchemaResultList-Einträge im Ergebnis, werden durch die UND-Verknüpfung der entsprechenden SchemaResultList-Einträge der SchemaResultMap und dem entsprechendem POI aus dem POIResult gebildet. Man kann sich dies vorstellen, als ob die Anfrage, die zur SchemaResultMap geführt hat, auf den Grafen, die durch die POIs des POIResults repräsentiert werden, ausgeführt werden würden.

Bemerkung: Es sind auch andere Interpretationen denkbar. Für die beiden Fälle *SchemaResultList AND POIResult* und *SchemaResultMap AND POIResult* wäre z. B. möglich, den Durchschnitt nicht mit dem gesamten POI-Grafen zu bilden, sondern nur mit den direkten Treffern. Dieser Ansatz wäre jedoch sehr restriktiv. Da ein Ziel von eRQL ist, auch eine gewisse Unschärfe zuzulassen, wurde die Interpretation wie oben gewählt.

- **POIResult AND POIResult** – ein Beispiel hierfür stellt die Anfrage: *Pablo AND Rodin* dar. In den bisherigen Fällen wurde jeweils ein Durchschnitt ermittelt und dieser auch im Ergebnis zurückgegeben. Bei der Durchschnittsbildung wurde jeweils auf die POI-Ebene hinuntergegangen. Würde man wie oben verfahren, müssten die POIs der Eingaben

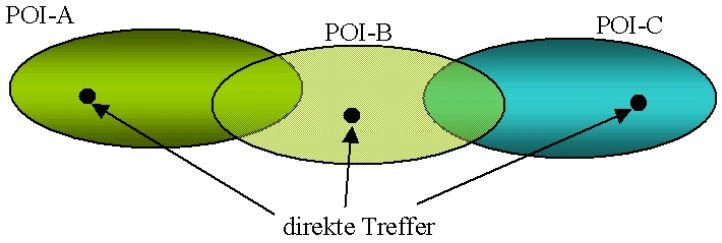
paarweise verglichen und falls Überschneidungen vorliegen, diese zurückgegeben werden. Bei diesem Verfahren könnte der Nutzer jedoch nur schwer den Zusammenhang mit der Anfrage verstehen. Es ist zu erwähnen, dass die POIs zusammenhängende Grafen darstellen. Liegt eine Überschneidung der POIs vor, so folgt daraus, dass ein Pfad zwischen den einzelnen direkten Treffern existieren muss. Diese sind potenziell wichtig für den Nutzer. Es wurde daher entschieden, falls der Durchschnitt zweier POIs nicht leer ist, die beiden POIs im Ergebnis zu vereinen. Hierdurch wird sichergestellt, dass alle Pfade, alle direkten Treffer und der Zusammenhang erhalten bleiben. Auch wenn dadurch diese UND-Verknüpfung nicht der Disjunkiven Normalform entspricht.

Die UND-Verknüpfung zweier POIResults wird wie folgt interpretiert:

Wenn es einen Pfad zwischen direkten Treffern der beiden Anfragen gibt, der kürzer ist als die Summe der POI-Umgebungen, dann wird die Vereinigung der entsprechenden Grafen zurückgegeben, andernfalls ist das Ergebnis leer.

Das Ergebnis wird als POIResult (Menge von POIs) dargestellt. Der Durchschnitt zweier POIs ist genau dann nicht leer, wenn die gleichen Quadrupel in ihnen zu finden sind. Es reicht also nicht aus, wenn sich die Grafen nur bezüglich einzelner Ressourcen überschneiden. Auch ist zu beachten, dass hierbei entschieden wurde, die Kontextknoten mitzuberücksichtigen. Als Erweiterung für eRQL könnte es ermöglicht werden, durch den Benutzer in der Anfrage spezifizieren zu lassen, ob die Berücksichtigung des Kontextknotens gewünscht ist oder nicht.

Eine Hintereinanderausführung mehrerer UND-Verknüpfungen kann als Nacheinanderausführung der einzelnen UND-Verknüpfungen interpretiert werden, da die Ergebnisse dem Durchschnitt entsprechen. Eine Ausnahme bildet der Fall, dass mehrere POIResults UND-Verknüpft werden. In Abbildung 48 wird verdeutlicht, dass die Reihenfolge bei der Betrachtung der POIs hierbei zu unterschiedlichen Ergebnissen führen würde. Mehrere UND-Verknüpfungen von POIResults sollten jedoch unabhängig von der Reihenfolge zu gleichen Ergebnissen führen. Um dies zu gewährleisten, muss die UND-Verknüpfung von POIResults gleichzeitig ausgeführt werden.



POI-A AND POI-B = POI-A ∪ POI-B
POI-B AND POI-C = POI-B ∪ POI-C
POI-A AND POI-C = {}
(POI-A AND POI-C) AND POI-B = {}
(POI-A AND POI-B) AND POI-C = (POI-A ∪ POI-B) ∪ POI-C
POI-A AND (POI-B AND POI-C) = POI-A ∪ (POI-B ∪ POI-C)
POI-A AND POI-B AND POI-C = POI-A ∪ POI-B ∪ POI-C

Abbildung 48 Darstellung für die unterschiedlichen Ergebnisse bezüglich der Hintereinanderausführung von UND-Verknüpfungen von POIs.

Die Vorgehensweise zur Berechnung mehrerer UND-Verknüpfungen von POIResults ist in Abbildung 49 grafisch für drei POIResults dargestellt. Dabei kann man sich die einzelnen POIResults als Spalten und die POIs darin als Knoten vorstellen. Jeder mögliche Weg von den Knoten der ersten Spalte zu den Knoten der letzten Spalte wird untersucht. Die auf einem Weg befindlichen POIs werden, falls sie Überschneidungen aufweisen, zu größeren Grafen vereinigt.

Haben sich am Ende eines Weges alle POIs zu einem Grafen vereinigt, wird dieser dem Ergebnis hinzugefügt, andernfalls nicht.

Die *Anzahl der Wege* kann dabei berechnet werden. Ist J_i die Anzahl der POIs im POIResult *i*, ergibt sich die Anzahl der Wege über das Produkt der J_i. Sollte ein POIResult keine POIs enthalten, also $J_i = 0$ sein, ist das Gesamtergebnis die leere Menge. Es wird daher weiter davon ausgegangen, dass $J_i > 0$ gegeben sei. Die *Länge der Wege* entspricht dabei der Anzahl der POIResults, die an der UND-Verknüpfung beteiligt sind.

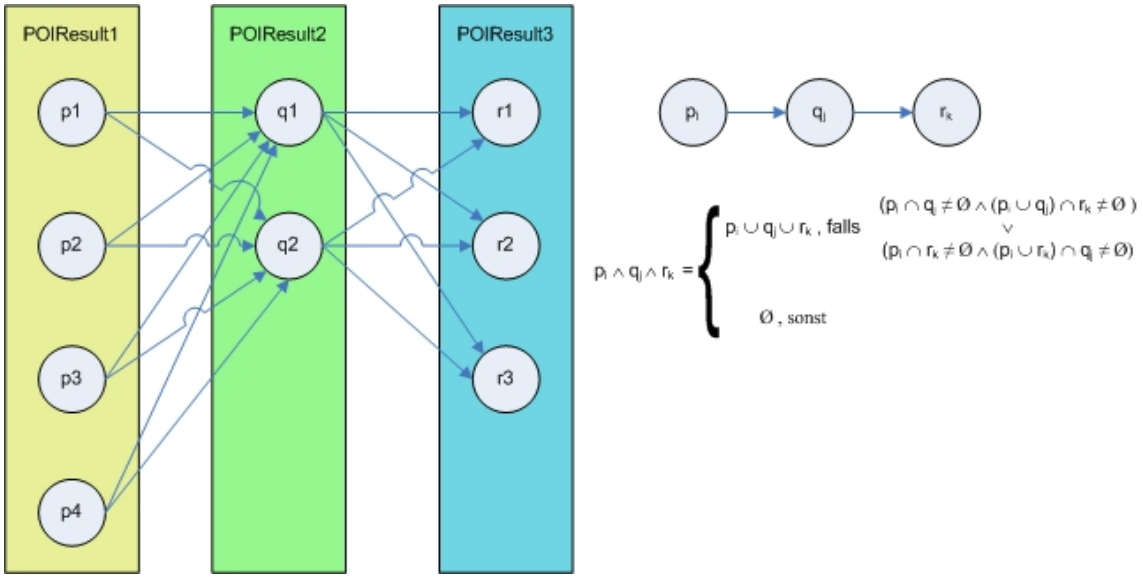


Abbildung 49 Darstellung der UND-Verknüpfung dreier POIResults.

Die Umsetzung der UND-Verknüpfung von *n* POIResults (*POIResult_1, ..., POIResult_n*) wurde folgender Algorithmus verwendet, dessen Java-Implementierung in der Klasse *eworks.eRQL.model.conjunction.java* zu finden ist.

1. Das *POIResult_1* bildet das erste Zwischenergebnis *ZErg_1*. Zwischenergebnisse stellen eine Menge dar, deren Elemente POIs oder Mengen von POIs sind.
2. Die Zwischenergebnisse *ZErg_i* werden jeweils mit *POIResult_i+1* verknüpft, woraus sich das neue Zwischenergebnis *ZErg_i+1* bildet ($1 \leq i \leq n-1$). Für die Verknüpfung *ZErg_i UND POIResult_i+1* wird wie folgt vorgegangen:

Menge ZErg_i+1 := {};
∀E ∈ ZErg_i {
∀poi ∈ POIResult_i+1 {
falls E ein POI ist:
falls E ∩ poi = ∅, füge {E, poi} in Zerg_i+1 ein,
sonst, füge E ∪ poi in ZErg_i+1 ein.
falls E eine Menge von POIs darstellt:
POI poi_z := poi; // zum Sammeln zusammenhängender POIs
Menge HMenge := {};
∀poi_e ∈ E {
falls poi_e ∩ poi = ∅, füge poi_e in HMenge ein,
sonst, poi_z := poi_z ∪ poi_e;
}
Alle POIs aus E, die sich mit poi überschneiden wurden in poi_z zusammengefasst. Galt dies für alle POIs aus E, ist also HMenge leer, füge poi_z in ZErg_i+1 ein. Sonst füge poi_z in HMenge ein und füge anschließend HMenge in ZErg_i+1 als Menge von POIs ein.
}
}

3. Wurde *ZErg_n* gebildet, bildet die Menge der einzelnen POIs, die in *ZErg_n* enthalten sind, das eigentliche Ergebnis. D. h. die POI-Mengen, die in *ZErg_n* enthalten sind, werden entfernt und die resultierende Menge wird als Ergebnis zurückgegeben.

Man beachte, dass in den Zwischenergebnissen, wie bei den POIResults, überdeckte oder doppelte POIs entfernt werden. Die Mehrarbeit durch diese Überprüfung steht dabei dem Gewinn durch das reduzierte Zwischenergebnis gegenüber und muss für übersichtliche Ergebnisse ohnehin erbracht werden.

5.2.2.3 Aufbau und Interpretation des Anfragebaums

Die Klassen des internen Modells der Anfrage sind im Paket *eworks.eRQL.model* bzw. dessen Unterpaket *eworks.eRQL.model.functions* enthalten. Für die einzelnen Anfrage Modi von eRQL, den Booleschen Verknüpfungen, wie auch für die Funktionen, werden Klassen, welche die entsprechende Funktionalität repräsentieren, bereitgestellt. In der Abbildung 50 sind die Klassen für die einzelnen Funktionen dabei nicht extra aufgeführt, sondern im Unterpaket zusammengefasst. Der Anfragebaum wird während der Übersetzung der Anfrage aus einzelnen Objekten dieser Klassen aufgebaut. Er stellt einem Operatorbaum dar, wobei die Knoten durch die Teilanfragen und den Booleschen Verknüpfungen gebildet werden.

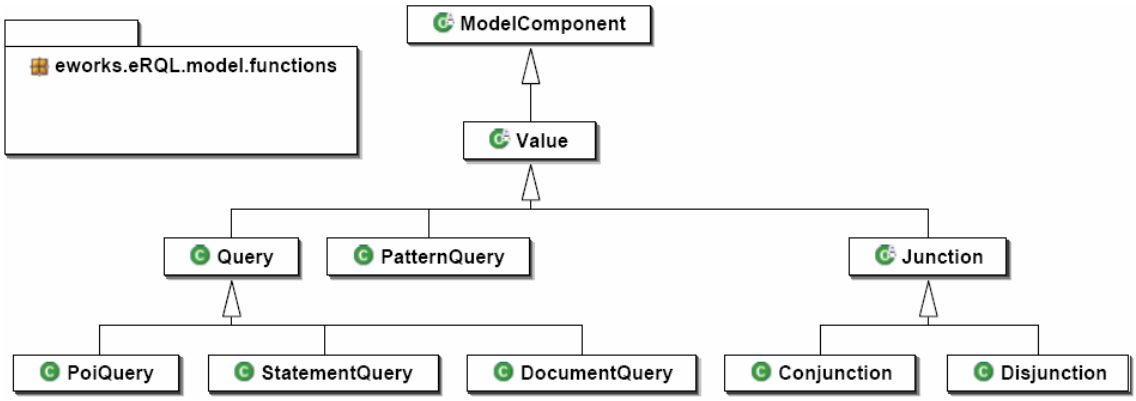


Abbildung 50 UML-Klassendiagramm des Pakets *eworks.eRQL.model*.

Die Bildung der internen Form der Anfrage kann sich dabei in mehrere Phasen vorgestellt werden. Diese laufen dabei wie folgt ab:

Phase 1: Der POI-Modus mit der Entfernung eins stellt die Grundeinstellung dar. Daher wird die eingegebene Anfrage erweitert. Hierbei werden Begrenzungsfunktionen oder Ein-Wort-Anfragen ohne Begrenzung genau dann um geschweifte Klammern erweitert, wenn sie nicht direkt von geschweiften oder eckigen Klammern umgeben sind.

Phase 2: Die eckigen Klammern des Statement-Modus dienen zum Verhindern der Erweiterung in Phase 1. Es wurde hierbei entschieden, dass der Statement-Modus nur an den Blättern wirksam ist. Die eckigen Klammern können daher nach der Phase 1 entfernt werden. Entsprechend kann das ~-Zeichen durch geschweifte Klammern {} ersetzt werden.

Phase 3: Durch die Interpretation der UND-Verknüpfung von POIResults muss diese bei mehreren UND-Verknüpfungen gleichzeitig erfolgen. Durch Anwendung der Regel 1 kann dies gewährleistet werden. Weiterhin wird der POI-Modus (geschweiften Klammern) für die ODER-Verknüpfungen (Regel 2) und den Dokumenten-Modus (Regel 3) zu den Blättern verschoben. Hierdurch kann die Berechnung der Umgebung in voller Größe am Stück erfolgen – z. B. durch einen rekursiven SQL-Befehl. Für die UND-Verknüpfung hingegen gilt dies nicht, da in diesem Fall das Ergebnis verändert werden würde.

Regel 1:

Query _1 AND Query _2 AND ... AND Query _i ⇒ AND(Query _1, Query _2, ..., Query _i)

Regel 2:

{OR(Query _1, Query _2, ..., Query _i)} ⇒ OR({Query _1}, {Query _2}, ..., {Query _i})

Regel 3[68]:

{<Query; QM; 0|1>} ⇒ <{ Query }; QM; 0|1>

Im Zusammenhang mit dem Dokumenten-Modus gibt es weitere Regeln, mit denen die Anfrage weiter umgeformt werden kann. Der Dokumenten-Modus *<Q; QM ;EXIN>* entspricht dabei einer Reduktion des Ergebnisses auf bestimmte Quellen. Dies kann mengentheoretisch betrachtet werden. Es sind jedoch zwei Interpretationen denkbar:

Interpretation 1: Sei *S* die Menge aller gespeicherten Quellen in der Datensenke, sei *QM* eine Menge von Quellen. Die Anfrage *<Q; QM ;1>* wird interpretiert als die Anfrage *Q* auf der Datensenke *S*, wobei das Ergebnis der Anfrage anschließend reduziert wird *auf* die Aussagen, die aus den Quellen von *QM* stammen. Die Anfrage *<Q; QM ;0>* wird interpretiert als die Anfrage *Q* auf der Datensenke *S*, wobei das Ergebnis anschließend reduziert wird *um* die Aussagen, die aus den Quellen aus *QM* stammen.

Interpretation 2: Sei *S* die Menge aller gespeicherten Quellen in der Datensenke, sei *QM* eine Menge von Quellen. Die Anfrage *<Q; QM ;1>* wird interpretiert als die Anfrage *Q* auf der Datensenke, für welche die Menge der gespeicherten Quellen $S_{neu} = QM \cap S$ gilt. Die Anfrage *<Q; QM ;0>* wird interpretiert als die Anfrage *Q* auf der Datensenke, für welche die Menge der gespeicherten Quellen $S_{neu} = S - QM$ gilt.

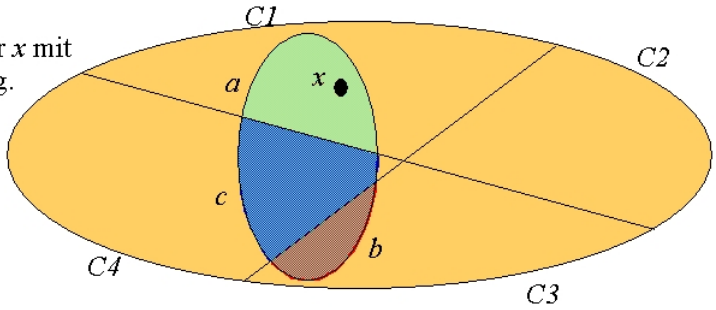
Von den beiden Interpretationen ist *Interpretation 1* weniger restriktiv, d. h., bei gleicher Anfrage kann *Interpretation 1* eine größere Ergebnismenge zurückliefern als *Interpretation 2*. Durch die spätere Reduktion des Ergebnisses können auch Ergebnisse entstehen, die keine

[68] Anstelle der Regel 3 wäre es auch denkbar die innere Anfrage im Dokumenten-Modus auszuführen, um anschließend eine Erweiterung der POIs im Ergebnis zu bewirken, bei der die Quellgrenzen der inneren Anfrage nicht mehr gelten.

direkten Treffer enthalten. Die folgende Abbildung 51 zeigt ein Beispiel, in dem der Unterschied der beiden Interpretationen grafisch dargestellt ist.

