Fachhochschule Karlsruhe

Hochschule für Technik

Ein Semantisches Web für die Universitätsbibliothek Heidelberg

eingereicht als

Masterthesis

Student: Michael Milvich
Matrikelnummer: 18176
3. Juli 2005

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich diese Arbeit selbständig und ohne unzulässige Hilfsmittel angefertigt habe. Die verwendeten Quellen sind im Literaturverzeichnis angegeben. Die Arbeit hat in dieser oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen.

Karlsruhe, den 3. Juli 2005 Michael Milvich

Danksagung

Ohne die vielfältige Unterstützung durch meinen Betreuer an der Fachhochschule Karlsruhe Prof. Dr. Ulrich Bröckl wäre mir die Anfertigung dieser Masterthesis nicht möglich gewesen.

Des Weiteren möchte ich mich bei folgenden Mitarbeitern des IT-Teams der Universitätsbibliothek Heidelberg bedanken: Leonard Maylein lies mir viel Freiraum für das Verfolgen eigener Ideen, Anette Langenstein für die gute Unterstützung bei bibliothekarischen Fragen und Martin Braun für fruchtbare Diskussionen und Vorschläge.

Stefan Bozic unterzog die Arbeit einer kritischen Prüfung und wies mich auf manche Verbesserungsmöglichkeit hin. Mein Vater Geza Milvich und meine Freundin Christine Hildebrandt, wiesen mich auf manchen syntaktischen Fehler hin und machten mir an Tagen, an denen es nicht so gut lief, wieder Mut.

Besonderer Dank gilt Dipl. Inf. Kevin Hausmann von der Universität Oldenburg, welcher mir mit guten Ratschlägen in oftmals längeren Telefonaten zur Seite stand.

Von großem Vorteil war es, das Programms LaTeX für die Masterthesis einzusetzen. Je mehr der Umfang der Masterthesis wuchs, desto strukturierter und logischer erschien mir das Konzept von LaTeX.

Inhaltsverzeichnis

Aufgabenstellung

Thema: Ein Semantisches Web für die Universitätsbibliothek Heidelberg.

Die Universitätsbibliothek Heidelberg bietet eine große Anzahl an verschiedenen Diensten an. Der bekannteste und verbreitetste Dienst ist der Verleihservice von Büchern. Viele Nutzer der Universitätsbibliothek wissen nicht, dass eine Vielzahl an elektronischen Diensten vorhanden sind. Diese Unkenntnis hat eine Benutzerumfrage im Winter 2004 bestätigt.

Zu den elektronischen Diensten gehören Zeitschriften (eJournals), digitalisierte Literatur, Dozentenbibliographien, ein Dokumentenserver für Seminar-, Diplom- und Doktorarbeiten, die im Umfeld der Universität entstanden sind sowie eine umfangreiche Sammlung bibliographischer Datenbanken.

Um die elektronischen Dienste innerhalb der Universitätsbibliothek populärer zu machen, ist ein Projekt namens *myUB* geplant, welches zum Ziel hat, den Benutzern eine individuelle und personalisierte Sicht auf die elektronischen Dienste zu ermöglichen. Derzeit gibt es diese Option nicht, so dass der jeweilige Dienst ausschließlich durch Navigieren innerhalb der Universitätsbibliothek-Webseite erreicht werden kann. Ein individuelles myUB-Konto wird dem Benutzer den Zugriff auf die gewünschten Medien erleichtern. Innerhalb des myUB-Konto ist es möglich, die benötigten elektronischen Dienste zu vernetzen, wodurch ein schnellerer Zugriff auf das entsprechende Medium ermöglicht wird. Das myUB-Projekt befindet sich noch in der Entwicklungsphase und wird durch die IT-Abteilung der Universtitäsbibliothek Heidelberg programmiert.

Damit die Individualisierung von myUB funktioniert, müssen sich die Benutzer über die vielseitigen Möglichkeiten bewusst sein. Durch die Vielzahl an verschiedenen Diensten ist das manuelle Durchsuchen der einzelnen Rubriken sehr zeitaufwändig. Aus diesem Grund ist ein wichtiges Ziel von myUB, eine Möglichkeit zur umfangreichen Suche über diese elektronischen Dienste anzubieten.

Für die Realisierung bietet sich das Konzept des Semantischen Webs an, welches eine intelligentere Suche ermöglicht, als mit herkömmlichen Volltextsuchmaschinen zurzeit möglich ist. Ein Semantisches Web über die elektronischen Dienste muss in der Lage sein, dem Benutzer aus den verschiedenen elektronischen Diensten die passendsten Treffer vorzuschlagen.

Die Masterthesis hat das Ziel, das Konzept, welches hinter dem Semantischen Web steht, zu analysieren und dieses Konzept auf eine myUB-Suche anzuwenden. Für die Prüfung der Effizienz wird ein erster Prototyp im Verlauf der Masterthesis entwickelt werden. Zur Realisierung des Prototypen müssen zunächst verschiedene mögliche Semantische Web Ansätze zusammengefasst und analysiert werden. Danach erfolgt der Entwurf einer so genannten Ontologie. Zuletzt wird der Prototyp mit Hilfe konkreter Semantik-Web Werkzeuge aufgebaut.

Das Ergebnis der Masterthesis soll aufzeigen, ob es durch das Semantische Web möglich ist, bessere Suchresultate zu erzielen als durch gegenwärtige Datenbankrecherchen oder ob sich der Mehraufwand zur Entwicklung eines semantischen Webs nicht lohnt.

Einleitung

Seit der Entstehung des World Wide Webs - kurz WWW - hat sich vieles in unserer stark technologisierten Gesellschaft verändert. Wurde anfangs noch das dezentralisierte Konzept des WWWs kritisiert, welches keine Möglichkeiten zur vollständigen Inventarisierung aller Inhalte anbietet, übernehmen heutige Suchmaschinen sehr erfolgreich diese Aufgabe [DOS03].

Nicht zuletzt dieser Idee verdankt das WWW seinen weltweiten Erfolg. Jeder Teilnehmer kann heutzutage einfach seine eigenen Informationen im Web veröffentlichen, ohne dass er sich bei einer zentralen Stelle registrieren muss, die die Aufgabe hat, die gewünschten Daten zu verwalten, um sie später anderen Benutzern zugänglich zu machen. Das Gegenteil ist der Fall. Der Benutzer muss lediglich warten, bis Suchmaschinen seine Seiten indizieren und schon können sie – zumindest theoretisch – im Web gefunden werden. Durch den großen Erfolg des WWW werden tagtäglich neue Informationen im Web verbreitet. Die gewünschten spezifischen Inhalte sind dank Datenbanken, Suchmaschinen und weltweiter Kommunikation einfacher zu erhalten als jemals zuvor.

Auch im Buch- und Bibliothekswesen wurde bereits sehr frühzeitig damit begonnen, große elektronische Datenbanken über die vorhandenen Medien anzulegen. Musste früher in Karteikästen nach gewünschten Medien gesucht werden, übernehmen heute Computer diese Aufgabe und erledigen sie in einem Bruchteil der Zeit, die ein Mensch dafür benötigen würde. Medienrecherchen in Datenbanken sind nicht zuletzt durch die Vernetzung im WWW einfacher denn je geworden.