S := {C1, C2, C3, C4}
Anfrage *q* habe den direkten Treffer *x* mit der eingezeichneten POI Umgebung.
Seien die Mengen *a, b* und *c* wie folgt definiert:

a := Ergebnis von q ∩ C1;
b := Ergebnis von q ∩ C3;
c := Ergebnis von q ∩ C4;



Dann folgt :
Interpretation 1: *<q;C4 ;1> = c*
Interpretation 2: *<q;C4 ;1> = {}*

Abbildung 51 Verdeutlichung der unterschiedlichen Interpretationsmöglichkeiten für den Dokumenten-Modus.

Die folgenden Regeln 4 und 5 zur Umformung gelten für beide Interpretationen.

Regel 4:

<Q; QM ; 1> = <Q; S – QM ; 0>

Hiermit können Anfragen mit dem Wert *1* für *EXIN* in eine äquivalente Anfrage mit dem Wert *0* für *EXIN* überführt werden und umgekehrt. Die Regel folgt sofort aus: *QM ∩ S = S – (S – QM)*.

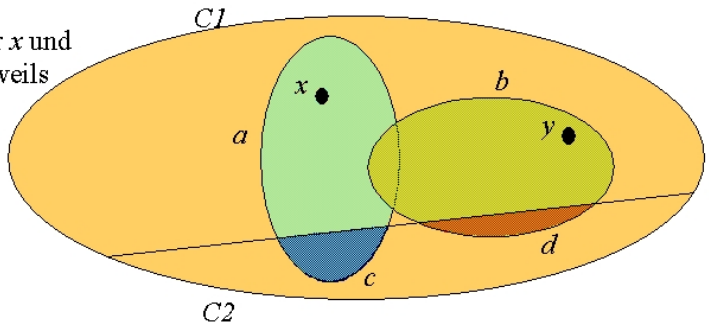
Weiterhin können ODER-Verknüpfungen aus dem Dokumenten-Modus herausgezogen werden.

Regel 5:

<Q1 OR Q2; QM ;EXIN> = <Q1;QM ;EXIN> OR <Q2;QM ;EXIN>

Für UND-Verknüpfungen gilt dies nicht für die *Interpretation 1*. Ein Gegenbeispiel ist in Abbildung 52 dargestellt.

S := {C1, C2}
Anfrage q_1 habe den direkten Treffer *x* und *q2* den direkten Treffer *y* mit den jeweils eingezeichneten POI-Umgebungen.



Dann folgt :
Interpretation 1:
<q1 AND q2;C2 ;1> = c ∪ d
<q1; C2 ;1> = c ∧ <q2 ; C2 ;1> = d, da aber *c ∩ d = {}* folgt:
<q1; C2 ;1> AND <q2 ; C2 ;1> = {}

Abbildung 52 Gegenbeispiel, dass bei der *Interpretation 1* für den Dokumenten-Modus die UND-Verknüpfung nicht aufgespaltet werden darf.

Bei der *Interpretation 2* hingegen wird die Anfrage auf die reduzierte Datenbasis QM durchgeführt. Daher gilt offensichtlich:

Regel 6 (nur *Interpretation 2*):

<Q1 AND Q2; QM ;EXIN> = <Q1;QM ;EXIN> AND <Q2;QM ;EXIN>

Die gleiche Situation ergibt sich für die Möglichkeit des Umformens von verschachtelten Anfragen im Dokumenten-Modus der Form *<<Q; QM1; EXIN_innen>; QM2; EXIN_außen>*. Hierbei reicht es, den Fall *EXIN_innen = EXIN_außen = 1* zu betrachten, da die anderen Fälle aus der Regel 5 folgen. Für die Interpretation 2 gilt dabei offensichtlich:

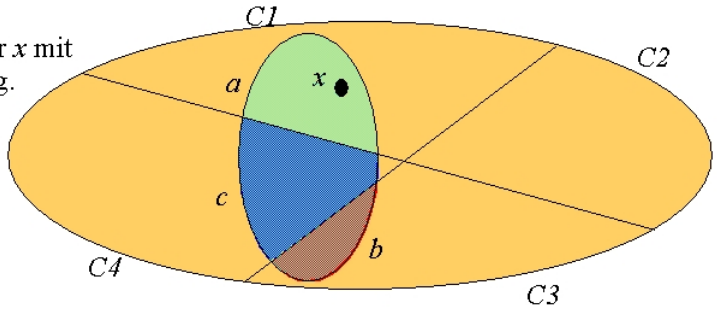
Regel 7 (nur *Interpretation 2*):

$$<<Q; QM1; EXIN\_i>; QM2; EXIN\_a> = \begin{cases} <Q; QM1 \cap QM2; EXIN\_i>, \text{falls } EXIN\_a = EXIN\_i = 1 \\ <Q; QM1 \cup QM2; EXIN\_i>, \text{falls } EXIN\_a = EXIN\_i = 0 \\ <Q; QM2 - QM1; EXIN\_a>, \text{falls } EXIN\_a = 1 \wedge EXIN\_i = 0 \\ <Q; QM1 - QM2; EXIN\_i>, \text{falls } EXIN\_a = 0 \wedge EXIN\_i = 1 \end{cases}$$

Für die *Interpretation 1* gilt dies nur, wenn auch: $QM1 \subseteq QM2$ gilt. Allgemein gilt es für die *Interpretation 1* nicht, da nicht nur die eigentlichen Treffer, sondern auch die POI-Umgebung mitberücksichtigt werden muss. Ein entsprechendes schematisches Gegenbeispiel wird in Abbildung 53 gezeigt.

$S := \{C1, C2, C3, C4\}$
Anfrage q habe den direkten Treffer x mit der eingezeichneten POI Umgebung.
Seien die Mengen a, b und c wie folgt definiert:

$a := Ergebnis\ von\ q \cap C1;$
$b := Ergebnis\ von\ q \cap C3;$
$c := Ergebnis\ von\ q \cap C4;$



Dann folgt :
$<<q;C1, C4\ ;1>;C4;1> = c$ | dies obwohl in c kein direkter Treffer enthalten ist!
$<q;\ \{C1, C4\} \cap \{C4\}\ ;\ 1> = <q;C4\ ;1> = \{\}$ | da hier kein direkter Treffer vorliegt!

Abbildung 53 Gegenbeispiel, welches zeigt, dass *<<Q; QM1; EXIN_innen>; QM2; EXIN_außen> ≡ <Q; QM1 ∩ QM2; EXIN_außen>* nicht allgemein für die *Interpretation 1* gilt.

In der Implementierung von eRQL wurde anfangs die *Interpretation 1* verwendet. Hierbei stand im Vordergrund, dem Nutzer möglichst viele Informationen zu liefern. Da Ergebnisse ohne direkte Treffer jedoch nicht hilfreich sind und zusätzlich die *Interpretation 2* durch kleinere Ergebnisse und durch die zusätzlichen Regeln einfacher zu handhaben ist, wurde auf die *Interpretation 2* gewechselt.

Die Regeln werden genutzt, um die gegebene Anfrage möglichst zu vereinfachen. In der Konsole wird zur Überprüfung der Umformung die eingegebene Anfrage sowie der erzeugte Anfragebaum und der umgeformte (*reduced*) Anfragebaum ausgegeben. Ein Beispiel hierfür ist in Abbildung 54 zu sehen. Neben der Zusammenfassung der ODER-Verknüpfungen werden die POI-Modi wenn möglich zusammengefasst. Die eingeklammerte Zahl *z* in *PoiQuery(z)* gibt dabei die Größe der Umgebung des POI-Modus an.

```
Query:      woman AND ~pablo OR Kuenstler

Query-Tree:
  Disjunction[ Disjunction[ PoiQuery(1)[ PatternQuery('Kuenstler') ]
  Conjunction[ Conjunction[ PoiQuery(1)[ PatternQuery('woman') ]
  PoiQuery(1)[ PoiQuery(1)[ PatternQuery('pablo') ] ] ] ] ] ]

(reduced):
  Disjunction[ PoiQuery(1)[ PatternQuery('Kuenstler') ] Conjunction[
  PoiQuery(1)[ PatternQuery('woman') ] PoiQuery(2)[
  PatternQuery('pablo') ] ] ]
```

Abbildung 54 Die Ausgabe in der Konsole, die den Anfragebaum und seiner reduzierte Form am Beispiel der Anfrage *woman AND ~pablo OR Kuenstler* zeigt.

5.2.3 Anfragegeschwindigkeit

In diesem Unterabschnitt werden einige Messergebnisse der eRQL Implementierung gezeigt, um die Stärken und Schwächen des Systems demonstrieren zu können. Getestet wurde auf einem Dell Latitude C610 Rechner mit PIII Mobile (1.0 GHz) Prozessor. Er verfügte über 512 MB Hauptspeicher und eine Festplatte vom Typ Hitachi-DK23DA-20. Als Betriebssystem war MS Windows 2000 (SP4) installiert und als DBMS wurde DB2 UDB v8.1 verwendet. Um repräsentative Ergebnisse zu erhalten, wurden die Anfragen jeweils zehn Mal wiederholt und der Durchschnittswert für die Ausführung berechnet. Man beachte dabei, dass keine Optimierung der Datenbank, z. B. durch zusätzliche Indizes, für die Anfrage durchgeführt wurde. Für größere Datenmengen wird dies unerlässlich sein.

Tabelle 9 Testumgebung zum Messen der Anfragegeschwindigkeit für eRQL-Anfragen.

Rechner	Prozessor	Arbeitsspeicher	Betriebssystem	RDBMS
Dell Latitude C610	PIII Mobile Prozessor (1.0 GHz)	512 MB	MS Windows 2000 (SP4)	IBM DB2 UDB v8.1 (Enterprise Edition)

Die Tests wurden auf den Kultur-Portal-Daten durchgeführt (siehe 2.2.1). Die dazugehörigen Dateien sind in der Distribution von RDF-S3 als Beispiel enthalten. Das Beispiel besteht aus fünf Dateien, wobei die Dateien *metaschema.rdf*, *admin.rdf* und *culture.rdf* Schemainformationen enthalten. Die beiden Dateien *culture_data.rdf (ID 7 / 4 KB / 71 RDF-Aussagen)* und *culture_data2.rdf (ID 5 / 2 KB / 34 RDF-Aussagen)* enthalten keine Schemainformationen. Die Testergebnisse sind in Tabelle 10 aufgelistet. In der oberen Hälfte der Tabelle wurde die Anfrage durch den Dokumenten-Modus von eRQL auf die Datei *culture_data.rdf* begrenzt. In der unteren Hälfte wurde auch die Datei *culture_data2.rdf* berücksichtigt. Dabei wurde die Ein-Wort-Anfrage *pablo* mit verschiedenen Radien für den POI-Modus verwendet.

Durch die Ergebnisse wird der Nutzen der Option *Follow double property usage* deutlicher. Insbesondere beim Radius von vier konnte durch Deaktivierung der Option eine deutliche Verbesserung der Antwortzeit festgestellt werden. Die Antwortzeit reduzierte sich von 10 auf 3 Sekunden. Bei der Anfrage auf zwei Quellen ergaben sich zwei direkte Treffer. Die deutlich höhere Antwortzeit hierbei ist insbesondere durch die Entfernung doppelter oder in anderen POIs enthaltener POIs zu erklären. Beim Radius von vier stieg die Antwortzeit auf 26 Sekunden bzw. auf fast 8 Sekunden, wenn doppelte Eigenschaftsverbindungen nicht berücksichtigt wurden.

Eine längere Antwortzeit mit wachsendem Radius war erwartet. Die Ergebnisse zeigen jedoch, dass hier noch nachgebessert werden muss. Kann keine Verbesserung, z. B. durch Indexnutzung, erreicht werden, sollte für einen produktiven Einsatz eine Begrenzung für den maximalen Radius eingeführt werden. Hierdurch kann vermieden werden, dass das System

überlastet und Benutzer zu lange warten müssen. Aus den gleichen Gründen sollten auch Anfragen mit zu vielen UND-Verknüpfungen oder zu unspezifischen Ein-Wort-Anfragen abgefangen werden.

Tabelle 10 Antwortzeiten und Ergebnisgrößen bei Anfragen nach *pablo* mit unterschiedlichen Radien und verschiedenen Einstellungen. Im oberen Bereich werden die Anfragen durch den Dokumenten-Modus auf nur eine Datei, im unteren Bereich auf zwei Dateien begrenzt.

Anfrage	Follow double property usage	Anzahl direkter Treffer / Grafen	Grafgröße	Antwortzeit (ms)
<pablo;7;1>	ja	1/1	5	17
<~pablo;7;1>	ja	1/1	11	54
<~~pablo;7;1>	ja	1/1	32	461
<~~~pablo;7;1>	ja	1/1	68	10201
<pablo;7;1>	nein	1/1	5	10
<~pablo;7;1>	nein	1/1	10	40
<~~pablo;7;1>	nein	1/1	15	285
<~~~pablo;7;1>	nein	1/1	27	2930
<pablo;7, 5;1>	ja	2/1	7	21
<~pablo;7, 5;1>	ja	2/1	16	98
<~~pablo;7, 5;1>	ja	2/1	45	1162
<~~~pablo;7, 5;1>	ja	2/1	99	26513
<pablo;7, 5;1>	nein	2/1	7	20
<~pablo;7, 5;1>	nein	2/1	12	82
<~~pablo;7, 5;1>	nein	2/1	24	755
<~~~pablo;7, 5;1>	nein	2/1	38	7796

Da die Schemastruktur der Kultur-Portal-Daten sehr flach ist, wurden für die Tests der Schemafunktionen andere Daten als Datenbasis verwendet. Öffentlich zugängliche RDF-Daten sind jedoch oftmals nicht konform zu den neusten RDF-Standards. Es wurden daher zufällig generierte RDF-Daten verwendet, die mittels des *RDF File Generatores*[69] erstellt wurden. Dieser wurde im Rahmen der Diplomarbeit von Jihua Xu [Xu04] entwickelt. Durch ihn werden vorwiegend Schemainformationen und Container erstellt, wobei eine maximale Tiefe für Klassenhierarchien angegeben werden kann. Die Tabelle 11 zeigt Anfrageergebnisse auf eine 1 MB-Datei (15.581 RDF-Aussagen), die vom RDF File Generator mit einer maximalen Hierarchietiefe von 20 erstellt wurde. Die Datei wurde in RDF-S3 unter der internen ID 3 abgelegt. Trotz großer Ergebnisgrößen verglichen mit den Anfragen im POI-Modus aus Tabelle 10, sind die Antwortzeiten hier niedriger. Dies kann durch die Möglichkeit des Zugriffs auf die SpecRepr und damit auf kleine Tabellen begründet werden. Auch sind keine rekursiven SQL-Befehle nötig, wie dies bei den Anfragen im POI-Modus der Fall ist.

[69] RDF File Generator, online unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/Comperator/Generator/

Tabelle 11 Antwortzeiten und Ergebnisgröße für eRQL-Schemafunktionen.

Anfrage	Ergebnisgröße	Antwortzeit (ms)
<classes();3;1>	642	34
<properties();3;1>	668	33
<container();3;1>	1192*	4636
<subClassOf(myClass98);3;1>	71	109
<superClassOf(myClass543);3;1>	153	151

* Die Elemente der Container werden im Ergebnis mit angezeigt. In den verwendeten Daten lagen pro Container zwischen 1 und 3 Elemente vor.

5.3 Zusammenfassung

Es wurden die Anforderungen an eine RDF-Anfragesprache für allgemeine Nutzer aufgezeigt und die Anfragesprache *eRQL* vorgestellt. Sie bietet folgende wesentlichen Vorteile gegenüber anderen RDF-Anfragesprachen:

1. Sie ist einfach zu erlernen und benötigt durch die *Ein-Wort-Anfragen* kein Umdenken im Vergleich zur Nutzung heutiger Internetsuchmaschinen.
2. Im Ergebnis kann der umgebende Graf mit angezeigt und so das Verständnis für das Ergebnis verbessert werden (*interner Kontext*).
3. Durch den Dokumenten-Modus kann die Herkunft der angefragten Daten berücksichtigt werden (*externer Kontext*). Dies erhöht das Vertrauen in die Ergebnisse, die Aktualität der Ergebnisse kann geprüft und weitere Informationen können von der Quelle bezogen werden.

Damit stellt eRQL eine Anfragesprache dar, die auch direkt in einem Informationsportal von Besuchern genutzt werden kann. Für einen produktiven Einsatz sind dabei Anpassungen empfehlenswert, wie sie in Unterabschnitt 6.2 dargestellt sind. Auch die funktionellen Erweiterungen, die dort beschreiben werden, sollten in betracht gezogen werden.

Man beachte, dass die einfache Nutzbarkeit von eRQL durch entsprechenden Mehraufwand in der Implementierung kompensiert wurde. Die verschiedenen Anfragemöglichkeiten auf Daten- und Schemaebene können in einer Anfrage verknüpft werden. Trotzdem werden die jeweiligen Anfragen durch unterschiedliche Ergebnisstrukturen so unterstützt, dass die Ergebnisse einfacher verstanden werden können. Die unterschiedlichen Ergebnisstrukturen mussten daher miteinander verknüpfbar sein, was für zusätzlichen Aufwand für die Interpretation und die Implementierung gesorgt hat. Beim Durchlesen des Kapitels kann es daher zu dem Gedanken kommen: *Was soll an eRQL so einfach sein?* Die Einfachheit wird jedoch aus Sicht des Benutzers erkennbar, der in eRQL Anfragen wie in heutigen Internetsuchmaschinen nutzen kann und aussagekräftigere Ergebnisse als bei anderen RDF-Anfragesprachen erhält.

Beispiel aus der Einleitung

In der Einleitung wurde das Beispiel genannt, dass nach einer Frau gesucht wird, von der man weiß:

- Person ist weiblich.
- Person heißt *Tolle* mit Nachnamen.
- Person ist verheiratet.
- Gatte arbeitet an der Universität.