Obwohl sich durch die Einführung der Computer im Bibliothekswesen vieles vereinfacht hat und die Vorteile der Suche nach Medien durch Computer gegenüber Karteikästen überwiegen, entstehen neue Probleme und Wünsche. Einerseits sind Computer nicht intelligent und können deshalb nicht erkennen, ob ein bestimmtes Medium für den Benutzer interessant sein könnte, andererseits passiert es häufig, dass ein Anwender durch eine Suche Informationen erhält, die er nicht benötigt. Computer sind nur in der Lage, bestimmte Medien zu finden, wenn diese ausdrücklich innerhalb der Datenbank mit entsprechenden Stichworten beschrieben wurden. Dieses Problem ist nicht auf das Bibliothekswesen beschränkt, sondern betrifft alle Bereiche in denen mit Hilfe von Computern, Dokumenten- oder Medienrecherchen durchgeführt werden.

Ein großes Problem beim Arbeiten mit Computern ist deshalb der Informationsüberfluss. Viele Daten, ganz gleich woher sie stammen, fluten tagtäglich auf uns ein. Durch diese unüberschaubare Menge an Daten wird der Benutzer von den eigentlich wichtigen Inhalten abgelenkt. Informationsüberfluss bezeichnet man auch als *Information Overkill* und kann

als der neue Fluch bei der Suche mit Computern angesehen werden [DOS03]. Häufig wird mit der Recherche und darauf folgender Selektion der Treffer, die eine Suchmaschine zurückliefert, sehr viel Zeit verbraucht. Die Datenforscher von Netcraft ermittelten im März 2005 exakt 60.442.655 Webseiten, erst im Mai 2004, also vor neun Monaten, kletterte diese Zahl über die 50-Millionen-Marke. Damit beschleunigt sich das Wachstum des Internets weiter [Hit05].

Aus den oben genannten Gründen wäre es wünschenswert, nur eine bestimmte Menge an wichtigen und nützlichen Informationen zu erhalten, die besser auf die entsprechende Person und deren Bedürfnisse zugeschnitten sind. Hier gilt das Motto: „Weniger ist mehr“. Deshalb sollte ein Suchprogramm in der Lage sein, eine Selektion für den Benutzer durchzuführen und nur die passenden Daten an den Anwender zu übermitteln.

Ein Konzept, das seit einiger Zeit von sich reden macht, ist die Idee des *Semantischen Webs*. Dieses wird als der Hoffnungsträger für das Web der nächsten Generation angesehen. Die Idee beruht darauf, Beschreibungsstandards und Technologien zu entwickeln, mit denen einerseits die Suche nach Informationen und Dokumenten verbessert werden kann, und andererseits die automatische Verarbeitung von Daten und Wissen aus unterschiedlichen Quellen unterstützt wird. Zur Realisierung dieser Idee wird auf bereits existierende Technologien aufgebaut [Fer03]. Bei einem Semantischen Netz handelt es sich um ein Gebilde, das vergleichbar ist mit so genannten assoziativen Netzwerken im menschlichen Gehirn.

Bereits um 1900 wurden von Psychologen, z.B. Gustav Aschaffenburg, Untersuchungen durchgeführt, wie Begriffe in unserem Gehirn miteinander verknüpft sind. Dabei wurde herausgefunden, dass bestimmte Worte bei den meisten Menschen die gleichen Assoziationen hervorrufen, beispielsweise weiß – schwarz oder Mutter – Vater. Durch Assoziationen ist es möglich, Begriffe auf der Semantischen Ebene miteinander zu verbinden, d.h. dass Wörter aus einem bestimmten Grund eine Beziehung zueinander haben. Wörter und Bedeutungen sind im mentalen Lexikon nicht alphabetisch oder völlig unorganisiert, sondern netzwerkartig gespeichert. Die Bedeutung eines Wortes wird in einem solchen Netzwerk durch Knoten repräsentiert sowie durch Nachbarschaftsbeziehungen zu anderen Inhalten bzw. Begriffen. Solche Netzwerke lassen sich im Ansatz aus den gerade erwähnten Assoziationen gewinnen [Man96].

Dieses Konzept der Assoziationen und Nachbarschaftsbeziehungen versucht das Semantische Web aufzugreifen. Dadurch kann es möglich werden, dass semantisch verwandte aber syntaktisch vollständig verschiedene Begriffe im Semantischen Web gefunden werden. Die folgende Abbildung 1 zeigt ein solches Assoziationsgebilde, wie es im menschlichen Gehirn vorhanden ist und die grundlegende Idee für das Semantische Web bildet.

Derzeit werden erste Anwendungen realisiert, die auf das Semantische Netz setzen. Diese verknüpfen Informationen miteinander, wo es sinnvoll erscheint. Dabei entsteht der benötigte Netzwerk-Effekt. Die Vision ist, dass eine kritische Masse erreicht wird, in der alles zu einem großen Ganzen verknüpft wird. Der Anreiz, mehr hinzuzufügen, steigt dann exponentiell, ebenso wie der Wert des bereits Vorhandenen. Weil man nicht sofort den Nutzen der Verknüpfung riesiger Datenmassen im semantischen Web erkennt, muss dieses durch einen fachlichen Personenkreis umgesetzt werden, welcher der Überzeugung ist, dass sich der enorme Aufwand letztlich lohnt [BL04b].

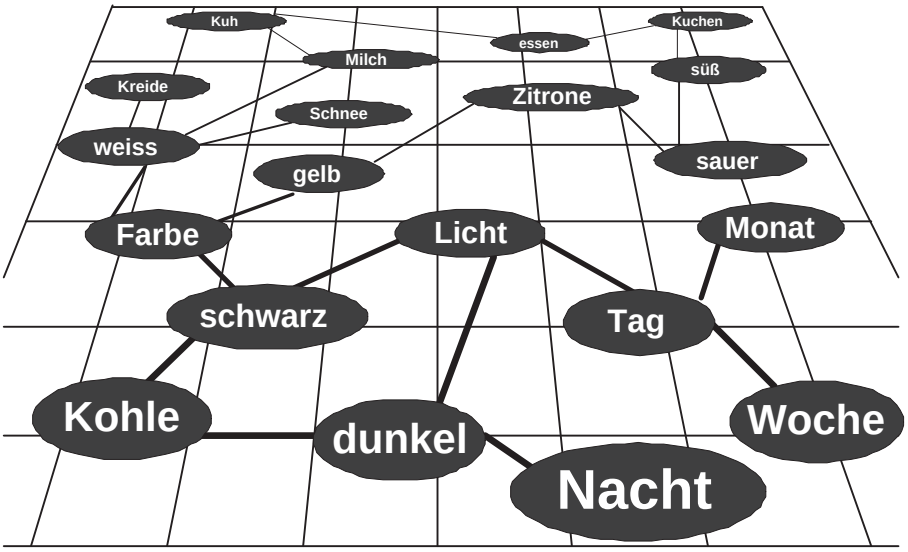


Abbildung 1: Semantisches Netzwerk im menschlichen Gehirn [Man96]

Die Masterthesis hat die folgende Struktur:

Kapitel 1 gibt eine Übersicht über den aktuellen Stand der Forschung im Semantischen Web und beschreibt die darauf basierenden Ideen. Anhand eines Beispiels werden die Vorteile dieses Konzepts gegenüber dem World Wide Web angegeben.

Kapitel 2 erklärt die fundamentalen Konzepte, mit denen das Semantische Web realisierbar ist. Dabei werden Begriffe wie Logik, Taxonomie und Ontologie genauer erklärt.

Kapitel 3 beschreibt die technischen Grundlagen, die zum Aufbau eines Semantischen Netzes notwendig sind. Die hierfür existierenden Sprachen RDF und das darauf aufbauende OWL werden vorgestellt und erläutert. Weiterhin wird dem Leser ein Einblick in die Einsatzmöglichkeiten dieser Sprachen im Bezug auf das Semantische Web gegeben.

Kapitel 4 ist ein Lastenheft, welches die detaillierten Anforderungen an ein Semantisches Web für die Universitätsbibliothek Heidelberg spezifiziert. Es wird analysiert, welche Bedürfnisse die Benutzer der Universitätsbibliothek Heidelberg bei der Suche haben und wie diese mit Hilfe von Semantischen Web Technologien innerhalb der Masterthesis gelöst werden können.

Kapitel 5 erläutert, wie die in Kapitel 3 angegebenen Anforderungen in der Masterthesis umgesetzt werden. Dabei wird begründet, warum eine bestimmte Technologie abgelehnt oder angewendet wurde. Danach erfolgt die Erklärung der modellierten Ontologie und zuletzt die technische Umsetzung.

Kapitel 6 zeigt einige Anwendungsbeispiele und gibt anhand von Abbildungen des erstellten Projekts einen ersten Eindruck.

Zuletzt erfolgt eine Zusammenfassung der Arbeit, wobei aufgezeigt wird, welche Ziele der Aufgabenstellung durch diese Masterthesis realisiert werden konnten. Des Weiteren wird ein Ausblick auf die zukünftige Entwicklung gegeben.

Kapitel 1

Einführung in das Semantische Web

Dieses Kapitel hat das Ziel, dem Leser einen allgemeinen Überblick über das Semantische Web zu verschaffen. Es wird eine ausführliche Beschreibung über die grundlegende Idee, die hinter dem Semantischen Web steckt, gegeben. Dabei wird anhand einiger Beispiele verdeutlicht, was durch diese Konzept möglich wird. Weiterhin wird gezeigt, wo die Grenzen dieses Konzepts und die Vorteile gegenüber dem World Wide Web liegen.

1.1 Das Semantische Web

Was ist das Semantische Web? Niemand kann diese Frage zurzeit vollständig beantworten, weil dieses Web noch gar nicht existiert. Es ist lediglich ein Konzept, welches im Entstehen ist und vielleicht irgendwann das Internet revolutionieren könnte.

Das Semantische Web ist eine Idee zur Verbesserung des World Wide Webs, mit dem Ziel, Daten zu erzeugen die von den verschiedensten Programmen verstanden und dadurch besser verarbeitet werden können. Das Semantische Web wurde vom World Wide Web Erfinder Tim-Berners-Lee und den Mitarbeitern am World Wide Web Consortium, kurz *W3C*, entworfen. Das W3C hat es sich zur Aufgabe gemacht, Spezifikationen und Empfehlungen für die Weiterentwicklung des Internets zu entwerfen.

Allgemein ausgedrückt, ist es das Ziel des Semantischen Webs, auf semantischer Ebene von beliebigen Programmen verstanden und abgearbeitet werden zu können. Der Begriff setzt sich aus den folgenden Wörtern zusammen:

Semantik: Bedeutung sprachlicher Ausdrücke

Web: Netz von Links (URL, URI)

Das Semantische Web kann als ein Netz von Inhalten verstanden werden, welches darauf abzielt, dass Maschinen exaktere Ergebnisse zurückliefern können, als es mit heutigen Programmen möglich ist [Wah01].

Seit den ersten Computern vor ca. 50 Jahren sind Rechner in der Lage, Daten zu verarbeiten. Diese Daten werden i.d.R. binär, Schritt für Schritt und ohne Zusammenhang an den Hauptprozessor übergeben. Dieser Prozess geschieht ausschließlich auf syntaktischer Ebene, d.h. der Prozessor prüft, ob die an ihn übergebenen Zeichen alle logisch und korrekt aneinander gereiht sind. Welche Bedeutung das Ganze hat, kann ein Computer nicht wissen. Dabei wäre es doch wünschenswert, dass Rechenmaschinen bestimmte Zusammenhänge besser erkennen würden, um beispielsweise Missverständnisse mit dem Anwender zu vermeiden oder besser mit anderen Programmen kommunizieren zu können. Dadurch würde viel Zeit und Geld eingespart werden können.

Ein bekanntes Beispiel zeigt das Problem, das auftritt, wenn eine Suchmaschine nach einem Begriff suchen soll, der mehrere Bedeutungen hat, z.B. *Geld*-Bank und *Park*-Bank. Hier kann leicht ein Missverständnis zwischen dem Anwender und dem Computer entstehen, da für einen Computer das Wort Bank lediglich eine Verkettung von 4 Zeichen ist. Der Computer kennt die dahinter stehende Bedeutung nicht. Eine Suchmaschine würde daher alle Artikel zurückliefern, ohne dabei eine tiefer gehende Selektion zur Unterscheidung zwischen (Geld) Bank und (Park) Bank durchzuführen.

Es existiert eine Vielzahl solcher semantischer Probleme, die während einer automatischen Suche durch den Computer auftreten können. Einige Beispiele werden in der folgenden Aufzählung zusammengefasst:

- **Synonyme** sind Wörter, die Bedeutungsähnlichkeit aufweisen. Eine Suchmaschine, die nach dem Volltextprinzip arbeitet, ist nicht in der Lage zu erkennen, dass beispielsweise ein *Stockwerk* und eine *Etage*, Synonyme für die gleiche Sache sind.
- **Homonyme** sind Bezeichnungen, die im Sprachgebrauch auch als *Teekessel* bezeichnet werden. Dabei handelt es sich um Wörter, die gleich geschrieben werden, jedoch oftmals völlig unterschiedliche Bedeutungen aufweisen.
- **Rechtschreibfehler** werden in der Zwischenzeit von einigen Suchmaschinen gut erkannt. Es können jedoch Probleme auftreten, wenn das Korrekturprogramm mehrere ähnlich geschriebene Verbesserungsvorschläge zur Auswahl hat. Hier ist es nicht möglich, anhand des Kontextes die beste Lösung zu finden. Es wird lediglich ein “best Match“-Algorithmus verwendet [Wle03].
- **Sinneszusammenhänge** können oftmals nicht erkannt werden. Die Suchmaschine ist nicht in der Lage herauszufinden, welches das Subjekt oder das Objekt einer komplexeren Suchanfrage ist [Wle03].
- **Wortformen** existieren gerade in der deutschen Sprache in vielfacher Weise. Eine Suchmaschine kann dabei nicht erkennen, um welchen Grundtyp des Wortes es sich handelt damit die Anfrage entsprechend kategorisiert werden kann. Beispielsweise müsste eine Suchmaschine in der Lage sein, die gleiche Anzahl an Treffern für das Adjektiv *finanziell* und das Subjekt *Finanzen* zu bringen oder zumindest einen Zusammenhang zwischen den Wörtern zu erkennen [Wle03].

Würde ein Semantisches Web existieren, gäbe es eine ganz andere Art von computergesteuerter Suche. Oberflächlich würden immer noch Wörter oder Schlagworte und keine

vollständigen Sätze eingegeben werden. Jedoch wäre eine semantische Suchmaschine in der Lage, eine andere Art der Trefferauswertung zu verwenden, als heutige Volltextsuchmaschinen. Die semantische Suchmaschine könnte ein eingegebenes Schlagwort direkt einem bestimmten Umfeld zuordnen und alle Informationen zurückliefern, die dem Umfeld dieses Schlagwortes zugeordnet wurden.