Eine eRQL-Anfrage hierfür könnte: *Tolle AND Universität* lauten. Durch die Anfrage würden Teilgrafen aus den vorhandenen Daten generiert werden, die beide Begriffe enthalten und zwischen denen maximal eine weitere RDF-Aussage steht. Die maximale Entfernung der direkten Treffer (RDF-Aussagen, die einen der gesuchten Begriffe enthalten) kann in der Anfrage verändert werden. Da im Ergebnis die Umgebung zu den direkten Treffern enthalten ist, könnte dort bereits die gesuchte Information vorhanden sein.

Natürlich kann die Anfrage mit weiteren Informationen verfeinert werden. So könnten Informationen über vorhandene Klassen gegeben sein. Ist z. B. die Klasse *ex:Frau* bekannt, die alle Frauen enthält, so kann dies genutzt werden, um aus den Ergebnisgrafen alle vorhandenen Frauen zu extrahieren. Die entsprechende eRQL-Anfrage hierfür würde lauten:

instancesOf(ex:Frau) AND Tolle AND Universität

Durch weiteres Schemawissen über die Klassenhierarchien bzw. welche Eigenschaften die Sachverhalte *‚arbeitet an'*, *‚ist Nachname von'* oder *‚ist verheiratet mit'* repräsentieren, könnte die Anfrage weiter verfeinert und so die Genauigkeit im Ergebnis erhöht werden.

6 Zukünftige Arbeiten

In diesem Kapitel werden Möglichkeiten aufgezeigt, wie RDF-S3 und eRQL erweitert werden können. Wie sinnvoll die jeweiligen Erweiterungen sind, hängt vom gegebenen Anwendungsfall ab.

6.1 Erweiterungsmöglichkeiten für RDF-S3

In 6.1.1 wird eine intelligentere Möglichkeit des Löschens bzw. Aktualisierens erörtert. Neben einer erwarteten Leistungssteigerung für diese Operationen kann dies dazu führen, dass auf die ausgiebige Nutzung von Fremdschlüsseln verzichtet werden kann. Bereits in 4.3.4.3 konnte für die Speicherung hierfür eine Leistungssteigerung von 33 % erzielt werden.

In RDF-S3 werden die einzelnen Dokumente als unabhängig voneinander betrachtet. Insbesondere für Anwendungsfälle, die in einem Intranet arbeiten, könnte es sinnvoll sein dieses Paradigma aufzuheben. In 6.1.2 wird daher beschrieben, wie sich Schemaänderungen auf andere Dokumente, die sie verwenden, auswirken können bzw. sollten.

6.1.1 Weitere Möglichkeiten der Aktualisierung

RDF-S3 bietet die Möglichkeit der Aktualisierung über die *RDF-S3 Administration GUI* (siehe Abbildung 20). Die Aktualisierung wird hierbei durch den naiven Ansatz des Löschens und anschließenden Ladens einer Quelle realisiert. In vielen Fällen bildet die Menge der unveränderten Daten dabei den größten Anteil. Deutlich effektiver scheint es daher, die Daten des alten und aktuellen Modells zu analysieren und nur die *gelöschten Daten* zu löschen und die *neuen Daten* zu speichern. Die Modelle können dabei als Mengen von Aussagen betrachtet werden. Die Aufteilung in die Mengen der *neuen*, *unveränderten* und *gelöschten Daten* für die Betrachtung von Quellen über einen Zeitraum ist in Abbildung 55 grafisch dargestellt.

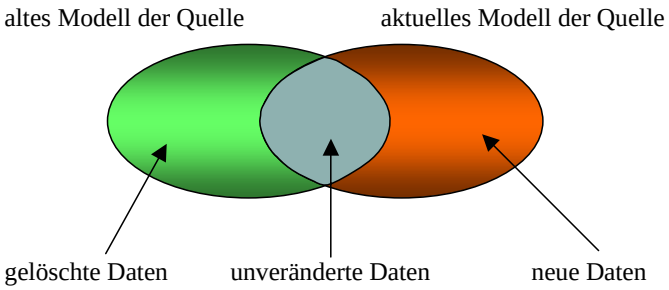


Abbildung 55 Mengenaufteilung der RDF-Aussagen bezüglich einer Quelle zu unterschiedlichen Zeiten.

Die Berechnung dieser einzelnen Mengen ist jedoch nicht ganz unproblematisch. Hierfür muss die *Gleichheit von RDF-Aussagen* geprüft werden. Zwei RDF-Aussagen sind gleich, wenn ihre Subjekte, Prädikate und Objekte gleich sind. Für die Gleichheit von Ressourcen mit URI-Referenz müssen die URIs übereinstimmen. Bei Literalen hingegen müssen deren Werte, Datentypen und auch die Sprachedefinitionen gleich sein. Schwierigkeiten bereiten *anonyme Ressourcen*, die als Subjekt oder Objekt auftreten können. Ihre URI wird während des Parsens durch den Übersetzer frei bestimmt. Das Wechseln des Übersetzers oder eine einfache Umstrukturierung in der RDF/XML-Datei kann daher dazu führen, dass eine anonyme Ressource eine andere URI erhält. Auch wenn sie unterschiedliche URIs haben, bleiben die anonymen Ressourcen *äquivalent*, bzw. umgekehrt kann es sich bei gleicher URI um unterschiedliche anonyme Ressourcen handeln. Um die Äquivalenz von anonymen Ressourcen bestimmen zu können, müssen daher die Aussagen, in denen die anonymen Ressourcen auftreten bis auf die anonyme Ressource selbst gleich sein. Am Besten kann man sich dies an der Grafrepräsentation von RDF vorstellen. Dort werden anonyme Ressourcen als leere Ovale

gezeichnet. Legt man diese Ovale übereinander und ist der Graf in der direkten Umgebung (Entfernung eins) deckungsgleich, so sind die anonymen Ressourcen äquivalent.

Dies erfordert einen sehr hohen Vergleichsaufwand. Als Alternative hierfür ist ein generelles Löschen und erneutes Einfügen von Aussagen, die anonyme Ressourcen enthalten, denkbar. Dieser Ansatz soll weiter untersucht werden. Die Aktualisierung einer Quelle könnte damit wie folgt durchgeführt werden:

1. Erstellen des aktuellen Modells der Quelle, z. B. mithilfe von VRP. Hierbei wird davon ausgegangen, dass keine Duplikate im Modell zugelassen sind.
2. Extrahieren der Aussagen des alten Modells aus der Tabelle *poi*.
3. Löschen aller Aussagen des alten Modells, die eine anonyme Ressource enthalten. Die Aussagen werden hierbei sowohl aus der Datensenke, als auch aus dem alten Modell gelöscht.
4. Durchlauf der Aussagen des alten Modells:
    a. Ist eine Aussage im neuen Model enthalten, so gehört sie zu den *unveränderten Daten*. Die Aussage wird aus dem neuen Modell entfernt.
    b. Ist die Aussage dort nicht vorhanden, gehört sie zu den *gelöschten Daten* und wird aus der Datensenke entfernt.
5. Nach dem Durchlaufen aller Aussagen des alten Modells sind im neuen Modell die *neuen Daten* und Aussagen mit anonymen Ressourcen übrig. Diese werden in die Datensenke eingefügt.
6. Abschließend ist der Zeitstempel im Kontextknoten der Quelle anzupassen.

Man beachte, dass diese Vorgehensweise darauf angewiesen ist, das aktuelle Modell erstellen zu können. Mit der Stream Based API ist dieses Vorgehen nicht vereinbar. Auch werden Duplikate bei der Nutzung von VRP nicht mit aufgenommen. Werden andere Werkzeuge zum Aufbau des aktuellen Modells verwendet, ist dies zu berücksichtigen. In diesem Fall muss das Modell jeweils bis zum Ende durchlaufen werden, um Duplikate entfernen zu können.

6.1.2 Behandlung von Schemaänderungen

Um Gegebenheiten, Zusammenhänge oder Situationen zu beschreiben, werden Modelle verwendet, die diese abstrahieren. Da die Welt nicht still steht, können sich Gegebenheiten, Zusammenhänge, Situationen oder zumindest deren Sichtweise ändern, z. B. die benötigte Genauigkeit der Abstraktion des Modells. Es ist daher unvermeidbar Änderungen im Modell durchzuführen zu können [Dec02].

Nehmen wir als Beispiel eine Reifenfabrik, die Reifen für Fahrräder produziert. Alle in der gleichen Größe. Ein entsprechendes Modell würde daher die Größe nicht unbedingt berücksichtigen. Ändert sich die Situation und es werden nun Reifen unterschiedlicher Größe produziert, muss das Modell angepasst werden. In diesem Fall durch das Hinzufügen einer Größenangabe zu den einzelnen Reifen.

Bisher werden in RDF-S3 einzelne Quellen als unabhängig betrachtet. D. h. eine Änderung in einem Dokument, hat keinen direkten Einfluss auf ein anderes gespeichertes Dokument. In RDF werden RDF-Schemata verwendet, um die Modelle zu beschreiben. Diese stellen ebenfalls RDF-Dokumente dar. Es stellt sich die Frage, ob eine Änderung in einem RDF-Schema auf die Dokumente, welche dieses verwenden, zu propagieren ist?

Da das Semantische Web ein verteiltes System darstellt, in dem man nicht unbedingt die Kontrolle über die verwendeten Schemata hat. Es ist daher berechtigt, wie in RDF-S3 vorzugehen, sodass Änderungen eines Schemas keinen Einfluss auf Dokumente haben.

Anders stellt ist die Situation, wenn eine Lösung für eine Firma in einem Intranet realisiert wird. In diesem Fall stellen die Schemata die Sichtweise auf die Firmenwelt dar. Ändert sich diese Sichtweise, wie oben bei der Reifenfabrik beschrieben, ist es sinnvoll diese Änderung auf alle Daten zu propagieren, um so eine einheitliche und gültige Darstellung zu erlangen. Es wird daher im Folgenden betrachtet, wie sich Schemaänderungen auswirken können. Hierbei wird darauf geachtet, wie die Gültigkeit der gespeicherten Daten in Bezug auf ein sich änderndes Schema zu gewährleisten ist.

6.1.2.1 Begriffe und Fokus

Im Semantischen Web stellt die *RDF Vocabulary Description Language 1.0 (RDFS)* die unterste Ebene zur Beschreibung von Schemata dar. Mit ihr können Klassen, Eigenschaften sowie deren Hierarchien beschrieben werden. In meinen Betrachtungen wird sich auf diese Ebene beschränkt. Ein weiterer Schritt wäre es, die Überlegungen bezogen auf die *Web Ontology Language* (*OWL*) zu erweitern. OWL baut auf RDFS auf und ergänzt es, sodass logische Axiome ausgedrückt werden können. Als Beispiel sei hier die Transitivität von Eigenschaften oder Mengenbeziehungen zwischen Klassen (disjunkt, überlappend) genannt.

Wie sich Schemaänderungen auf gespeicherte Daten auswirken können bzw. sollten, ist im Bereich der Datenbanken bereits in der Vergangenheit untersucht worden. Durch den objektorientierten Ansatz von RDF bietet sich ein Vergleich mit objektorientierten Datenbanken (OODB) an. Bei OODB haben wir Klassen, Attribute und Methoden. In RDF sind Methoden nicht vorgesehen. Dagegen entsprechen RDF-Klassen den Klassen der OODB und in beiden Ansätzen existieren Klassenhierarchien. Eigenschaften in RDF sind den Attributen der OODB ähnlich. Sie sind jedoch eigenständig und werden global definiert. Dabei werden sie eindeutig über ihre URI identifiziert. Attribute in OODB hingegen können in einer Klasse lokal definiert werden beziehungsweise vererbte Attribute können überschrieben werden. Anstatt der Vererbung von Attributen gibt es in RDF die Eigenschaftshierarchien.

Unter einer *Schemaänderung* wird ein einzelner Änderungsschritt verstanden. Einzelne Änderungen können sein (vergleiche Abschnitt 17.3 aus [Zan+97]):

1. Änderungen bezogen auf Eigenschaften
    a. Erstellen einer Eigenschaft
    b. Löschen einer Eigenschaft
    c. Umbenennen einer Eigenschaft
    d. Änderung des Definitionsbereiches einer Eigenschaft
    e. Änderung des Wertebereiches einer Eigenschaft
2. Änderungen bezogen auf Klassen
    a. Erstellen einer Klasse
    b. Löschen einer Klasse
    c. Umbenennen einer Klasse
3. Änderungen von Hierarchien
    a. Erzeugen einer Ober/Unter-Klassenbeziehung
    b. Löschen einer Ober/Unter-Klassenbeziehung
    c. Erzeugen einer Ober/Unter-Eigenschaftsbeziehung
    d. Löschen einer Ober/Unter-Eigenschaftsbeziehung

Neben dem Begriff der Schemaänderung ist auch der Begriff der *Schemaevolution* zu finden. Eine Schemaevolution stellt eine endliche Menge von Schemaänderungen dar, die durch Hintereinanderausführung von einer Version eines Schemas zu einer neuen Version führt. Werden im Nachhinein nur der beiden Versionen eines Schemas betrachtet, sind die einzelnen Änderungsschritte nicht immer nachvollziehbar. Das Anpassen der Daten, die mit dem alten Schemata konform sind, an das neue Schemata ist allgemein nicht möglich. Um eine Schemaevolution unterstützen zu können, müssen die einzelnen Schemaänderungen oder zumindest eine Abbildungsanweisung zur neuen Version bekannt sein. Solche Informationen können als *Schema-Kontext-Informationen* angesehen werden. In diesem Bereich setzt z. B. das *Context Modelling Framework* an, welches in [Bon04] beschrieben wird.

Mit der Definition der Schemaänderung, die nur einen Änderungsschritt enthält, kann eine Anpassung der Daten in eine für das neue Schema gültige Form vollzogen werden. Die nötigen Anpassungen für die jeweiligen Fälle werden in 6.1.2.2 beschrieben.

Dabei muss berücksichtigt werden, dass im Semantischen Web teilweise andere Voraussetzungen als bei Datenbanken gelten. In [NK03] wird genauer auf den Unterschied zwischen der Änderung eines Datenbankschemas und eines Ontologieschemas eingegangen. Für die Änderung von RDF-Schemainformationen, wie sie hier untersucht werden, gelten vor allem die Unterschiede, die durch die dezentrale Nutzung entstehen. Um die RDF-Schemainformationen möglichst effektiv nutzen und wiederverwenden zu können, werden diese üblicherweise in separaten RDF-Dokumenten (Namensräume) gehalten. Durch Verweise aus anderen RDF-Dokumenten können die definierten Eigenschaften und Klassen wiederverwendet werden. Da Namensräume ebenfalls RDF-Dokumente darstellen, können auch sie auf andere Namensräume verweisen und so von diesen abhängen. Bei frei verfügbaren Namensräumen führ dies dazu, dass der Produzent eines Schemas nicht weiß, in welchen Dokumenten und in welchen Anwendungen sein Schema genutzt wird. Die Auswirkungen einer Änderung lassen sich daher nicht immer vorab bestimmen. Dies macht einen sorgfältigen und verantwortungsvollen Umgang von Schemaänderungen notwendig. Das W3C empfiehlt hierzu, einem geänderten Namensraum auch eine neue URL zuzuweisen. Die Realität zeigt jedoch, dass hiervon nicht immer ausgegangen werden kann. Selbst die RDF- und RDFS-Namensräume unterlagen in der Vergangenheit vieler Änderungen ohne einen Wechsel der URL.

Ein weiterer Unterschied ist die fehlende Trennung von Daten und Schemadaten. Beide können in einem RDF-Dokument gemischt auftreten. Dies stellt eine zusätzliche Problematik dar, die im Rahmen einer Ontologie-Versionierung [KF01] betrachtet werden müsste. Bei der Ontologie-Versionierung können Daten bezüglich neueren oder älteren Ontologien betrachtet werden (Sichtweisen), wobei Kompatibilitätsbedingungen erfüllt sein müssen.

Interessante weiterführende Informationen zum Bereich der Schemaänderungen und der Ontologien sind auch in Bezug auf die *Semantic HTML Ontology Extension (SHOE)* unter [HHL99, Hef01] zu finden. SHOE gilt als einer der Vorläufer des Semantischen Webs.

6.1.2.2 Anpassungen der Daten an Schemaänderungen

Eine Schemaänderung in RDF entspricht dem Entfernen bzw. Hinzufügen von Aussagen eines Grafen. Die Interpretation der RDF-Grafen ist in [W3C Semantic 04] festgelegt. Wie in Abschnitt 2.3 erwähnt, existieren jedoch insbesondere für die Eigenschaften *rdfs:range* und *rdfs:domain* verschiedene Interpretationsmöglichkeiten. Die Interpretation als Beschränkung soll hier im Vordergrund stehen. In der folgenden Liste sind die Auswirkungen jeder der möglichen Änderung auf existierende Daten angegeben.

Änderungen der RDF- oder RDFS-Namensräume haben dabei u. U. weitergehende Auswirkungen. Mögliche Änderungen an diesen Namensräumen werden hier nicht betrachtet. Diese beiden Namensräume werden als unveränderbar angenommen.

1. Änderungen bezogen auf **Eigenschaften**

   a. Erstellen einer Eigenschaft – durch die Erstellung einer Eigenschaft wird diese für alle Ressourcen des Definitionsbereiches nutzbar. Die Nutzung einer Eigenschaft kann nicht erzwungen werden. Daher werden durch die Erstellung einer Eigenschaft weder existierende Daten ungültig, noch müssen diese erweitert werden.

   b. Löschen einer Eigenschaft – beim Löschen einer Eigenschaft ist auf die Eigenschaftshierarchie zu achten. Wird eine Eigenschaft gelöscht, die Untereigenschaften besitzt, so sollten die Untereigenschaften ebenfalls gelöscht werden. Ist dies nicht gewünscht, sollte vorher die Eigenschaftsbeziehung gelöscht werden. Zum eigentlichen Löschen werden die Verbindungen, welche die zu löschende Eigenschaft vornimmt und die Aussagen, welche die Eigenschaft als Subjekt oder Objekt enthält, entfernt.

   c. Umbenennen einer Eigenschaft – dies entspricht in RDF der Änderung der URI. Zu beachten ist, dass diese nicht bereits für eine andere Ressource verwendet werden darf. Die Änderung der URI muss in jeder Aussage vorgenommen werden, die diese Eigenschaft nutzt.

   d. Änderung des Definitionsbereiches (*rdfs:domain*) einer Eigenschaft – in RDF sind Definitionsbereiche für Eigenschaften als Mengen von Klassen definiert. Die mit der Eigenschaft verwendeten Subjekte müssen Instanzen aller Klassen des Definitionsbereichs sein. Werden Klassen des Definitionsbereiches entfernt, hat dies an sich keine Auswirkungen. Wurde *rdfs:domain* jedoch als Instanzierungshilfe verwendet, sind eventuelle Instanzierungen aufzuheben. Es stellt sich jedoch die Frage, ob dies rückverfolgt werden kann, da nicht auszuschließen ist, dass die Instanzierung zusätzlich über die Eigenschaft *rdf:type* vorgenommen wurde.