Falls beispielsweise ein Anwender den oben beschriebenen Begriff *Bank* in eine semantische Suchmaschine eingibt, wird diese nicht nur auf syntaktischer, sondern auch auf semantischer Ebene versuchen, den Begriff zu analysieren. Dabei könnte die semantische Suchmaschine erkennen, dass es zwei völlig unterschiedliche Bereiche gibt, die den Benutzer möglicherweise interessieren. Im nächsten Schritt müsste die Suchmaschine nachfragen, ob der Anwender Informationen über eine Bank im Bereich *Wirtschaft* wünscht oder im Bereich *Gartenbau*. Nachdem der Benutzer seine Wahl getroffen hat, kann die Suchmaschine alle Ergebnisse zurückliefern, die im entsprechenden Umfeld des ausgewählten Bereichs liegen. Die folgende Abbildung 1.1 verdeutlicht dieses Suchprinzip. Zur Modellierung dieser Abbildung wird das vom W3C empfohlene Modellierungsschema für RDF verwendet [MM04]. Darin werden die einzelnen Objekte, welche auch als Ressourcen bezeichnet werden, immer als Ellipse modelliert.

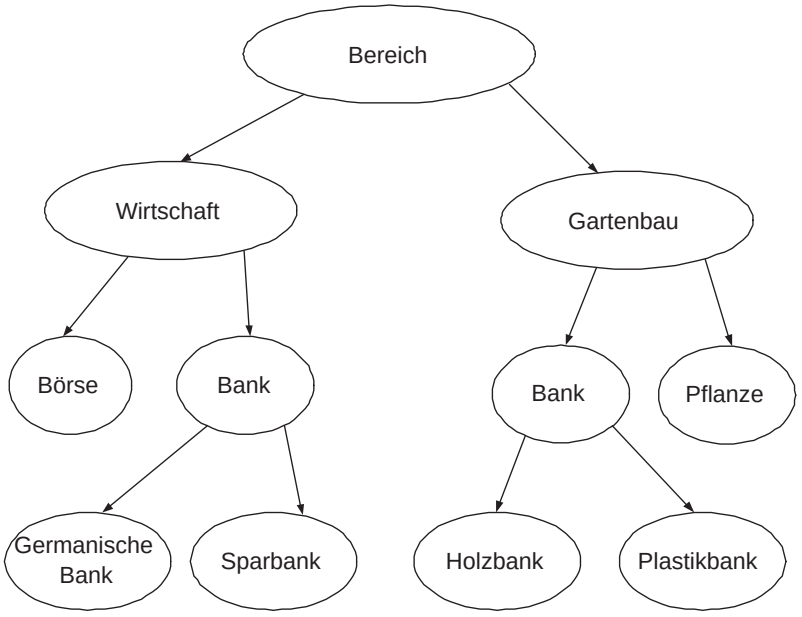


Abbildung 1.1: Unterscheidung eines Wortes mit zwei Bedeutungen

Innerhalb der Abbildung sind die verschiedenen Kategorien des obigen Beispiels aufgezeigt. Des Weiteren sind bereits einzelne Objekte diesen Kategorien zugeordnet worden. Davon abhängig, was der Benutzer sucht, liefert die Suche die entsprechenden Inhalte aus der passenden Kategorie zurück. Die Unterteilung ist, wie bereits erwähnt, nur möglich, weil bereits eine Klassifizierung der Daten im Voraus getroffen wurde. Durch die Klassifizierung wird es möglich, die Information *Germanische Bank* und *Sparbank* dem Bereich *Wirtschaft* zuzuordnen. Genauso wird die *Holzbank* und die *Plastikbank* im Bereich *Gartenbau* eingeordnet.

Die Suchmaschine *Seekport* [Tea02a] verwirklicht das beschriebene Beispiel. Dabei verarbeitet sie jedoch nur die Eingabe des Benutzers auf semantischer Ebene. Wird beispielsweise ein Homonym eingegeben, dann macht Seekport einen Vorschlag zur Verfeinerung des Suchbegriffs. Die eigentliche Suche ist jedoch eine normale Volltextsuche.

Selbstverständlich können Maschinen diese Inhalte nicht im eigentlichen Sinne verstehen, jedoch können sie diese Inhalte exakter erfassen, als bei einer Suche nach den passenden Wörtern. Wie bereits erwähnt wird dies dadurch ermöglicht, dass eine formale Logik aufgebaut wird, welche die Suchmaschine mit den entsprechenden Daten versorgt [EE04].

Wie in dem Katalog einer Bibliothek, welcher mit Schlagwortverkettungen (s. 4.1.4) arbeitet, könnten die Suchmaschinen nicht lediglich wahllos Dokumente mit vermeintlich "richtigem" Inhalt ans Tageslicht befördern, sondern genau diejenigen Informationen bereitstellen, die auch wirklich semantisch erfragt worden sind. Auf diese Weise könnten Synonyme und ganze Wortstämme miteinander verbunden, abgeglichen und letztlich ausgewertet werden [Tea02b].

In der gezeigten Abbildung 1.1 ist es wichtig zu beachten, dass die bereits gezeigte Hierarchie der Kategorien von Menschen erzeugt werden muss. Diese Hierachie bildet später einen Teil der Wissensbasis. Der andere Teil wird aus den eigentlichen Informationen gebildet. Die folgende Abbildung 1.2 zeigt eine solche Wissensbasis. Jedoch werden in dieser Abbildung nur die einzelnen Kategorien aufgelistet. Das Thema greift auf das einführende Beispiel (s. Abbildung 1.1) zurück. Für das Modellieren der Wissensbasis wird eine Vorlage von [Sow00] verwendet.

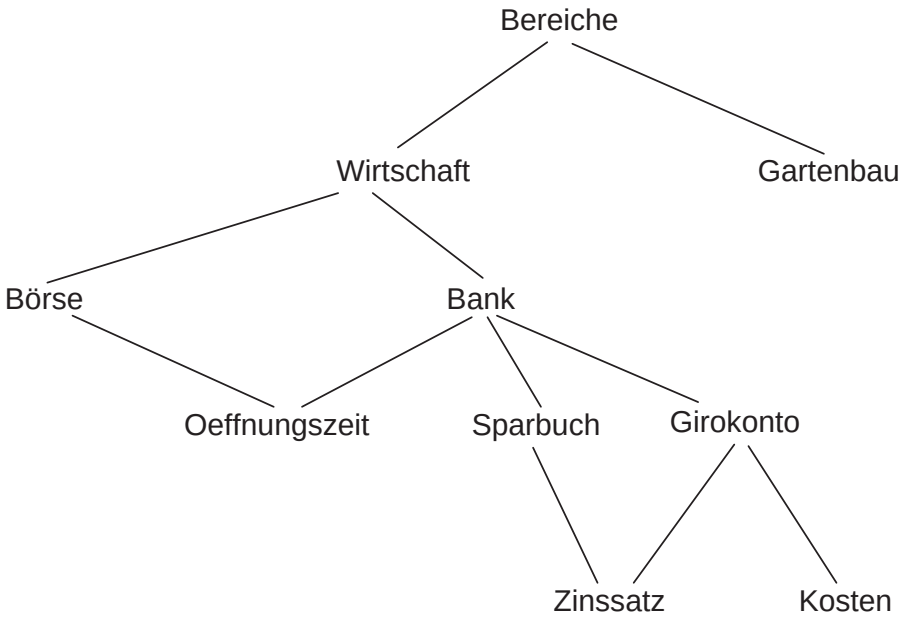


Abbildung 1.2: Wissensbasis/Ontologie

Der in Abbildung 1.2 gezeigte Teil der Wissensbasis wird als *Ontologie* bezeichnet. Ontologien und deren Design sind ein komplexer Prozess, der versucht, einen bestimmten Bereich der Welt abzubilden. Dabei ist die Modellierung des Bereiches auch Ansichtssache. Viele Menschen besitzen eine unterschiedliche Betrachtungsweise für einen bestimmten

Bereich der Welt. Da es oftmals nicht möglich ist, einen ganzen Bereich, beispielsweise die ganze Welt der Mathematik, vollständig in einer Modellierungssprache abzubilden, ist es wichtig, sich bereits am Anfang der Modellierung genau zu überlegen, welche Teile für das Modell notwendig sind. Aus diesen Gründen ist die Erstellung einer Ontologie ein langwieriger und komplizierter Prozess. Der Abschnitt 2.4.3 beschreibt einige bekannte Ansätze zum Produzieren von Ontologien.

Um eine vollständige Wissensbasis, bestehend aus Ontologie und den dazugehörigen Daten zu erhalten, müssen alle benötigten Informationen, den einzelnen Kategorien in der Ontologie zugeordnet werden. In der obigen Abbildung 1.1 erfolgt die Zuordnung des Objektes *Germanische Bank* innerhalb des Bereichs *Wirtschaft*.

Damit die Zuordnung funktioniert, ist es nötig, dass die Daten mit zusätzlichen Informationen – so genannten *Metadaten* – ausgezeichnet werden. Dadurch kann der oben beschriebene Prozess angestoßen werden, in welchem ein Computer in der Lage ist, die mit Metadaten versehenen Webseiten, der entsprechenden Kategorie in der Wissensbasis zuzuordnen.

Das folgende Beispiel verdeutlicht nochmals das Konzept. Ein Webdesigner könnte die Webseite für die *Germanische Bank* mit bestimmten Metainformationen versehen. Durch diese Metadaten wird die *Germanische Bank* eindeutig dem Bereich *Wirtschaft* und der Kategorie *Bank* zuordnet. Des Weiteren könnte der Webdesigner die *Öffnungszeiten* der Bank, den aktuellen *Zinssatz* für ein *Sparbuch* oder die *Kosten* für ein *Girokonto* als Metainformation integrieren.

Eine semantische Suchmaschine wäre durch die korrekte Auszeichnung der Metainformationen in der Lage, diese zusätzlichen Informationen innerhalb der eigentlichen Daten zu verarbeiten und damit die Webseite an der korrekten Stelle innerhalb der Wissensbasis zu inventarisieren. Im Folgenden wird, unabhängig von einer Beschreibungssprache wie XML oder RDF, gezeigt, wie die Metainformationen beschrieben werden könnten.

```
wirtschaft
    bank germanische Bank
         oeffnungszeit 10-17
         sparbuch
              zinssatz 4,5
         girokonto
              zinssatz 0,5
              kosten 5 Euro montl.
```

Wie gelangen die Metainformationen in die Suchmaschine? Dies geschieht durch regelmäßige Prüfung der Metainformationen durch so genannte *Web-Spiders*. Die Suchmaschine wird die oben beschriebenen Metainformationen nur verarbeiten können, wenn sie diese in ihrer Wissensbasis einem vorhandenen Element korrekt zuordnen kann. Deshalb ist es wichtig, die Metadaten genauso zu definieren, wie in der Wissensbasis vorgeschrieben.

Natürlich könnte die Wissensbasis auch mehrere Einträge für ein Wort enthalten, beispielsweise Synonyme für ein Element oder verschiedenen Definitionen in mehreren

Sprachen. Das zweite Kriterium ist zwingend notwendig für eine internationale Suchmaschine.

Die folgende Abbildung 1.3 verdeutlicht den Prozess der Zuordnung der Metainformationen zu den Elementen der Ontologie. Dazu wurde wieder das Modellierungsschema für RDF vom W3C verwendet.

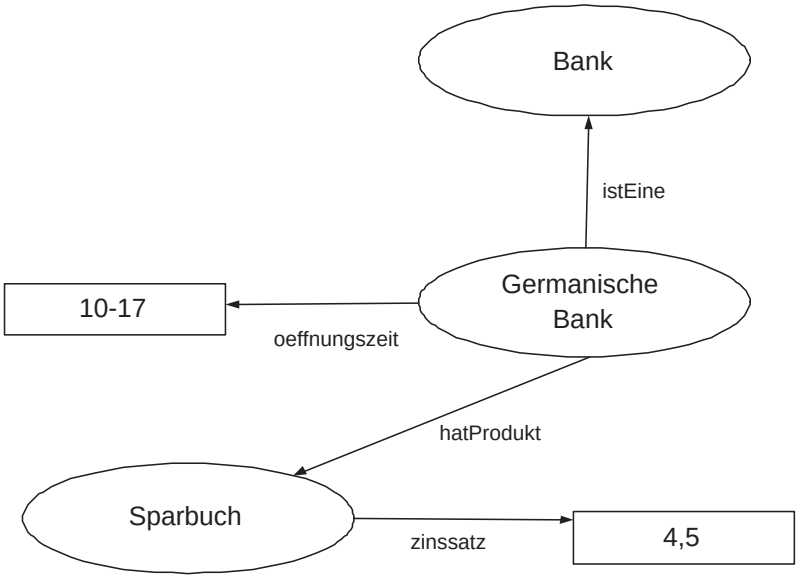


Abbildung 1.3: Einordnung einzelner Objekte in die Wissensbasis

Die Abbildung zeigt die Metadaten innerhalb eines Netzwerks von Knoten [Hje01]. Jeder Knoten kann einen direkten Wert – in rechteckigen Feldern dargestellt – oder eine Ressource darstellen, die auf weitere Ressourcen oder auf Werte verzweigt.

Dabei gilt zu beachten, dass sowohl Kategorien wie *Bank* oder *Sparbuch*, als auch das zu inventarisierende Objekt *Germanische Bank* als Ressource dargestellt werden. Diese Art der Modellierung von Inhalten ist ein Konzept, welches man als *RDF* bezeichnet und in den nächsten Kapiteln ausführlich besprochen wird.

1.2 Anwendungsbereiche des Semantischen Web

Das im vorherigen Abschnitt präsentierte Beispiel zeigt einen Teilaspekt, der durch ein Semantisches Web gelöst werden kann. Es existieren jedoch viele Möglichkeiten, welche die folgende Aufzählung wiedergibt:

- **Semantisch unterstützte Suchmaschinen** werden in absehbarer Zeit, auch auf semantische Definitionen und damit verbundene Begriffshierarchien Rücksicht nehmen. In den Begriffshierarchien der Ontologie erfolgt die Beschreibung der Verwandschaftsbeziehungen. Dadurch sind auch Seiten auffindbar, die nicht genau die gesuchten, sondern nur verwandte Begriffe enthalten [EE04].

- **Content- bzw. Dokument-Managementsysteme** haben die Aufgabe, Dokumente zu verwalten um daraus große Webseiten aufzubauen (CMS) und weiterhin, um in Unternehmen die Vielzahl von Dokumenten nutzbar zu halten (DMS). Hier tritt genau wie im Web das Problem der Suche nach dem richtigen Dokument auf, aber auch die Verarbeitung von Informationen, die in den Dokumenten stehen.

  CMS können beispielsweise aus verschiedenen Informationen personalisierte Webseiten generieren. Hierfür müssen Informationen von Nutzern wie zum Beispiel Profile mit den Metadaten über die Inhalte, in Verbindung gebracht werden [EE04]. Das CMS wäre dadurch in der Lage, mit Hilfe von Metadaten, die im Profil eingetragen sind, die passenden Information in einer Webseite für den individuellen Benutzer zu liefern.