      Werden weitere Klassen im Definitionsbereich mit aufgenommen, ist bei den Verbindungen der Eigenschaften zu prüfen, ob die Subjekte Instanzen dieser Klasse sind. Geprüft werden sollte vorher, ob die neuen Klassen Oberklassen von (oder gleich mit) bisherigen Definitionsbereichsklassen sind. In diesem Fall kann auf eine Überprüfung verzichtet werden. Ist ein mit der Eigenschaft verwendetes Subjekt nicht Instanz der neuen Domainklassen, ist dies ungültig. Je nach Interpretation kann dies zu einem Fehler (Interpretation als Beschränkung) oder zu zusätzlichen Instanzierungen (Interpretation als Instanzierungshilfe) führen.

   e. Änderung des Wertebereiches (*rdfs:range*) einer Eigenschaft – dies ist zu behandeln wie die Änderung des Definitionsbereiches. Auch diese sind in RDF als Mengen von Klassen definiert. Die mit der Eigenschaft verwendeten Objekte müssen Instanzen aller Klassen des Wertebereichs sein. Für die Entfernung bzw. das Hinzufügen von Klassen des Wertebereiches gilt daher das Gleiche wie für die Änderung des Definitionsbereiches oben.

2. Änderungen bezogen auf **Klassen**

   a. Erstellen einer Klasse – ist nicht problematisch, alle bisherigen Daten bleiben gültig.

   b. Löschen einer Klasse – beim Löschen einer Klasse ist auf die Klassenhierarchie zu achten. Wird eine Klasse, die Unterklassen besitzt, gelöscht, so sollten die Unterklassen ebenfalls gelöscht werden. Ist dies nicht gewünscht, sollte zuerst die Klassenbeziehung gelöscht werden. Zum eigentlichen Löschen einer Klasse werden alle Instanzen der Klasse gelöscht. Zu beachten ist die Möglichkeit der Mehrfachinstanzzierung. Ist eine Ressource Instanz einer zu löschenden Klasse jedoch auch Instanz einer Klasse, die nicht gelöscht werden soll, so ist nur der entsprechende Instanzzierungseintrag (*rdf:type*) zu löschen. Bei dieser Betrachtung dürfen nur explizite Instanzzierungen beachtet werden, andernfalls könnte keine Ressource gelöscht werden, da jede Ressource Instanz der

Klasse *rdfs:Resources* ist. Weiterhin sind alle Aussagen mit der Klasse als Subjekt oder Objekt zu entfernen. Man beachte, dass dies zum Ändern von Definitions- oder Wertebereiche von Eigenschaften führen kann.

Besondere Vorsicht ist geboten, falls die zu löschende Klassen Unterklasse der Klassen *rdfs:Classes* oder *rdf:Property* ist. Die Instanzen dieser Klasse sind ihrerseits Klassen bzw. Eigenschaften. Existieren für diese keine weiteren expliziten Instanzierungen, würden diese ebenfalls gelöscht werden. Ein solcher Eingriff auf der Metaebene sollte nur erfahrenen Nutzern erlaubt sein und mit entsprechenden Warnhinweisen versehen werden.

c. Umbenennen einer Klasse – dies entspricht in RDF der Änderung der URI. Zu beachten ist, dass diese nicht bereits für eine andere Ressource verwendet werden darf. Die Änderung der URI muss in jeder Aussage vorgenommen werden.

3. Änderungen von **Hierarchien**

a. Erzeugen einer Ober/Unter-Klassenbeziehung – hierbei können Zyklen entstehen. In RDF sind diese zwar grundsätzlich erlaubt, jedoch würde daraus folgen, dass alle Klassen des Zyklus gleich sind (siehe 2.3). Einen weiteren Einfluß auf die Gültigkeit der Daten hat das Einfügen einer Klasse in die Klassenhierarchie nicht.

b. Löschen einer Ober/Unter-Klassenbeziehung – hierdurch verlieren alle Instanzen der Unterklasse und deren Unterklassen die Zugehörigkeit zur Oberklasse und deren Oberklassen, sofern zu diesen nicht eine weitere Klassenbeziehung besteht. Die Eigenschaften, die eine der Oberklasse als Definitions- oder Wertebereich haben, müssen nun auf die Nutzung von Instanzen der Unterklasse überprüft werden, um die Gültigkeit der Daten zu gewährleisten. Es kann hierbei auch zum Aufbrechen vorhandener Zyklen kommen, wodurch die Gleichheit der Klassen wieder aufgehoben wird.

c. Erzeugen einer Ober/Unter-Eigenschaftsbeziehung – hier gilt das Gleiche wie für das Erzeugen einer Ober/Unter-Klassenbeziehung. Zusätzlich muss jedoch geprüft werden, ob die Definitions- und Wertebereiche der Untereigenschaft Teilmengen der Definitions- und Wertebereiche der Obereigenschaft sind.

d. Löschen einer Ober/Unter-Eigenschaftsbeziehung – darf immer durchgeführt werden und führt zu keinen ungültigen Daten.

6.2 Erweiterungsmöglichkeiten für eRQL

Die Erweiterungsmöglichkeiten für eRQL werden aufgeteilt in Erweiterungen funktioneller Art (siehe 6.2.1) und Erweiterungen, die die Anfragegeschwindigkeit verbessern können (siehe 6.2.2).

6.2.1 Erweiterungen auf funktioneller Ebene

- **Dokumenten-Modus mit Bedingungen für Quellen** – bisher ist im Dokumenten-Modus nur die Angabe von Quellen möglich. Auf weitere Informationen, die in den Kontextknoten enthalten sind, wie die Zeitstempel des Ladens bzw. des letzten Aktualisierens, kann nicht zugegriffen werden. Eine zweckmäzige Anfrage könnte sein: *Führe Anfrage q auf allen Quellen aus, die innerhalb der letzten beiden Tagen eingefügt oder aktualisiert wurden!*

  Agenten, welche die Anfrage *q* bereits vor zwei Tagen durchgeführt hatten und nur Neuerungen suchen, könnten hiervon profitieren. Um hier flexibel genug zu sein, wären Anfragen auf die Menge der Kontextknoten sinnvoll, die in den Dokumenten-Modus aufgenommen werden könnten. Der Dokumenten-Modus hätte damit die Form: *<Anfrage; KK-Anfrage; EXIN>*, wobei *KK-Anfrage* eine Anfrage an die Menge der Kontextknoten

darstellt, die als Ergebnis eine Menge von Kontextknoten zurückgibt. Das Ergebnis der KK-Anfrage entspricht dabei der jetzigen Quellenmenge *QM*.

- **Zusätzliche Funktionen** – hierzu zählen die bis jetzt noch nicht implementierten RQL-Funktionen *typeof(), leafclass(), topclass(), leafproperty()* und *topproperty()*. Es sind weiterhin Funktionen vorstellbar wie: *path(x,y)*, die neben der Aussage, ob es Pfade von *x* nach *y* gibt, die kürzesten Pfade aufzeigt.
- **Erweiterung auf OWL** – bisher wird sich auf das RDF-Kernvokabular begrenzt. Da die *Web Ontology Language (OWL)* immer mehr an Bedeutung gewinnt, ist eine Ausweitung auf deren Klassen und Eigenschaften wünschenswert.
- **Einbeziehung von Thesauren und semantischen Beziehungen** – eines der Ziele von eRQL ist die Unterstützung von Benutzern, die nicht genau wissen, wonach sie eigentlich suchen. Hierfür bietet eRQL die Ein-Wort-Anfrage, die über die Nutzung des *like*-Operators eine gewisse Unschärfe in die Anfrage bringt. Dies kann über die Nutzung eines Thesaurus weiter unterstützt werden, sodass auch nach semantisch ähnlichen Begriffen gesucht wird. Ebenso könnten die semantischen Beziehungen der Eigenschafts- und Klassenhierarchien mit einbezogen werden. Dies wird bereits standardmäßig von RQL angeboten.
- **Ergebnisdarstellung** – die Ergebnisdarstellung ist von zentraler Bedeutung, wenn es um das Verständnis für den Nutzer geht. Die Ergebnisse können entsprechend persönlicher Profile sortiert und gewichtet werden. Weiterhin würde eine Visualisierung von Ergebnisgrafen eine deutliche Verbesserung zur jetzigen textuellen Darstellung bedeuten, da dies der menschlichen Wahrnehmung entgegenkommt. Für die Visualisierung von RDF-Grafen gibt es bereits Anwendungen wie IsaViz[70]. Hierbei sollte der Nutzer bei größeren Ergebnisgrafen nicht überlastet werden. Es ist keinem Nutzer zumutbar, einen Grafen mit über 100 Knoten als Ganzes anzuzeigen. Um die Lesbarkeit und das Verständnis von Ergebnissen zu verbessern, sollten direkte Treffer hervorgehoben werden. Dies ist auch für bestimmte Eigenschaften, Klassen oder weiterer RDF-Konstrukte sinnvoll, was über persönliche Einstellungen des Benutzers gesteuert werden könnte. Weiterhin sei erwähnt, dass Literale für das menschliche Verständnis eine hohe Bedeutung haben und daher ebenfalls hervorgehoben werden sollten.

6.2.2 Erweiterungen zur Verbesserung der Anfragegeschwindigkeit

Eine Verbesserung der Anfragegeschwindigkeit kann auf verschiedenen Ebenen erfolgen. So sind Änderungen auf Programmierebene möglich, aber auch Erweiterungen von eRQL oder Änderungen von Grundsatzentscheidungen.

- **Ergebniszugriff** – um auf die Ergebnisse der Datenbankanfragen (*ResultSets*) zuzugreifen, werden die Java-Methoden *getInt* bzw. *getString* verwendet. Als Parameter erwarten diese entweder die Spaltennummer oder den Spaltennamen. Aus Geschwindigkeitsgründen ist hierbei die Spaltennummer vorzuziehen [Goo04], da bei der Verwendung der Spaltennamen diese erst in die entsprechende Spaltennummer übersetzt werden müssen. In der Implementierung wurden jedoch die Spaltennamen verwendet, um eine bessere Anpassungsfähigkeit auf Änderungen der Tabellenstrukturen oder der SQL-Anfragen zu erreichen. Ein Wechsel auf Spaltennummern könnte daher die Anfragegeschwindigkeit verbessern.
- **Anfrageoptimierung** – für einige Anfragen ist durch eine ausgefeiltere Anfrageoptimierung innerhalb von eRQL eine Verbesserung denkbar. Sei z. B. die Anfrage *[obj(http://www.myschema.de)] AND [„test“]* gegeben. Beide Teilanfragen sind im

[70] IsaViz, online unter: http://www.w3.org/2001/11/IsaViz

Statement-Modus. Daher würden die Ergebnisse vom Typ *POIResult* jeweils nur *POIs* enthalten, die aus genau einer Aussage bestehen. Dabei gilt für die erste Teilanfrage, dass die Aussagen als Objekt die Ressource *http://www.myschema.de* aufweisen müssen. Für die zweite Teilanfrage muss in den Aussagen die Zeichenkette *test* innerhalb des Wertes eines Literals enthalten sein. Da Literale in RDF nur als Objekte auftreten dürfen, ist klar, dass das Ergebnis zur zweiten Teilanfrage nur Aussagen enthält, die Literale als Objekte besitzen. Daraus folgt unabhängig vom Inhalt der Datensenke, dass das Ergebnis für die gegebene Anfrage leer sein muss, da sich die Teilanfragen widersprechen.

Neben dem Auffinden von Fällen, in denen sich die Teilanfragen widersprechen, können sich Anfragen überdecken, z. B. die Anfrage *„test“ OR „tes“*. Hierbei ist klar, dass das *POIResult* der ersten Teilanfrage eine Untermenge des *POIResults* der zweiten Teilanfrage darstellt. Die Anfrage ist also äquivalent zur Anfrage *„tes“*. Neben der gesparten Zeit für die erste Teilanfrage, würde in diesem Fall die Anfrageoptimierung auch die aufwendige Duplikatentfernung im Ergebnis erleichtern.

- **Nutzung von Indizes** – Ein-Wort-Anfragen werden in eRQL mit den ganzen URI-Referenzen bzw. den Werten der Literale verglichen. Hierbei wird der *like*-Operator in den SQL-Anfragen verwendet und die Anfragezeichenkette sowohl am Anfang, als auch am Ende, um ein *-Wildcard ergänzt. Diese Grundsatzentscheidung verhindert die Nutzung von Indizes, da diese nur genutzt werden können, wenn der Anfang des Suchbegriffes feststeht. Je nach Anwendung könnte hier eine Reduktion auf den Ressourcen Namen, wie dies bereits bei Klassen und Eigenschaften vorgenommen wird, stattfinden. Dies würde einen Verzicht auf Wildcards am Anfang ermöglichen und so die Nutzung von Indizes erlauben.

  Bezogen auf die URI-Referenzen der Ressourcen wäre auch ein Unique-Index denkbar. Hierfür ist eine Aufspaltung der Tabelle *resources* der ComRepr zum Vermeiden von Null-Werten nötig.

- **UND-Verknüpfungen** – die Berechnung mehrerer UND-Verknüpfungen bleibt in eRQL sicherlich sehr kostenspielig, da diese nicht als einzelne binäre UND-Verknüpfungen interpretiert werden können. Der verwendete Berechnungsalgorithmus auf Seite 92 könnte hierbei noch weiter formal untersucht und eventuell verbessert werden. Trotzdem sollten dabei die Kosten und Nutzen der Interpretation je nach Anwendungsfall geprüft und eventuell die Interpretation entsprechend angepasst werden.

- **POI-Modus** – im POI-Modus werden direkte Treffer im Ergebnis um ihre umgebenden RDF-Grafen erweitert. Der Benutzer kann dabei den Radius der hinzuzufügenden Umgebung selbst bestimmen. Insbesondere bei großen Radien ist die Berechnung der Umgebung sehr kostenspielig. Um Benutzer vor zu langen Wartezeiten zu bewahren, sollte eine Begrenzung des maximalen Radius vorgesehen werden.

7 Zusammenfassung

Durch das Semantische Web soll es Maschinen ermöglicht werden Metadaten zu verstehen. Hierin steckt ein enormes Potenzial, wodurch sich der Umgang mit dem heutigen Internet grundlegend ändern kann. Das Semantische Web steht jedoch noch am Anfang. Es gilt noch einige offene und strittige Punkte zu klären. Das Fundament des Semantischen Webs wird durch das Resource Description Framework (RDF) gebildet, worauf sich diese Arbeit konzentriert.

Hauptziel meiner Arbeit war die Verbesserung der Funktionalität und der Nutzungsfreundlichkeit für RDF-Speicher- und Anfragesysteme. Dabei stand die allgemeine Nutzung für ein Informationsportal oder eine Internetsuchmaschine im Vordergrund. Meine Überlegungen hierzu wurden in dem Speichersystem *RDF-Source related Storage System (RDF-S3)* und der darauf aufsetzenden Anfragesprache *easy RDF Query Language (eRQL)* umgesetzt. Insbesondere wurden die folgende Kernpunkte berücksichtigt:

- **Allgemeine Nutzbarkeit der Anfragesprache**, sodass auch unerfahrene Nutzer einfach und schnell Anfragen erstellen können.

  Um auch von unerfahrenen Nutzern bedient werden zu können, konnte keine komplexe Syntax verwendet werden, wie dies bei den meisten existierenden Anfragesprachen der Fall ist. Es wurde sich daher an Anfragesprachen existierender Suchmaschinen angelehnt. Entsprechend bilden sogenannte *Ein-Wort-Anfragen*, die den Suchbegriffen entsprechen, eine wichtige Rolle. Um gezieltere Anfragen stellen zu können, sind jedoch die Schemainformationen der gespeicherten Daten sehr wichtig. Hier bietet bereits die *RDF Query Language (RQL)* viele hilfreiche Kurzschreibweisen, an die sich eRQL anlehnt.

- **Bereitstellung glaubwürdiger Metadaten**, sodass den Anfrageergebnissen vertraut werden kann.

  Das Semantische Web ist ein verteiltes System, wobei keine Kontrolle auf die Datenquellen ausgeübt werden kann. Den Daten kann daher nicht ohne weiteres vertraut werden. Anders ist dies mit Metadaten, die von eigenen Systemen erzeugt wurden. Man weiß wie sie erzeugt wurden und kann ihnen entsprechend vertrauen. Wichtig ist eine klare Trennung zwischen den Daten und den Metadaten über diese, da sonst eine absichtliche Nachbildung der Metadaten von außen (Suchmaschinen-Spamming) das System unterlaufen kann.

  Für die Glaubwürdigkeit von Anfrageergebnissen sind vor allem die Herkunft der Daten und deren Aktualität entscheidend. In den umgesetzten Entwicklungen zu dieser Arbeit wurde sich daher auf diese Informationen konzentriert.

  In RDF-S3 wird die Verknüpfung der RDF-Aussage mit ihren Herkunftsdaten im Speichermodell abgebildet. Dies ermöglicht eine gezielte Ausnutzung dieser Daten in eRQL-Anfragen. Durch den sogenannten *Dokumenten-Modus* bietet eRQL die Möglichkeit Anfragen auf eine Gruppe von Quellen zu begrenzen oder bestimmte unglaubwürdige Quellen auszuschließen. Auch können die Herkunftsdaten das Anfrageergebniss erweitern und dadurch das Verständnis und die Glaubwürdigkeit für das Ergebnis erhöhen.

- **Anfrageergebnisse können um ihre Umgebung erweitert werden**, sodass sie besser verstanden werden können.

  Für eRQL-Anfragen besteht die Möglichkeit die Umgebnung zu den Treffern (RDF-Aussagen) mit zu berücksichtigen und im Ergebnis mit anzuzeigen. Dies erhöht das Verständnis für die Ergebnisse. Weiterhin ergeben sich hierdurch neue Möglichkeiten wie das Auffinden von Pfaden zwischen Teilergebnissen einer Anfrage.