- **Virtuelle Agenten** sind Programme, die meist auf Webseiten angebracht werden und die Aufgabe haben, mit dem Benutzer zu kommunizieren. Um die vom Benutzer eingegebenen Sätze zu verarbeiten, benötigt der virtuelle Agent eine Wissensbasis. Diese hat die Aufgabe, die semantische Bedeutung des eingegebenen Satzes zu erkennen, und die passende Antwort zu liefern. Dazu kommt eine Ontologie zum Einsatz, die wiederum auf den Bereich, in dem der virtuelle Agent tätig ist, angepasst und gepflegt wird.

  Beispielsweise könnte ein System die Frage „*wo gibt es die billigsten Banken*“ automatisch dem entsprechenden Bereich *Gartenbau* und nicht *Wirtschaft* zuordnen, wenn sich der Benutzer bereits vorher über Pflanzen und Steine bei dem virtuellen Agenten erkundigt hat. Im Bibliothekswesen werden diese bereits auf einigen Webseiten eingesetzt, beispielsweise bietet die Staats- und Universitätsbibliothek Hamburg [Chr05] mit *Stella* eine virtuelle Agentin an.

- **Bilddatenbanken** werden in Zeitschriftenredaktionen eingesetzt. Oftmals existiert eine Vielzahl an Bilden. Deshalb kann es sich als schwierig herausstellen, das passende Bild zu finden. Ein Semantisches Web über eine Bilddatenbank kann dabei helfen, Bilder nach verschiedenen Kategorien zu suchen.

- **Web-Services** sind ein aufstrebender Bereich, bei dem über ein Web Programm Funktionen als Dienste angeboten werden. Durch die Kombination von verschiedenen Web-Services miteinander ist es möglich, größere Dienste aufzubauen, die mehrere Aufgaben durch einen Befehl erledigen. Zu den grundlegenden Problemen hierbei gehört das Finden von geeigneten Web-Services, die Beschreibung der Schnittstellen sowie weitere Inkonsistenzen, die entstehen können. Um diese Kombination zu unterstützten, können ebenfalls semantische-Web-Technologien zur Automatisierung eingesetzt werden [EE04].

  Die Abbildung 1.4 verdeutlicht das Konzept von Semantischen Web-Services. Semantische Web-Services, werden erst dann interessant, wenn es nötig ist, dass mehrere Web-Services miteinander interagieren müssen.

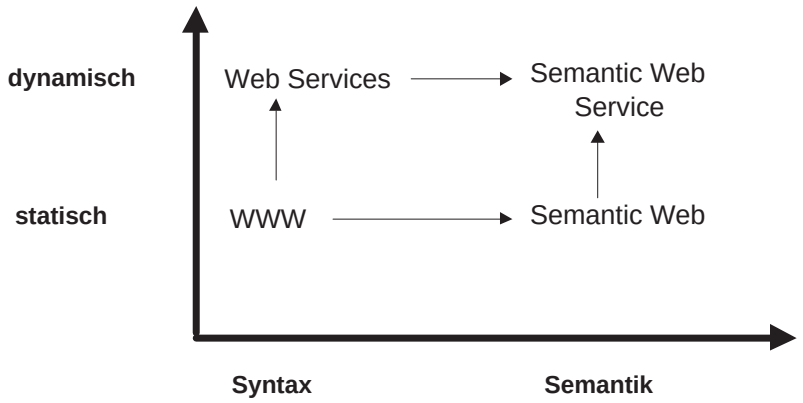


Abbildung 1.4: Semantische Web Services [DOS03]

1.3 Aktuelle Suchmaschinen

Die meisten Suchmaschinen haben einen speziellen Algorithmus, um zur Gewichtung ihrer Treffer zu gelangen. Die Suchmaschine Google verwendet für ihre Auswertung der Trefferreihenfolge den bekannten Algorithmus, wie viele Links auf eine Seite zeigen. Die Seite auf die am häufigsten verwiesen wird und die den entsprechenden Begriff enthält, wird ganz oben angezeigt. Amazon zeigt Artikel an, die dem Benutzer möglicherweise noch gefallen könnten, indem die Kaufaktivitäten früherer Konsumenten des gleichen Artikel untersucht werden. Obwohl diese beiden Methoden gute Ergebnisse liefern, wird keine wirkliche Intelligenz, noch nicht einmal Semantik eingesetzt. Stattdessen kommen einfache Statistikfunktionen zum Einsatz, die lediglich in einer Datenbank bestimmte Daten zusammenzählen und die Treffer mit den meisten Werten ganz oben anzeigen. Bei beiden Methoden wird darauf gehofft, dass möglichst viele Anwender damit zufriedengestellt werden können. Beide Methoden werden mit großem Erfolg eingesetzt. Dies kommt vor allem daher, dass im Augenblick keine effektiveren Möglichkeiten für eine spezifischere Suche existieren, die sich in mit realistischem Aufwand einsetzen lassen würden.

1.4 Künstliche Intelligenz

Computer sind nach wie vor lediglich Systeme, die Einsen und Nullen verstehen können. Wie soll beispielsweise ein Computer in der Lage sein, den in der Einleitung besprochenen Informationsüberfluss zu überwinden? Ein Computer hat keinerlei Möglichkeiten von sich aus, aus Fehlern, die der Anwender macht, zu lernen. Ein Computer kann immer nur eine bestimmte vorgegebenen Anzahl an Schritten, beispielsweise wie bei einem Schachprogramm vorausdenken. Querverweise, Assoziation und spontane instinktive Entscheidungen, wie sie beispielsweise durch Neuronen im Menschlichen Gehirn vorkommen, sind bei Computern nur schwierig möglich.

Das Gebiet der *Künstlichen Intelligenz* (KI) sowie das Spezialgebiet *Neuronale Netze* befassen sich mit diesen Themen. Die KI versucht Maschinen zu entwickeln, die sich so verhalten, als ob sie Intelligenz besitzen [KR96]. Dadurch könnten diese intelligenten

Maschinen in der Lage sein, zu erkennen, was für den individuellen Anwender von Nutzen ist, um den Informationsüberfluss zu stoppen. Neuronale Netze sind ein Konzept der Informatik, welches versucht, die Funktionsweise der menschlichen Neuronen im Gehirn nach zu modellieren [KR96].

Die Idee eines Semantischen Webs ist kein Konzept, welches aus dem Bereich der KI stammt. Während die KI versuchen würde, die Sprache des Menschen zu verstehen, muss beim Semantischen Web die Sprache für die Maschinen entwickelt werden [BL04c].

Bei einem Semantischen Web ist es zwingend nötig, dass einzelne Elemente des Netzes miteinander verbunden werden. Diese Konzept hat gewisse Ähnlichkeiten mit den Neuronalen Netzen. Jedoch ist es nicht möglich, dass das Netz von sich aus lernt. Es muss nach wie vor manuell von Menschen gepflegt werden [BL04c]. Neuronale Netze hingegen werden vom Menschen solange trainiert, bis dieser mit den Ergebnissen der Klassifizierung zufrieden ist. Danach kann das Neuronale Netz – zumindest theoretisch – selbständig arbeiten.

Die KI könnte oberhalb des Semantischen Webs aufsetzten, indem sie neue Webseiten erfasst, analysiert und den entsprechenden Knoten im Semantischen Web zuordnet. Damit würde es die oben beschriebenen Arbeit eines Webdesigners übernehmen, welcher die entwickelten Webseiten mit Metadaten für die semantische Suchmaschinen ausstattet. Dieser Schritt würde entfallen, wenn es möglich wäre, Werkzeuge zum Analysieren von Text (s. Gate [Cun04]) einzusetzen, um die Webseiten anhand des Inhaltes katalogisieren zu können. Allerdings ist die Technik in diesem Bereich noch nicht weit genug fortgeschritten, als dass es zufriedenstellende Ergebnisse gibt.

Natürlich hätte auch die KI durch den Einsatz von intelligenten Agenten Vorteile bei der Verwendung von Semantischen Netzen. Diese Agenten können zurzeit nur in bestimmten proprietären Bereichen, so genannten Expertensystemen, eingesetzt werden. Das hat den Grund, dass die Agenten ein bestimmtes Vokabular benötigen, um Ergebnisse zu liefern. Leider hat jeder Hersteller von Expertensystemen ein eigenes Vokabular entwickelt, dass größtenteils inkompatibel zu anderen Systemen ist. Würde das Semantische Web oberhalb des World Wide Webs realisiert werden, hätte man eine Informationsvielfalt ohnegleichen für Expertensysteme [DOS03].

Vermutlich wird letztendlich der Erfolg des Semantischen Webs stark mit der Weiterentwicklung der KI zusammenhängen, da der Aufbau eines Semantischen Webs einen starken Mehraufwand für die Entwicklung und Pflege bestehender Webseiten bedeutet, den viele Unternehmen aus Zeit und Kostengründen nicht bereit sind einzugehen. Erst wenn diese Pflege ebenfalls von Programmen vorgenommen wird, wird sich das Semantische Web möglicherweise ähnlich stark durchsetzen, wie es das WWW vor einigen Jahren getan hat.

1.5 Heutiger Zustand

Die Vorteile des Semantischen Webs sind eindeutig. Jedoch stellt sich die Frage, wie weit der Stand der Technologie zur Realisierung dieser Idee ist. Seit dem Formulieren der

Idee im Jahr 1998 durch Tim-Berners-Lee hat sich einiges verändert. Es wurden viele Konzepte entwickelt, wie die Beschreibungssprachen RDF und OWL (s. Kapitel 3). Viele große Firmen entwickeln zurzeit Produkte in diesen Bereichen:

- Die *Hewlett Packard Labs Semantic Web research group* entwickelt eine sehr gute API, zum Erstellen eigener Semantik Web Entwicklungen unter Java [Tea04b].
- Adobe strukturiert seine Metadaten über verschiedene Softwareprodukte mit Hilfe von RDF um [DOS03].
- Die deutsche Firma *Ontoprise*, hat sich bereits fest auf den Bereich Ontologie Produktion spezifiziert und erstellt Programme zur Modellierung und Wissensfindung innerhalb von Ontologien [DOS03].

Es existiert noch eine Vielzahl an weiteren Techniken und Projekten. Bisher kann man beim Semantischen Web sicherlich nicht von einem “Hype“ wie beispielsweise bei WAP oder XML sprechen. Jedoch wird kontinuierlich daran gearbeitet, das Semantische Web zu weben [DOS03].

Skeptiker mögen argumentieren, dass es bisher immer Probleme gegeben hat, wenn es darum geht, menschliche Informationsverarbeitung mit automatischen Systemen zu simulieren. Vielversprechende Pläne aus dem Bereich der KI und der intelligenten Informationsverarbeitung, die in den letzten 40 Jahren gemacht wurden, mussten immer wieder verschoben werden. Wenn diese Systeme erfolgreich waren, lag das oft mehr an der Anpassung der Menschen an die Maschinen als umgekehrt, der Anpassung der Maschinen an die Menschen [Fer03]. Trotz allem hat die KI, wenn auch langsamer als erwartet deutliche Fortschritte gemacht, die beispielsweise bei Computerspielen wie Schach oder Backgammon, deutlich zu sehen sind.

Ontologien sind in vielen Firmen bereits integriert oder zumindest am Entstehen. Das Interesse an diesen Konzepten lässt sich sehr leicht am vorhandenen und steigenden Literaturangebot zum Thema Semantisches Web und Ontologien erkennen.

Daher kann zusammengefasst behauptet werden, dass die Technologie schon vorhanden ist. Viele Firmen investieren bereits Geld in semantische Projekte. Es werden Programme entwickelt, die diese Technologien nutzen [DOS03]. An vielen Universitäten existieren Projekte oder Diplomarbeiten, mit dem Ziel, die Konzepte des Semantischen Webs, in konkreten Projekten umzusetzen.

Das Semantische Web existiert noch nicht. Lediglich ein paar kleine Inseln sind zurzeit am Entstehen. Es ist jedoch ein allgemein bereits akzeptiertes Konzept, über das zumindest in der Fachwelt der Computerspezialisten, fragmentarisches Wissen vorhanden ist. Was noch fehlt, ist die so genannte “Killer Applikation“ [TE03]. Erst dann wird das Semantische Web, genauso wie das WWW, von speziellen Anwendungsdomänen, allmählich zu einem großen Ganzen verschmelzen.

Kapitel 2

Wissensrepräsentation

Das folgende Kapitel gibt eine detaillierte Beschreibung zu den grundlegenden Techniken des Semantischen Webs. Dabei wird beschrieben, wie bestimmte Konzepte, die aus verschiedenen wissenschaftlichen Bereichen stammen, für das Semantische Web zusammengeführt werden können. Der Schwerpunkt des Kapitels liegt in der Untersuchung und Beschreibung von Ontologien, dem eigentlichen Kern des Semantischen Webs.

2.1 Grundlagen

In der Informatik steht man in vielen Bereichen vor der Aufgabe, Erkanntes oder Erdachtes zu repräsentieren und Wissen zu kommunizieren, z. B. über Fakten, Sachverhalte oder Regeln in einem technischen Anwendungsbereich, in einem Geschäftsprozess oder in einem juristischen Verfahren oder über die Inhalte von Dokumenten oder Webseiten. Menschen können sich gespeichertes Wissen zunutze machen, indem sie auf ihr Grund- und Kontextwissen des jeweiligen Wissensbereichs zurückgreifen, Lehrbücher, Regelwerke, Lexika und Schlagwortregister verwenden und mit den gespeicherten Inhalten verbinden. Sollen dagegen Automaten Such-, Kommunikations- und Entscheidungsaufgaben in Bezug auf das gespeicherte Wissen übernehmen oder Daten austauschen, die selbst Information darüber enthalten, wie sie zu strukturieren und zu interpretieren sind, so benötigen sie dazu eine Repräsentation der zugrunde liegenden Begriffe und derer Zusammenhänge [Hes04].

Eine Möglichkeit zur Lösung dieses Problems zeigt das Konzept der Wissensrepräsentation – auch als *Knowledge Representation* bezeichnet. Dieses stammt aus dem Bereich der Künstlichen Intelligenz (siehe 1.4) und hat die Aufgabe, ein Modell eines bestimmten Bereichs der Welt in maschinenlesbarer Form abzubilden. Zur Realisierung beschreibt eine Wissensrepräsentation einen Wissensbereich – auch als *Knowledge Domain* bezeichnet – mit Hilfe einer standardisierten Technologie sowie Beziehungen und ggf. Ableitungsregeln zwischen den dort definierten Begriffen [Hes04]. Die Wissensrepräsentation setzt sich aus drei Bereichen anderer wissenschaftlicher Felder zusammen [Sow00]:

1. **Logik** stellt die formale Struktur bereit, um Regeln zu formulieren mit deren Hilfe das Computersystem Rückschlüsse bilden kann.