- **Unterstützung und Kombination von Daten- und Schemaanfragen**.

  Mit eRQL werden beide Anfragetypen unterstützt und können sinnvoll miteinander kombiniert werden. Die Einbeziehung der Umgebung ermöglicht für die Kombination von Daten- und Schemaanfragen neue Möglichkeiten. Dabei werden sowohl Daten- als auch Schemaanfragen (oder deren Kombination) durch das Speichermodell von RDF-S3 optimal unterstützt.

Weitere nennenswerte Eigenschaften von RDF-S3 und eRQL sind:

- Durch die Möglichkeit gezielt einzelne Quellen wieder zu entfernen oder zu aktualisieren, bietet RDF-S3 eine gute Wartbarkeit der gespeicherten Daten.
- RDF-S3 und eRQL sind zu 100 % in Java entwickelt, wodurch ihr Einsatz unabhängig vom Betriebssystem möglich ist.
- Der Datenbankzugriff erfolgt über JDBC, wobei keine besonderen Eigenschaften für die verwendete RDBMS nötig sind[71]. Dies sorgt für eine hohe Portabilität.

RDF-S3 und eRQL wurden als Beispielimplementierungen entwickelt. Für einen produktiven Einsatz sollten die Systeme an die gegebene Hardware-Umgebung und Anwendungsfall angepasst werden. In Kapitel 6 werden Erweiterungen und Änderungsmöglichkeiten genannt, die je nach Situation geprüft werden sollten.

Ein noch vorhandenes Problem für einen produktiven Einsatz auf großen Datenmengen ist die aufwendige Berechnung der Umgebungen für Anfrageergebnisse. Die Berechnung von Umgebungen im Vorhinein könnte hier eine Lösung sein, die jedoch durch die Möglichkeit der Einschränkung auf glaubwürdige Quellen erschwert wird.

Augenblickliche Entwicklung am W3C

Die augenblickliche Entwicklung am W3C zeigt, dass es auch dort den Trend zum Umgang mit externen Kontextinformationen gibt. Anfang 2004 wurde die *W3C RDF Data Access Working Group (DAWG)* gegründet. Im Oktober 2004 wurde ein erstes Arbeitspapier (Working Draft) für eine neue RDF-Anfragesprache, namens SPARQL, veröffentlicht [W3C Sparql 04]. SPARQL ist dabei stark an RDQL angelehnt und gilt mitlerweile als offizeller Nachfolger von RDQL. Wohl nicht zuletzt, da Andy Seaborne einer der Autoren ist, der an der Entwicklung von RDQL und dessen Vorgänger SquishQL [MSR02] maßgeblich beteiligt war. Im dritten Arbeitspapier vom 19. April 2005 [W3C Sparql 05a], wurde für SPARQL der Umgang mit *RDF Datasets* eingefügt. Ein RDF Dataset ist eine Menge von RDF-Grafen, die über eine URI identifiziert werden können (*Named Graphs*). Weiterhin existiert ein *Hintergrund Graf (Background Graph)*, der beispielsweise Metadaten über die Grafen aufnehmen kann. Dies entspricht der Trennung von Daten und Metadaten, wie sie in RDF-S3 vorgenommen wurde. Mittlerweile wurde die fünfte Version des Arbeitspapieres vom 23. November 2005 [W3C Sparql 05a] veröffentlicht.

Verglichen mit SPARQL ist eRQL restriktiver, auch da die Interpretation der Kontextknoten nicht offen gehalten wird, sondern als Herkunftsinformation feststeht. Diese Einschränkung hat jedoch auch seine gute Seite, da dies Sicherheit im Umgang bietet. Weiterhin ist in SPARQL vorgesehen Anfragen an den Hintergrundgrafen stellen zu können. Dies ist vergleichbar mit der Erweiterung des Dokumenten-Modus von eRQL um Bedingungen für die Quellenmenge, die in 6.2 beschrieben ist.

[71] Die Lauffähigkeit der Systeme wurde mit DB2 UDB v8.1 und MySQL v4.1 in Kombination mit dem MySQL Connector/J v3.1.6 getestet.

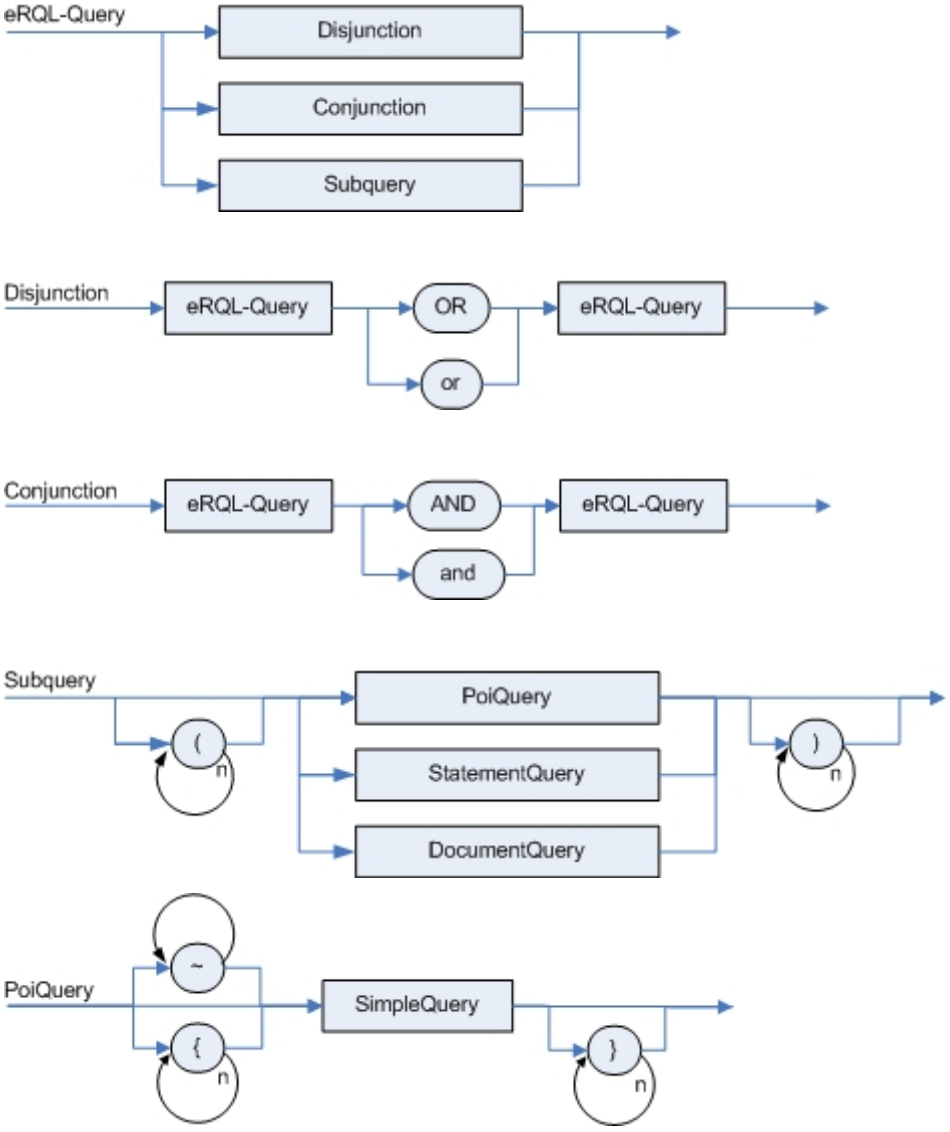
8 Anlagen

Als Anlagen sind angefügt:

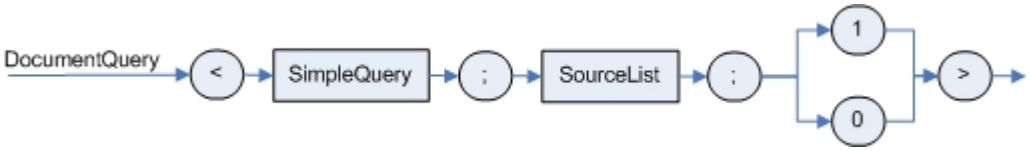
- 8.1 Syntaxbeschreibung von eRQL
- 8.2 Schnellübersicht der Klassenarchitektur für RDF-S3
- 8.3 Schnellübersicht der Klassenarchitektur für eRQL
- 8.4 Installation und Start von RDF-S3 und eRQL
- 8.5 RDF-S3-Namensraum

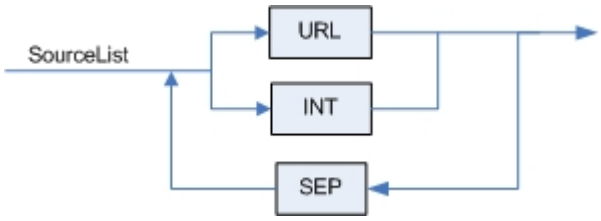
8.1 Syntaxbeschreibung von eRQL

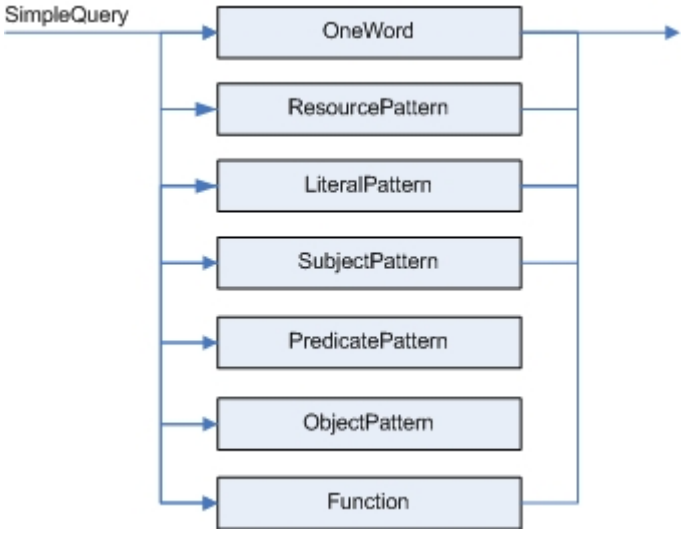
Um die Verzweigungen besser verdeutlichen zu können, werden einige Produktionen als Syntaxdiagramme grafisch dargestellt. Andere Produktionen, insbesondere für die verschiedenen Schreibweisen der Funktionen, werden am Ende unter Verwendung von BNF beschrieben.

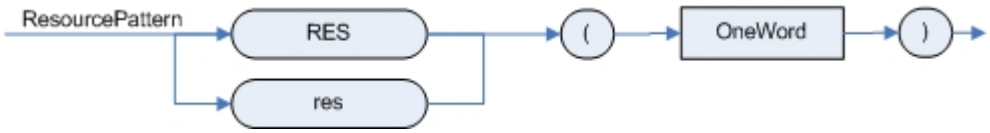


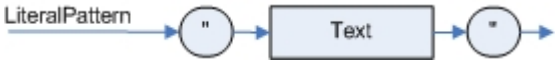

StatementQuery
n
[
SimpleQuery
n
]

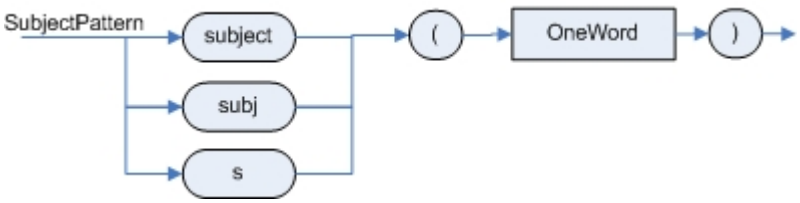

DocumentQuery
<
SimpleQuery
;
SourceList
;
1
0
>

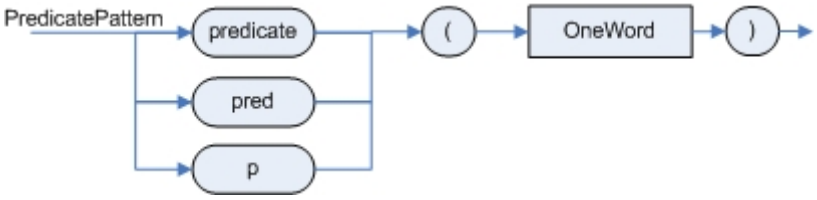

SourceList
URL
INT
SEP

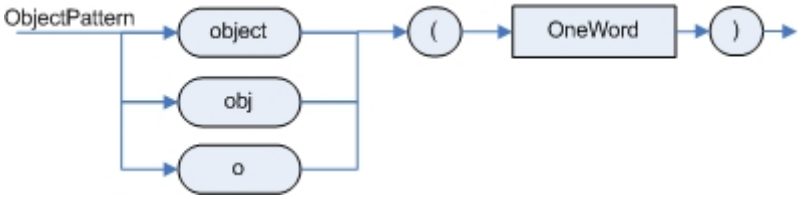

SimpleQuery
OneWord
ResourcePattern
LiteralPattern
SubjectPattern
PredicatePattern
ObjectPattern
Function

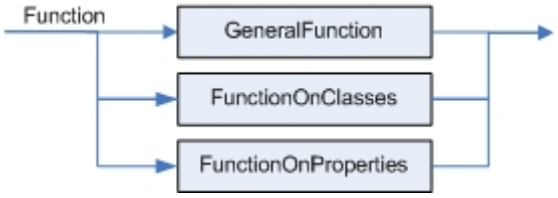

ResourcePattern
RES
res
(
OneWord
)

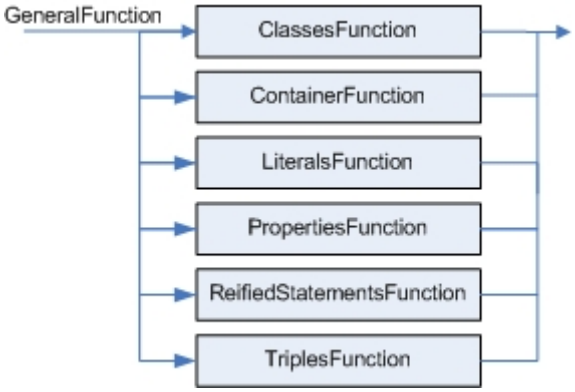

LiteralPattern
"
Text
"


SubjectPattern
subject
subj
s
(
OneWord
)









ClassesFunction ::= "classes()" | "c()" | "C()"

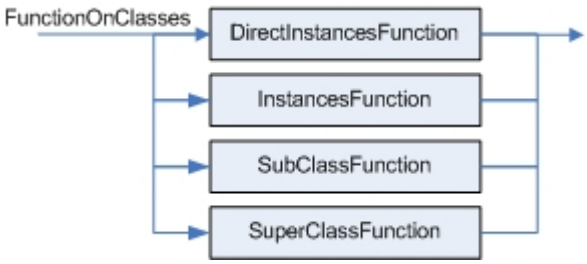
ContainerFunction ::= "container()" | "con()" | "CON()"

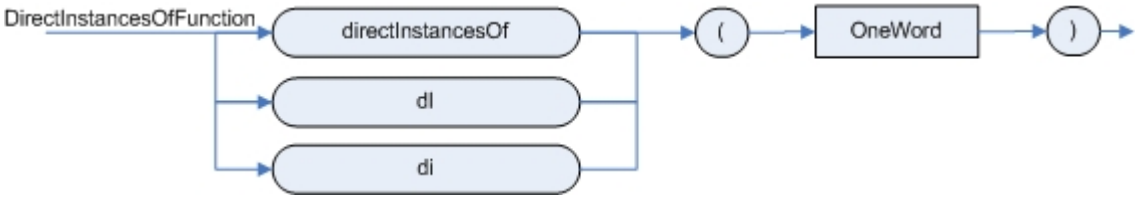
LiteralsFunction ::= "literals()" | "l()" | "L()"

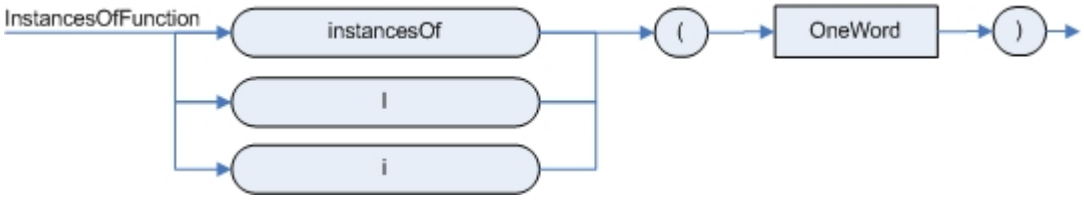
PropertiesFunction ::= "properties()" | "p()" | "P()"

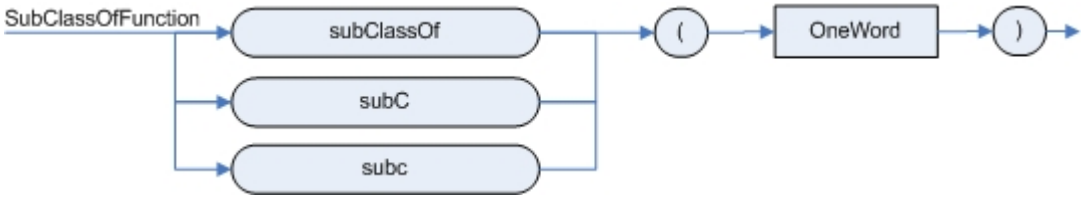
ReifiedStatementsFunction ::= "reifiedStatements()" | "rs()" | "RS()"

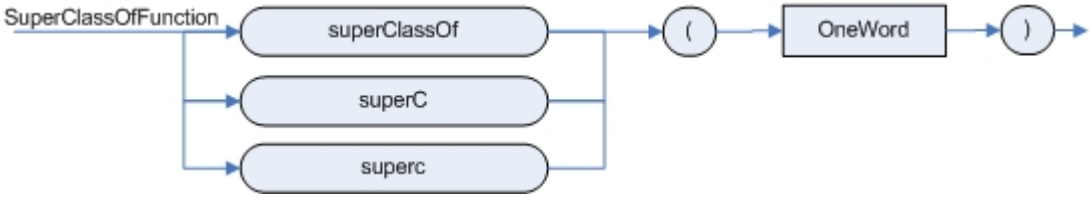
TriplesFunction ::= "triples()" | "t()" | "T()"

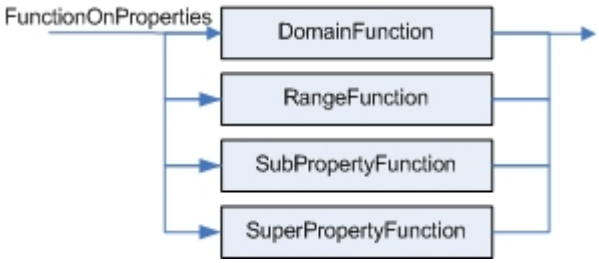

FunctionOnClasses
DirectInstancesFunction
InstancesFunction
SubClassFunction
SuperClassFunction

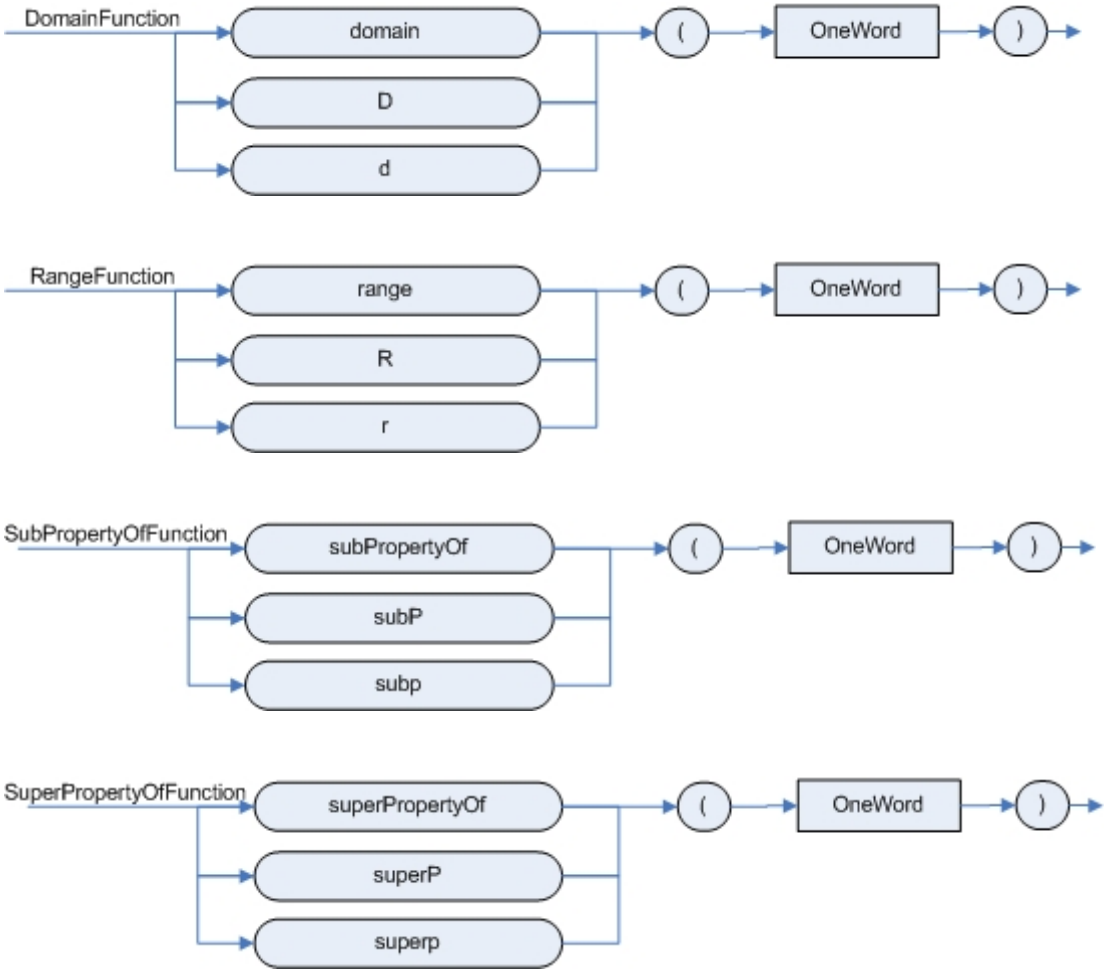

DirectInstancesOfFunction
directInstancesOf
dI
di
(
OneWord
)


InstancesOfFunction
instancesOf
I
i
(
OneWord
)


SubClassOfFunction
subClassOf
subC
subc
(
OneWord
)


SuperClassOfFunction
superClassOf
superC
superc
(
OneWord
)


FunctionOnProperties
DomainFunction
RangeFunction
SubPropertyFunction
SuperPropertyFunction



OneWord ::= [^ ;()\[\]{}<>~\"\t\n]+

Text ::= [^\n\r\"\\]+

URL ::= &?[a-zA-Z]{3,10}"://"[^]+

INT ::= [0-9]*

SEP ::= WhiteSpace | [,]

WhiteSpace ::= {LineTerminator} | [\t\f]

LineTerminator ::= \r|\n|\r\n

8.2 Schnellübersicht der Klassenarchitektur für RDF-S3

In Abbildung 56 und Abbildung 57 werden die Pakete und ihre Klassen für RDF-S3 aufgelistet und für die Pakete deren Aufgabe geschildert. Für Pakete die in der Arbeit näher beschrieben werden, wird auf den entsprechenden Abschnitt verwiesen. Weitere Informationen können der Java-Dokumentation[72] entnommen werden.

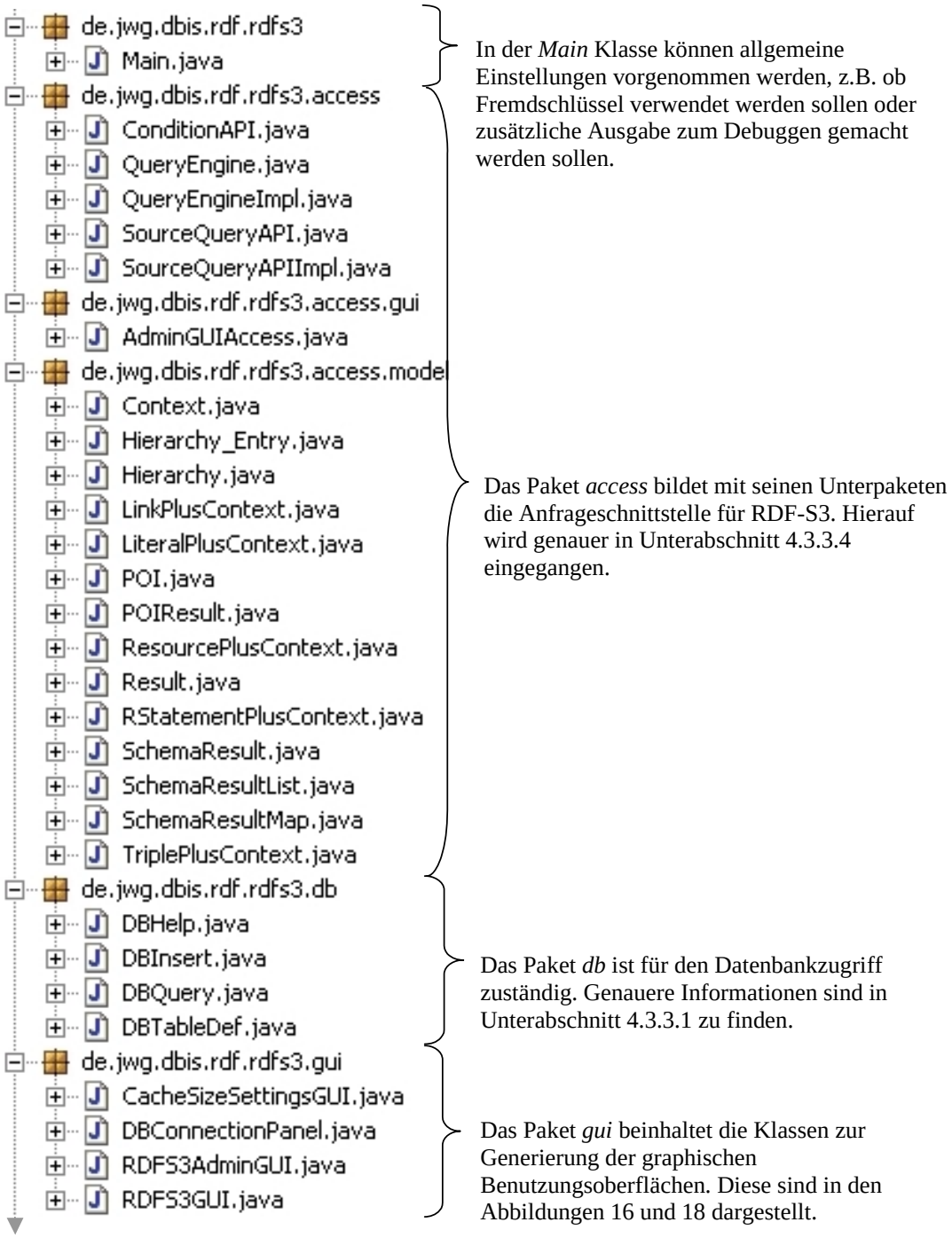


Abbildung 56 Schnellübersicht über die Klassenarchitektur von RDF-S3 (Teil 1).

[72] Die Java-Dokumentation ist online unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/RDFS3/Doc/index.html zu finden.

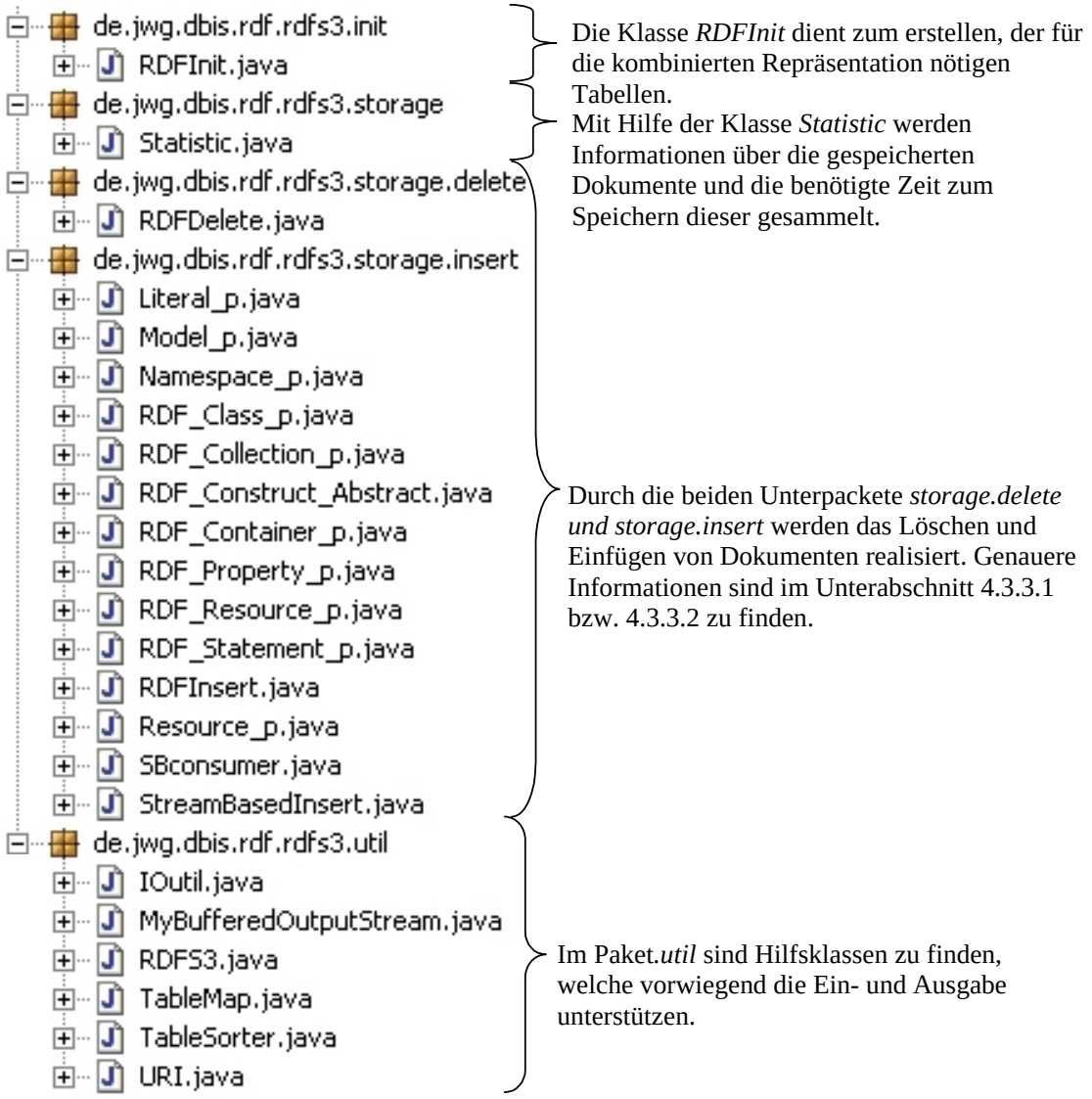


Abbildung 57 Schnellübersicht über die Klassenarchitektur von RDF-S3 (Teil 2).

8.3 Schnellübersicht der Klassenarchitektur für eRQL

In Abbildung 58 und Abbildung 59 werden die Pakete und ihre Klassen für eRQL aufgelistet und für die Pakete deren Aufgabe geschildert. Für Pakete die in der Arbeit näher beschrieben werden, wird auf den entsprechenden Abschnitt verwiesen. Weitere Informationen können der Java-Dokumentation[73] entnommen werden.

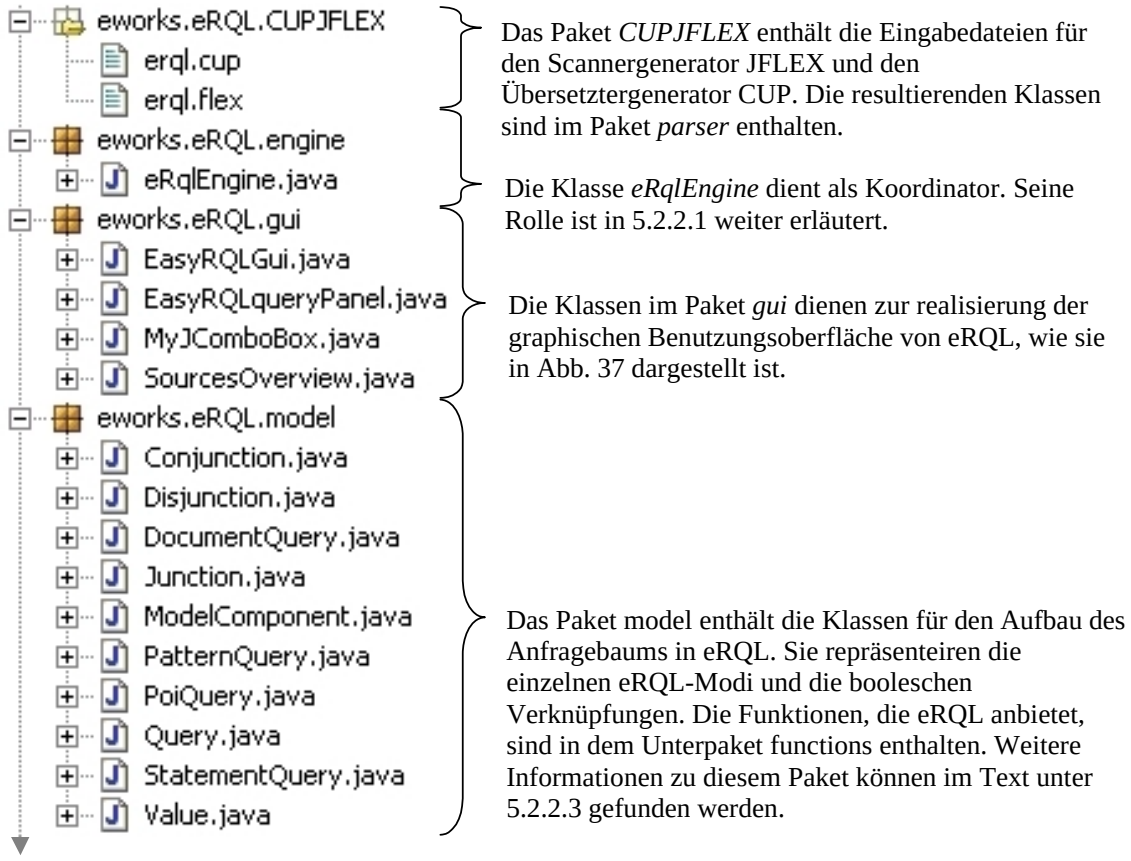


Abbildung 58 Schnellübersicht über die Klassenarchitektur von eRQL (Teil 1).

[73] Die Java-Dokumentation ist online unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/RDFS3/Doc/index.html zu finden.

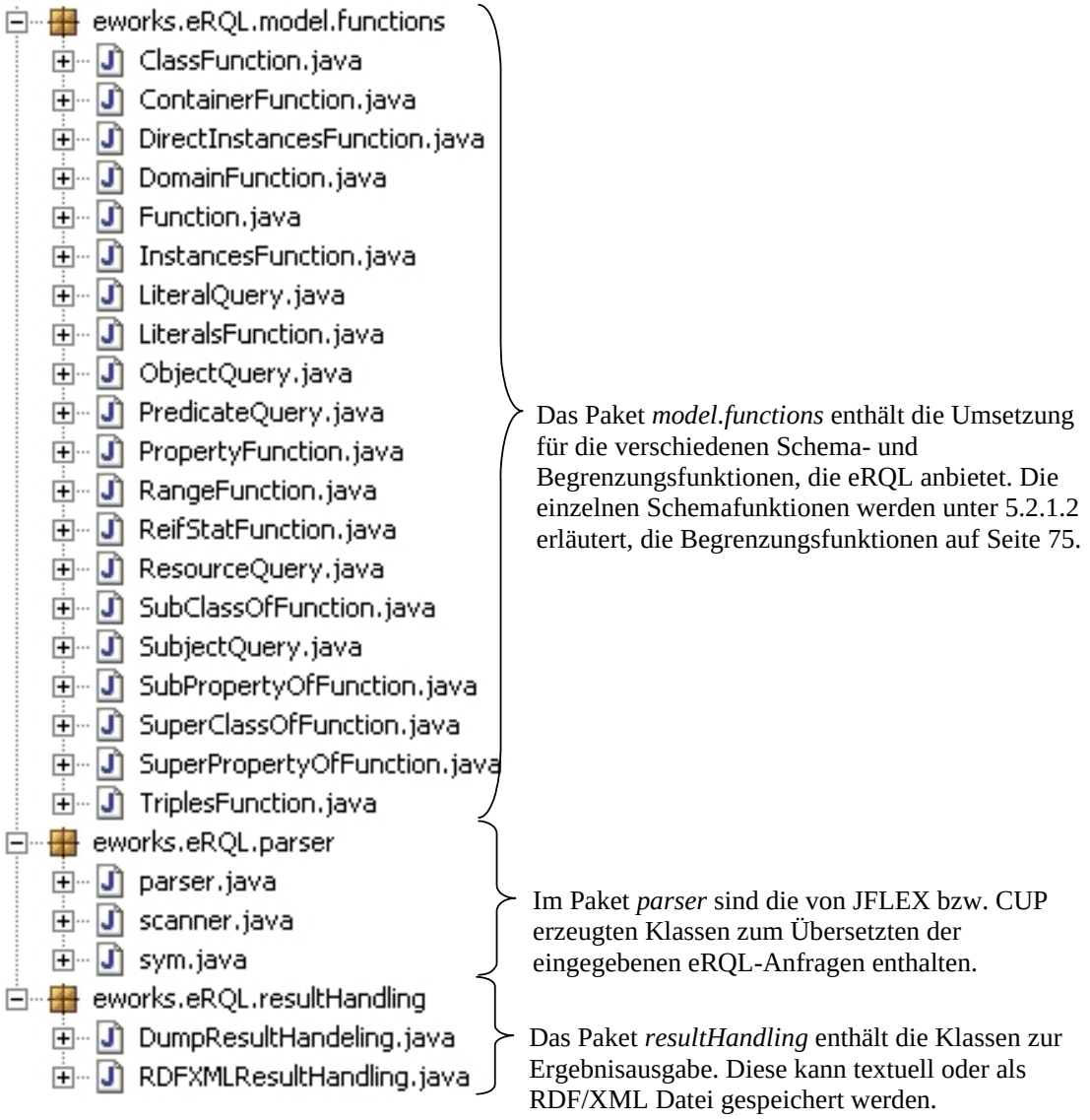


Abbildung 59 Schnellübersicht über die Klassenarchitektur von eRQL (Teil 2).

8.4 Installation und Start von RDF-S3 und eRQL

Für die Installation von RDF-S3 bitte wie folgt vorgehen:

1. Herunterladen der neusten Version von RDF-S3 unter:
   http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/RDFS3/index.html.
2. Entpacken der *ZIP*-Datei. Hierdurch wird ein neues Verzeichnis Namens *RDFS3* angelegt, welches folgenden Inhalt hat:
   - *classes/* (Verzeichnis) – enthält die ausführbaren Javaklassen.
   - *Doc/* (Verzeichnis) – enthält die Java-Dokumentation zu RDF-S3 und eRQL.
   - *src/* (Verzeichnis) – enthält die Dateien mit dem Javaquellcode.
   - *jars/* (Verzeichnis) – enthält die verschiedenen benötigten JAR-Bibliotheken.
   - *pdf/* (Verzeichnis) – enthält Hintergrundinformationen zum System und relevanten Forschungsbereichen in PDF.
   - *RDF_examples/* (Verzeichnis) – enthält einige RDF-Dateien zum Testen.

- *Manifest/* (Verzeichnis) – enthält die Manifest-Datei Manifest.mf, die in der JAR-Datei jars/rdfs3.jar integriert ist.
- *GIF/* (Verzeichnis) – enthält einige Grafiken wie Logos und Bildschirmfotos.
- *HowToUse.htm* (Datei) – eine Beschreibung zur Installation und Benutzung (engl.).
- *runRDFS3.bat* und *runRDFS3* – Skripte zum Starten der RDF-S3 Benutzungsoberfläche unter Windows bzw. Unix/LINUX. Vor dem Aufruf der müssen die Pfade für die Variablen JAVA_HOME, JDBC_Driver und RDFS3_HOME innerhalb der Skripte angepasst werden.
- *runeRQL.bat*, *paths.bat* und *runeRQL* – Skripte zum Starten der eRQL Benutzungsoberfläche unter Windows bzw. Unix/LINUX. Vor dem Aufruf der müssen die Pfade für die Variablen JAVA_HOME, JDBC_Driver und RDFS3_HOME innerhalb der Skripte angepasst werden.
- *Changes.txt* – enthält Informationen über die Entwicklung von RDF-S3 und den Neuerungen von einer Version zur nächsten.
- *.project* und *.classpath* – können genutzt werden, um RDF-S3 als Projekt in Eclipse oder in WebSphere Studio von IBM zu importieren.

3. Zum Starten der RDF-S3-Benutzungsoberfläche (Speichern von RDF-Daten):
   - Unter Windows
     i. Setzen der Variablen **JAVA_HOME**, **RDFS3_HOME** und **JDBC_Driver** im Skript *paths.bat*.
     ii. Ausführen des Skripts runRDFS3.bat, z. B. durch doppeltes Anklicken.
   - Unter Unix/Linux
     i. Setzen der Variablen **JAVA_HOME**, **RDFS3_HOME** und **JDBC_Driver** im Skript *runRDFS3*.
     ii. Ausführen des Skripts *runRDFS3*, z. B. durch die Eingabe von **source ./runRDFS3**.

4. Zum Starten der eRQL-Benutzungsoberfläche (Anfragen auf gespeicherte RDF-Daten):
   - Unter Windows
     i. Setzen der Variablen **JAVA_HOME**, **RDFS3_HOME** und **JDBC_Driver** im Skript *paths.bat* (falls noch nicht vor dem Start von RDF-S3 getan).
     ii. Ausführen des Skripts runeRQL.bat, z. B. durch doppeltes Anklicken.
   - Unter Unix/Linux
     i. Setzen der Variablen **JAVA_HOME**, **RDFS3_HOME** und **JDBC_Driver** im Skript *runeRQL*.
     ii. Ausführen des Skripts *runeRQL*, z. B. durch die Eingabe von **source ./runeRQL**.

Anmerkungen:

a) RDF-S3 und eRQL benötigen mindestens eine Java-Laufzeitumgebung der Version 1.4 oder höher.
b) Sollten größere RDF-Dateien bearbeitet werden, sollten in den aufrufenden Skripten die Parameter für die Java-Heap-Größe (-mx und -Xmx) angepasst werden.
c) Es muss ein RDBMS installiert sein, welches JDBC und SQL unterstützt. Die Pfadangabe zu dem entsprechenden JDBC-Treiber muss vor dem Start eingetragen werden (s.o. Punkt 3 bzw. 4).

8.5 RDF-S3-Namensraum

Der RDF-S3-Namensraum stellt das RDF-Vokabular bereit, welches genutzt wird, um Speicherinformationen oder eRQL-Ergebnisse in RDF darstellen zu können.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
   <!ENTITY xmldt "http://www.w3.org/2001/XMLSchema#">
]>

<!-- by Karsten Tolle, 06.01.2005 -->

<rdf:RDF
   xmlns:rdf="&rdf;"
   xmlns:rdfs="&rdfs;"
   xml:base = "http://www.dbis.informatik.uni-
frankfurt.de/~tolle/RDF/RDFS3/RDFS3.rdf#">


<!-- For statistic output of RDF-S3 -->
<rdfs:Class rdf:ID="Source">
        <rdfs:comment>Representing the sources.</rdfs:comment>
</rdfs:Class>


<rdf:Property rdf:ID="timestamp">
        <rdfs:comment>The timestamp in textual representation.</rdfs:comment>
        <rdfs:range rdf:resource="&xmldt;string"/>
</rdf:Property>


<rdf:Property rdf:ID="numberOfTriples">
        <rdfs:comment>The number of RDF triples contained in the
file.</rdfs:comment>
        <rdfs:domain rdf:resource="#Source"/>
        <rdfs:range rdf:resource="&xmldt;nonNegativeInteger"/>
</rdf:Property>


<rdf:Property rdf:ID="totalLoadingTime">
        <rdfs:comment>The total loading time (ms) RDF-S3 needed to load this
file.</rdfs:comment>
        <rdfs:domain rdf:resource="#Source"/>
        <rdfs:range rdf:resource="&xmldt;nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="deletionTime">
        <rdfs:comment>The time (in ms) RDF-S3 needed to delete a source out
of the database.</rdfs:comment>
        <rdfs:domain rdf:resource="#Source"/>
        <rdfs:range rdf:resource="&xmldt;nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="numberOfVRPErrors">
        <rdfs:comment>The number of errors VRP reported during parsing this
file.</rdfs:comment>
        <rdfs:domain rdf:resource="#Source"/>
        <rdfs:range rdf:resource="&xmldt;nonNegativeInteger"/>
</rdf:Property>


<rdf:Property rdf:ID="number">
        <rdfs:comment>The number of elements of this kind.</rdfs:comment>
        <rdfs:range rdf:resource="&xmldt;nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="time">
```

```
        <rdfs:comment>The time / duration (ms) needed for
loading.</rdfs:comment>
        <rdfs:range rdf:resource="&xmldt;nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="classes">
        <rdfs:comment>Pointing to the node containing the load information
        for classes contained in the model.</rdfs:comment>
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="#Source"/>
</rdf:Property>

<rdf:Property rdf:ID="properties">
        <rdfs:comment>PPointing to the node containing the load information
        for properties contained in the model.</rdfs:comment>
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="#Source"/>
</rdf:Property>

<rdf:Property rdf:ID="resources">
        <rdfs:comment>Pointing to the node containing the load information
        for resources contained in the model.</rdfs:comment>
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="#Source"/>
</rdf:Property>

<rdf:Property rdf:ID="container">
        <rdfs:comment>Pointing to the node containing the load information
        for container contained in the model.</rdfs:comment>
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="#Source"/>
</rdf:Property>

<rdf:Property rdf:ID="collections">
        <rdfs:comment>Pointing to the node containing the load information
        for collections contained in the model.</rdfs:comment>
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="#Source"/>
</rdf:Property>

<rdf:Property rdf:ID="reifiedStatements">
        <rdfs:comment>Pointing to the node containing the load information
        for reified statements contained in the model.</rdfs:comment>
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="#Source"/>
</rdf:Property>

<rdf:Property rdf:ID="pois">
        <rdfs:comment>Pointing to the node containing the load information
        to enter the triples to the poi-table. The poi-table represents the
        generic representation of the model.</rdfs:comment>
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="#Source"/>
</rdf:Property>

<!-- For result output of eRQL embedded in RDF-S3 -->

<rdfs:Class rdf:ID="Result">
        <rdfs:comment>An result to a eRQL query.</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Query">
        <rdfs:comment>An eRQL query.</rdfs:comment>
</rdfs:Class>
```

```
<rdf:Property rdf:ID="result">
        <rdfs:range rdf:resource="#Query"/>
        <rdfs:domain rdf:resource="#Result"/>
</rdf:Property>

<rdf:Class rdf:ID="POI">
        <rdfs:comment>Represents a POI in a result.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#Result"/>
</rdf:Class>

<rdf:Property rdf:ID="hit">
        <rdfs:range rdf:resource="#Result"/>
        <rdfs:comment>A hit of an eRQL query.</rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="directHit">
        <rdfs:comment>A direct hit of an eRQL query.</rdfs:comment>
        <rdfs:subPropertyOf rdf:resource="#hit"/>
</rdf:Property>

<rdf:Property rdf:ID="querytext">
        <rdfs:comment>The query text of the eRQL query.</rdfs:comment>
        <rdfs:domain rdf:resource="#Query"/>
        <rdfs:range rdf:resource="&xmldt;string"/>
</rdf:Property>

<rdf:Property rdf:ID="resourceHit">
        <rdfs:comment>For schema queries on resources, e.g., domain(q). Will
return the domain classes for the
 properties fitting the query q.</rdfs:comment>
        <rdfs:domain rdf:resource="#Result"/>
        <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="queryType">
        <rdfs:comment>A string representation for the query. For a better
understanding.</rdfs:comment>
        <rdfs:domain rdf:resource="&rdfs;Resource"/>
        <rdfs:range rdf:resource="&xmldt;string"/>
</rdf:Property>

<rdf:Property rdf:ID="context">
        <rdfs:comment>Contains the source information and when it was entered
to RDFS3.</rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="sourceURL">
        <rdfs:comment>The source URL for the given context.</rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="lastUpdate">
        <rdfs:comment>The timestamp when it was entered to
RDFS3.</rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="literal">
        <rdfs:comment>To denote a literal.</rdfs:comment>
</rdf:Property>

</rdf:RDF>
```

9 Literaturverzeichnis

[ACKP01] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis: On Storing Voluminous RDF Descriptions: The case of Web Portal Catalogs, 4th International Workshop on the Web and Databases (WebDB'01), May 2001.

[Ale+01] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, Karsten Tolle, The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, 2nd International Workshop on the Semantic Web, in conjunction with Tenth International World Wide Web Conference (WWW10), Hongkong, May 2001.

[BC04] Chris Bizer, Jeremy Carroll: Modelling Context using Named Graphs. SWIG Meeting, Cannes, online unter: http://lists.w3.org/Archives/Public/www-archive/2004Feb/att-0072/swig-bizer-carroll.pdf, 2004.

[Ber98] Tim Berners-Lee: Why RDF model is different from the XML model. Online unter: http://www.w3.org/DesignIssues/RDF-XML.html, 1998.

[BHL01] Tim Berners-Lee, James Hendler, Ora Lassila: The Semantic Web. Scientific American, May 2001.

[BHS03] Valerie Bönström, Annika Hinze, Heinz Schweppe: Storing RDF as a Graph. First Latin American Web Congress, LA-WEB, Chile, 2003.

[Biz05] Chris Bizer: The TriG Syntax. Working Draft, online unter: http://www.wiwiss.fu-berlin.de/suhl/bizer/TriG/Spec/TriG-20050606/, 2005.

[BK02] Jeen Broekstra, Arjohn Kampman: Sesame: A generic Architecture for Storing and Querying RDF and RDF Schema. In Proceedings of the First International Semantic Web Conference (ISWC 2002), Sardinia, Italy, pg. 54-68. Springer-Verlag Lecture Notes in Computer Science (LNCS) no. 2342, 2002.

[BM02] Tim Berners-Lee, Eric Miller: The Semantic Web lifts off. ERCIM News Nr. 51, online unter: http://www.ercim.org/publication/Ercim_News/enw51/berners-lee.html, Oktober 2002.

[Bon04] Elena Paslaru Bontas: Representing Context on the Semantic Web. Doktorandenworkshop Technoloigien und Anwendungen von XML, Berliner XML Tage, 2004.

[Bro99] Annie Brooking: Corporate Memory. Thomson Business Press, ISBN 1-86152-268-1, 1999.

[Cat+00] Roderick G. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, Fernando Velez: The Object Database Standard: ODMG 3.0. Morgan Kaufmann Publishers, ISBN 1-55860-647-4, 2000.

[CBHS04] Jeremy Carroll, Chris Bizer, Patrick Hayes, Patrick Stickler: Named Graphs, Provenance and Trust. Technical Report HPL-2004-57R1, online unter: http://www.hpl.hp.com/techreports/2004/HPL-2004-57R1.html, 2004.

[CPST03] Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, Sotirios Tourtounis: On Labeling Schemes for the Semantic Web. 12th International World Wide Web Conference (WWW'03), Mai 2003.

[CS04] Jeremy Carroll, Patrick Stickler: TriX: RDF Triples in XML. Technical Report HPL-2003-268, HP Labs, 2004.

[Dec+00] Stefan Decker, Frank van Harmelen, Jeen Broekstra, Michael Erdmann, Dieter Fensel, Ian Horrocks, Michel Klein, Sergey Melnik : The Semantic Web - on the respective Roles of XML and RDF, IEEE Internet Computing vol. 4 (5) pp. 63-74, Sept./Oct. 2000.

[Dec02] Stefan Decker: Semantic Web and Databases: Relationships and some Open Problems. NSF-OntoWeb Invitational Workshop on DB-IS Research for Semantic Web and Enterprises, 2002.

[Doc01] Cory Doctorow: Metacrap: Putting the torch to seven straw-men of the meta-utopia. Online unter: http://www.well.com/~doctorow/metacrap.htm, 2001.

[EE04] Rainer Eckstein, Silke Eckstein: XML und Datenmodellierung – XML-Schema und RDF zur Modellierung von Daten und Metadaten einsetzen. Erschienen im dpunkt.verlag, ISBN 3-89864-222-4, 1. Auflage 2004.

[End04] Bartholomäus Ende: Formale Betrachtung von Anfragen auf RDF Datenbanken. Seminararbeit an der Johann Wolfgang Goethe-Universität Frankfurt am Main, DBIS, Februar 2004.

[FHLW03] Dieter Fensel, James Hendler, Henry Lieberman, Wolfgang Wahlster (Hrsg.): Spinning the Semantic Web. MIT Press, ISBN 0-262-06232-1, 2003.

[FHVB04] Flavius Frasincar, Geert-Jan Houben, Richard Vdovjak, Peter Barna: RAL: An Algebra for Querying RDF. WWW Journal, online unter: http://wwwis.win.tue.nl/~rvdovjak/publications/RALwwwj2004.pdf, March 2004.

[Fra01] Mark Frauenfelder: A Smarter Web. Technology Review, online unter: http://www.technologyreview.com/InfoTech/wtr_12640,258,p1.html, November 2001.

[Gar02] Lars Marius Garshol: Topic maps, RDF, DAML, OIL - A comparison. Ontopia, online unter: http://www.ontopia.net/topicmaps/materials/tmrdfoildaml.html, 2002.

[Goo04] John Goodson: Performance Tips for the Data Tier (JDBC) Part III: Selecting Functions that Optimize Performance. Online Artikel unter: http://www.theserverside.com/articles/content/JDBCPerformance_PartIII/article.html, 2004.

[Gro03] Jan Große: Speicherverfahren und Werkzeuge für RDF/S. XML Clearinghouse Report, Robert Tolksdorf und Rainer Eckstein (Hrsg.), Dezember 2003.

[Guh91] Ramanathan V. Guha: Contexts: A Formalization and Some Applications. Dissertationsschrift an der Stanford Universität, 1991.

[HBEV04] Peter Haase, Jeen Broekstra, Andreas Eberhart, Raphael Volz: A comparison of RDF query languages. In Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, November 2004.

[Hef01] Jeff Heflin: Towards the Semantic Web: Knowledge representation in a dynamic, distributed environment. Dissertation, University of Maryland, 2002.

[Heß02] Martin Heß: Verteiltes Information Retrieval für nicht-kooperative Suchserver im WWW. Dissertationsschrift an der Johann Wolfgang Goethe-Universität Frankfurt am Main, Juni 2002.

[HHL99] Jeff Heflin, James Hendler, Sean Luke: Coping with Changing Ontologies in a Distributed Environment. AAAI Conference Ontology Management Workshop, 1999.

[HMD00] Martin Heß, Christian Mönch, Oswald Drobnik: QUEST - Querying Specialized Collections on the Web. In Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Lissabon, September 2000.

[Hje01] Johan Hjelm: Creating the Semantic Web with RDF. Wiley, ISBN 0-471-40259-1, 2001.

[Jan93] Bob Jansen: Context: A real problem for large and shareable knowledge bases, Building/Sharing Very Large Knowledge Bases (KBKS'93), Tokyo, December 1993.

[Kar+03] Gregory Karvounarakis, Aimilia Magkanaraki, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, Karsten Tolle: Querying the Semantic Web with RQL, Computer Networks and ISDN Systems Journal, Vol. 42(5), August 2003.

[Kar+04] Gregory Karvounarakis, Aimilia Magkanaraki, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, Karsten Tolle: RQL: A Functional Query Language for RDF. Kapitel 18 in: Functional Approaches to Computing With Data, P.M.D.Gray, L.Kerschberg, P.J.H.King, A.Poulovassilis (Hrsg.), LNCS Series, Springer-Verlag ISBN 3-540-00375-4, 2004.

[Kar04] Christoph Karwoth: Semantic Web & Caché - Objektorientierte Speicherung des RDF-Modells in die postrelationale Datenbank Caché von InterSystems. Diplomarbeit an der Johann Wolfgang Goethe-Universität Frankfurt am Main, DBIS, Dezember 2004.

[KC02] Gregory Karvounarakis, Vassilis Christophides: RQL v2.1 User Manual. Online unter: http://139.91.183.30:9090/RDF/RQL/Manual.html, 2003.

[KF01] Michel Klein, Dieter Fensel: Ontology versioning on the Semantic Web. International Semantic Web Workshop Symposium (SWWS), USA, 2001.

[Kly00] Graham Klyne: Contexts for RDF Information Modelling. Online unter: http://www.ninebynine.org/RDFNotes/RDFInfoModelling.pdf, October 2000.

[Kly02] Graham Klyne: Circumstance, provenance and partial knowledge. Online unter: http://www.ninebynine.org/RDFNotes/UsingContextsWithRDF.html, 2002.

[KR03] Joseph B. Kopena, William C. Regli: Design Repositories for the Semantic Web with Description-Logic Enabled Services. First International Workshop on Semantic Web and Databases, Sept. 7-8 2003.

[Mag+02] Aimilia Magkanaraki, Grigoris Karvounarakis, Ta Tuan Anh, Vassilis Christophides, Dimitris Plexousakis: Ontology Storage and Querying. ICS-FORTH, Technical Report No. 308, 2002.

[McB02] Brian McBride: The Jena Semantic Web toolkit. Jena Project, Hewlett-Packard Laboratories (HPL), O'Reilly xml.com, http://www.xml.com/pub/r/1285, 2002.

[McC93] John McCarthy: Notes on Formalizing Context. Proceedings of IJCAI, online unter: http://www-formal.stanford.edu/jmc/context.html, 1993.

[MK03] Robert MacGregor, In-Young Ko: Representing Contextualized Data using Semantic Web Tools. In Practical and Scalable Semantic Systems (Workshop bei der 2. ISWC), 2003.

[Mos02] Marie-Luise Moschgath: Kontextabhängige Zugriffskontrolle für Anwendungen im Ubiquitous Computing. Dissertationsschrift an der Universität Darmstadt, Mai 2002.

[MSR02] Libby Miller, Andy Seaborne, Alberto Reggiori: Three Implementations of SquishQL, a Simple RDF Query Language. Proceedings of the First International Semantic Web Conference (ISWC), 2002.

[NK03] Natayl F. Noy, Michel Klein: Ontology Evolution: Not the Same as Schema Evolution. Knowledge and Information Systems, 2003.

[OR23] Charles Kay Ogden, Ivor Armstrong Richards: The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism. Routledge and Kegan Paul Ltd. London, Auflage von 1989 mit einer Einleitung von Umberto Eco, ISBN 0-15-658446-8, erste Veröffentlichung 1923.

[Poo02] Krishnakumar Pooloth: High Insert Performance on DB2 UDB EEE using Java. Comfactory SETLabs, copyright by Infosys Technologies Ltd., 2002.

[Pow03] Shelley Powers: Praktical RDF. O'Reilly, ISBN 0-596-00263-7, July 2003.

[PS03] Steve Pepper, Silvia Schwab: Curing the Web's Identity Crises. XML Europe 2003, online unter: http://www.ontopia.net/topicmaps/materials/identitycrisis.html, 2003.

[RCDS01] Dave Reynolds, Steve Crayzer, Ian Dickinson, Paul Shabajee: SWAD-Europe Deliverable 12.1.1: Semanic web applications – analysis and selection. Online: http://www.w3.org/2001/sw/Europe/reports/chosen_demos_rationale_report/hp-applications-selection.html, 2001.

[Rex05] Jashar Rexhepi: Performance Optimierung der Nutzung von RDF-S3 Daten am Beispiel von IBMs DB2 v8.1. Diplomarbeit an der Johann Wolfgang Goethe-Universität Frankfurt am Main, DBIS, März 2005.

[Rus48] Bertrand Russell: Human Knowledge, its Scope and Limits. Routledge London, Auflage von 1992, ISBN: 0-415-08302-8, erste Veröffentlichung 1948.

[Sch05] Holger Schmidt: Googles Superhirn. Frankfurter Allgemeine Zeitung (FAZ), Netzwirtschaft & Medien, Nr. 31 / Seite 19, Februar 2005.

[Sea02] Andy Seaborne: Jena Tutorial - A Programmer's Introduction to RDQL. Online unter: http://jena.sourceforge.net/tutorial/RDQL/, 2002.

[Sea04] Andy Seaborne: RDQL - A Query Language for RDF. W3C Member Submission, online unter: http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/, January 2004.

[Sha90] Benny Shanon: What is Context. Jornal for the Theory of Social Behaviour, 20, 158-166, 1990.

[SS99] Martin Schader, Lars Schmidt-Thieme: Java - Eine Einführung. Zweite Auflage, Springer Verlag, ISBN 3-540-65716-9, 1999.

[Sta04] Titus Stahl: On Labeling Schemes for the Semantic Web. Seminararbeit an der Johann Wolfgang Goethe-Universität Frankfurt am Main, DBIS, 2004.

[TAC03] Karsten Tolle, Sofia Alexaki, Vassilis Christophides: The VRP v2.5: Implementing the Updated RDF Syntax. W3C Report, online unter: http://www.w3.org/2003/03/for-rdf.htm, März 2003.

[TC00] Karsten Tolle, Vassilis Christophides: ICS - VRP: a Tool for Parsing and Validating RDF Metadata & Schemas. ERCIM News No.41, online unter: http://www.ercim.org/publication/Ercim_News/enw41/tolle.html, April 2000.

[Tol00] Karsten Tolle: Analyzing and Parsing RDF. Diplomarbeit in Zusammenarbeit der Universität Hannover und der University of Crete, 2000.

[Tol01] Karsten Tolle: Comparing Namespaces for XML and RDF. Arbeitspapier, online unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/Publications/NspXMLRDF.htm, September 2001.

[Tol04] Karsten Tolle: Understanding data by their context using RDF. AISTA'04: International Conference on Advances in Intelligent Systems - Theory and Applications in cooperation with IEEE Computer Society, Centre de Recherche Public Henri Tudor, Luxembourg, 15-18 November 2004.

[Tol05] Karsten Tolle: easy RDF Query Language (eRQL) v1.8 Tutorial. Online unter: http://www.dbis.informatik.uni-frankfurt.de/~tolle/Publications/eRQLTutorial_07.04.2005.pdf, 2005.

[TW04a] Karsten Tolle, Fabian Wleklinski: RDF-S3 und eRQL: RDF Technologien für Informationsportale, Semantische Technologien für Informationsportale, Workshop im Rahmen der GI Jahrestagung in Ulm, 2004.

[TW04b] Karsten Tolle, Fabian Wleklinski: Trust and Context using the RDF-Source related Storage System (RDF-S3) and easy RQL (eRQL). XSW2004 – XML-Technologien für das Semantic Web, Workshop im Rahmen der Berliner XML Tage 2004.

[W3C Concept 04] Graham Klyne, Jeremy J. Carroll: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, online unter: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 10 Februar 2004.

[W3C LC 03] Last Call Comments on 23 Jan 2003 RDF Working Drafts, online unter: http://www.w3.org/2001/sw/RDFCore/20030123-issues/, 2003.

[W3C M&S 99] Ora Lassila, Ralph R. Swick: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, online unter: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, 22 Februar 1999.

[W3C Primer 04] Frank Manola, Eric Miller: RDF Primer. W3C Recommendation, online unter: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 10 Februar 2004.

[W3C Primer 03] Frank Manola, Eric Miller: RDF Primer. W3C Working Draft, online unter: http://www.w3.org/TR/2003/WD-rdf-primer-20030123/, 23 Januar 2003.

[W3C RDF-TM 05] Steve Pepper, Fabio Vitali, Lars Marius Garshol, Nicola Gessa, Valentina Presutti: A Survey of RDF/Topic Maps Interoperability Proposals. W3C Working Draft, online unter: http://www.w3.org/TR/2005/WD-rdftm-survey-20050329/, 29 März 2005.

[W3C Schema 04] Dan Brickley, R.V. Guha: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, online unter: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/, 10 Februar 2004.

[W3C Semantic 04] Patrick Hayes: RDF Semantics. W3C Recommendation, online unter: http://www.w3.org/TR/2004/REC-rdf-mt-20040210/, 10 Februar 2004.

[W3C SPARQL 04] Eric Prud´hommeaux, Andy Seaborne: SPARQL Query Language for RDF. W3C Working Draft, online unter: http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/, 12. Oktober 2004.

[W3C SPARQL 05a] Eric Prud´hommeaux, Andy Seaborne: SPARQL Query Language for RDF. W3C Working Draft, online unter: http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050419/, 19. April 2005.

[W3C SPARQL 05b] Eric Prud´hommeaux, Andy Seaborne: SPARQL Query Language for RDF. W3C Working Draft, online unter: http://www.w3.org/TR/2005/WD-rdf-sparql-query-20051123/, 23. November 2005.

[W3C Syntax 04] Dave Beckett: RDF/XML Syntax Specification (Revised). W3C Recommendation, online unter: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, 10 Februar 2004.

[W3C Syntax 02] Dave Beckett: RDF/XML Syntax Specification (Revised). W3C Working Draft, online at: http://www.w3.org/TR/2002/WD-rdf-syntax-grammar-20021108, 08. November 2002.

[W3C Tracking] W3C RDF Issue Tracking, online unter: http://www.w3.org/2000/03/rdf-tracking/

[Wei66] Harald Weinrich: Linguistik der Lüge. Sechste erweiterte Auflage, 2000, ISBN 3-406-45912-9, erste Auflage 1966.

[Wle03] Fabian Wleklinski: Suche im Semantic Web – Erweiterung des VRP um eine intuitive und RQL-basierte Anfrageschnittstelle. Diplomarbeit an der Johann Wolfgang Goethe-Universität Frankfurt am Main, DBIS, November 2003.

[WLS03] Timo Weithoener, Thorsten Liebig,, Guenther Specht: Storing and Querying Ontologies in Logic Databases, First International Workshop on Semantic Web and Databases, Sept. 7-8 2003.

[WSKR03] Kevin Wilkinson, Craig Sayers, Harumi Kuno, Dave Reynolds: Efficient RDF Storage and Retrieval in Jena2. First International Workshop on Semantic Web and Databases, Sept. 7-8 2003.

[WSW03] Jan Wielemaker, Guus Schreiber, Bob Wielinga: Prolog-based Infrastructure for RDF: Scalability and Performance. D. Fensel, K. Sycara and J. Mylopoulos (Hrsg.), The Semantic Web - Proceedings ISWC'03, Sanibel Island, Florida. Lecture Notes in Computer Science, volume 2870, pp. 644-658. Berlin/Heidelber, Springer-Verlag, 2003.

[Xu04] Jihua Xu: Evaluierung von Java RDF Parsern. Diplomarbeit an der Johann Wolfgang Goethe-Universität Frankfurt am Main, DBIS, Januar 2004.

[Zan+97] Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard T. Snodgrass, V. S. Subrahmanian, Roberto Zicari: Advanced Database Systems. Morgan Kaufmann, ISBN: 1-55860-443-X, 1997.

[Zie02] Cai Ziegler: Deus ex Machina – Das Web soll lernen, sich und uns zu verstehen. c’t Report Heft 6, 2002.

10 Lebenslauf

Zur Person:

Dipl.-Math. Karsten Tolle

geboren am 23.03.1971 in Korbach

E-Mail: tolle@dbis.informatik.uni-frankfurt.de

Lebenslauf:

08/1977 – 07/1981	Grundschule in Hannover
08/1981 – 07/1983	Orientierungsstufe in Hannover
08/1983 – 05/1991	Gymnasium Goetheschule, Hannover
27.05.1991	Abschluss zur allgemeine Hochschulreife
10/1991 – 09/1993	Bundeswehrdienst, als Soldat auf Zeit zur Ausbildung zum Reserve Offizier Anwärter (SAZ 2 / ROA) (01.10.1994 Ernennung zum Leutnant der Reserve)
10/1993 – 04/2000	Studium an der Universität Hannover, Fachbereich Mathematik, Studiengang: *Mathematik mit der Studienrichtung Informatik* Anwendungsfach: *Betriebswirtschaftslehre*
05/1999 – 10/1999	Auslandsaufenthalt auf Kreta (Griechenland) am ICS-FORTH, im Rahmen meiner Diplomarbeit mit dem Thema: „Analysing and Parsing RDF“ – Betreut von Prof. Dr. Vassilis Christophides, Gutachter: Prof. Dr. Wolfgang Nejdl und Prof. Dr. Udo Lipeck
01.04.2000	Abschluss des Studiums als *Diplom-Mathematiker*
05/2000 – 04/2005	Wissenschaftlicher Mitarbeiter (Doktorand) an der Professur Datenbanken und Informationssysteme (DBIS) von Herrn Prof. Dott.-Ing. Roberto V. Zicari an der Johann Wolfgang Goethe-Universität Frankfurt am Main
seit 06/2005	Selbstständige Tätigkeit – unter anderem in einem Projekt bei EMC^2
seit 01/2006	Wissenschaftlicher Mitarbeiter und Projektleiter für das EU-Projekt ABILITIES an der Professur Datenbanken und Informationssysteme (DBIS) von Herrn Prof. Dott.-Ing. Roberto V. Zicari an der Johann Wolfgang Goethe-Universität Frankfurt am Main

Liste eigener begutachteter Veröffentlichungen:

- **Understanding data by their context using RDF**, Karsten Tolle, International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA) in Kooperation mit der IEEE Computer Society, Luxembourg, November 15-18, 2004.

- **Trust and context using the RDF-Source related Storage System (RDF-S3) and easy RQL (eRQL)**, Karsten Tolle, Fabian Wleklinski. XML-Technologien für das Semantic Web im Rahmen der Berliner XML Tage 2004, Germany, Oktober 2004.
- **RDF-S3 und eRQL: RDF Technologien für Informationsportale**, Karsten Tolle, Fabian Wleklinski. Informatik 2004 Workshop über Semantische Technologien für Informationsportale, im Rahmen der 34. GI Jahrestagung 2004, Ulm, Germany.
- **Building and Evaluating Non-Obvious User Profiles for Visitors of Web Sites**, Naveed Mushtaq, Karsten Tolle, Peter Werner and Roberto Zicari. IEEE Conference on E-Commerce Technology (CEC 04) July 6-9, 2004, San Diego, California, USA.
- **RQL: A Functional Query Language for RDF**, Gregory Karvounarakis, Aimilia Magkanaraki, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, Karsten Tolle, Functional Approaches to Computing With Data, P.M.D.Gray, L.Kerschberg, P.J.H.King, A.Poulovassilis (eds.), LNCS Series, Springer-Verlag, 2004.
- **Querying the Semantic Web with RQL**, Gregory Karvounarakis, Aimilia Magkanaraki, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, Karsten Tolle, Computer Networks and ISDN Systems Journal, Elsevier Science, 2003.
- **Agent-Based Services for Information Portals**, Ciaran Bryce, Michel Pawlak, Karsten Tolle, Peter Werner and Roberto Zicari. The Eighteenth Annual ACM Symposium on Applied Computing March 9 to 12, 2003, Melbourne, Florida, USA.
- **The ICS-FORTH RDFSuite: High-level Scalable Tools for the Semantic Web**, Sofia Alexaki, Nikos Athanasi, Vassilis Christophides, Gregory Karvounarakis, Aimilia Maganaraki, Dimitris Plexousakis, Karsten Tolle. The Eleventh International World Wide Web Conference (WWW2002), Honolulu, Hawaii, USA, May 7-11, 2002.
- **The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases**, Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, Karsten Tolle, 2nd International Workshop on the Semantic Web, in conjunction with Tenth International World Wide Web Conference (WWW10), pp. 1-13, Hongkong, May 1, 2001.
- **Managing RDF Metadata for Community Webs**, Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, Karsten Tolle, Bernd Amann, Irini Fundulaki, Michel Scholl, Anne-Marie Vercoustre, 2nd International Workshop on the World Wide Web and Conceptual Modeling (2000), pp. 140-151.