2. **Ontologien** definieren die Objekte, die in einem bestimmten Umfeld existieren.
3. **Berechenbarkeit** ist eine Eigenschaft einer Wissensbasis, die diese vom Umfeld der puren Philosophie abgrenzt.

Ohne Logik ist eine Wissensrepräsentation unklar, da keine Kriterien existieren, um zu prüfen, ob bestimmte Aussagen überflüssig, redundant oder sogar inkonsistent sind. Ohne eine Ontologie, können die Aussagen nur schwer bestimmt werden und sind verwirrend, da diese nicht ausformuliert wurden. Zuletzt ist es nicht möglich, die beiden wissenschaftlichen Felder Logik und Ontologie auf einem Computersystem zu implementieren, wenn diese nicht berechenbar sind [Sow00]. Im Folgenden werden die einzelnen Komponenten einer Wissensbasis näher untersucht.

2.2 Logik

Innerhalb einer Wissensbasis ist Logik nicht zwingend erforderlich. Es können durchaus auch ohne Logik Informationen gewonnen werden. Informationen ohne logische Verknüpfungen haben die Qualität von relationalen Datenbanken, da hierdurch lediglich Daten aus bestimmten Beständen abgefragt werden können. Erst durch die Logik wird es möglich, Informationen so zu verknüpfen, dass diese im Vergleich mit einer relationalen Datenbank bessere Resultate erzielen. Dies resultiert daher, dass es durch logische Verknüpfungen und Querverweise - ähnlich wie in Abbildung 1 in der Einleitung gezeigt – möglich wird, Informationen zu verbinden, die keine direkte Beziehung zueinander besitzen. Im Folgenden werden die grundlegenden Techniken zur Integration von Logik im Semantischen Web näher erläutert.

Die einfachste Art der Logik ist die *Aussagenlogik*, die sich mit der logischen Bewertung von Aussagen befasst. Diese kann als eine moderne Variation von Boolscher Algebra angesehen werden. Bei der Aussagenlogik werden Operatoren wie Negation ($\neg$), Konjunktion ($\wedge$) , Disjunktion($\vee$), Implikation ($\Rightarrow$) und Äquivalenz ($\equiv$) eingesetzt [Sow00]. Durch Integration von Aussagenlogik im Semantischen Web wird es möglich, einfache zusätzliche Informationen zu hinterlegen. Beispielsweise ist es bei der Zusammenführung von mehreren Wissensbasen zu einer einzigen sehr hilfreich, bestimmte gleichartige Elemente als äquivalent kennzeichnen zu können. Dadurch wird es möglich, dem Semantischen Netz mitzuteilen, dass beispielsweise ein in der Wissensbasis A definiertes Element *Stockwerk* gleichwertig mit einem in der Wissensbasis B definiertem Element *Etage* ist. Das Element *Etage* stammt möglicherweise aus fremdsprachigen Wissensbasis und wurde aus Kompatibilitätsgründen in die eigene aufgenommen.

Bereits die alten Griechen beschäftigten sich mit der Logik. Aristoteles erkannte Logik als eine präzise Methode, um Erkenntnisse über Wissen zu gewinnen. Der *Syllogismus*, ein Mechanismus, der drei Aussagen enthält und durch den eine logische Ableitung durchgeführt werden kann, geht auf Aristoteles zurück [Sow00]. Das folgende Beispiel zeigt einen Syllogismus:

> Wenn alle Reptilien aus dem Ei schlüpfen, und alle Schlangen Reptilien sind, dann schlüpfen alle Schlangen aus dem Ei.

Diese Aussage ist in unserer normalen Sprache definiert, weshalb es zu den bereits im Abschnitt 1.1 gezeigten Probleme führen kann, da menschliche Sprache oftmals ungenau ist. Aus diesem Grund unterteilte bereits Aristoteles seinen Syllogymus in ein *Subjekt* und ein *Prädikat*. Im obigen Beispiel wäre in der ersten Zeile das Subjekt *alle Reptilien* und das Objekt und sein Verb bilden zusammen das Prädikat [Sow00].

Aus dem von Aristoteles entwickeltem Konzept entstand die *Prädikatenlogik*, welche als Erweiterung der Aussagenlogik angesehen werden kann. Die Grundlagen wurden von Gottlob Frege im Jahr 1879 entwickelt [Sow00]. Die Prädikatenlogik gibt einen formalen Rahmen für konkrete Schlussfolgerung und darüber hinaus für viele andere weniger offensichtliche Fälle. Häufig spricht man präziser von Prädikatenlogik erster Stufe, auch als *first-order predicate calculus* oder *first order logic (FOL)* bezeichnet. Der Beispielsyllogismus wird in der Prädikatenlogik folgendermaßen dargestellt:

Alle Reptilien schlüpfen aus dem Ei.

$$\forall rep.\exists ei : schluepfen(rep, ei) \tag{2.1}$$

Alle Schlangen sind Reptilien.

$$\forall snake.\exists rep : is(schlange, rep) \tag{2.2}$$

Alle Schlangen schlüpfen aus dem Ei.

$$\forall snake.\exists ei : schluepfen(schlange, ei) \tag{2.3}$$

Innerhalb des Beispiels wird mit dem $\forall$-Quantor (für alle) und dem $\exists$-Quantor (es existiert) gearbeitet. Der $\forall$-Quantor besagt, dass alle X ein Teil von Y sein müssen, damit die Regel zutrifft. Im obigen Beispiel wird dadurch ausgedrückt, dass **alle** Reptilien die es gibt, aus einem Ei schlüpfen müssen. Beim $\exists$-Quantor hingegen genügt es wenn ein X Teil von Y ist. Die Detailgenauigkeit einer in Prädikatenlogik definierten Regeln hängt von der Menge an Prädikaten ab, welche nicht direkt zur Logik gehören, sondern innerhalb einer Wissensbasis beschrieben sind. In Abhängigkeit davon, welche Prädikate für ein Logikkonstrukt verwendet werden, entstehen unterschiedliche Bindungen [Sow00].

Im Bezug auf das Semantische Web, wird es durch Prädikatenlogik möglich, bestimmte Regeln zu definieren, die zusätzliche Informationen aus der Datenbank schöpfen. Beispielsweise könnte man innerhalb einer Wissensbasis über Tiere, den obigen Syllogismus definieren. Dadurch würde im System nur die Information hinterlegt werden müssen, ob ein Tier aus einem Ei schlüpft, um zu prüfen, ob der Syllogismus gültig ist und das Tier damit ein Säugetier oder ein Reptil ist. Selbstverständlich wäre diese Prüfung, mit einer relationalen Datenbank durchführbar. Jedoch ist es der Netzwerkeffekt, der in der Lage ist, über verschiedene Wissensbasen einfach zu iterieren sowie die leichte Erstellung von Regeln, welche die zukünftige Überlegenheit semantischer Netze zeigen wird.

Zur Integration von Logik stehen in Ontologiesprachen wie OWL (s. 3.3) Konzepte, wie beispielsweise Inversität, Transitivität oder Disjunktivität zur Verfügung, mit welchen sich logische Relationen verwenden lassen. Die folgende Abbildung 2.1 zeigt ein Beispiel für eine transitive Beziehung zwischen verschiedenen Fächern. In dieser wird dargestellt, dass *objektorientierte Datenbanken* ein Teilgebiet von *Datenbanken* sind. Weiterhin sind *Datenbanken* ein Teilgebiet der *Informatik*, deshalb sind *objektorientierte Datenbanken* auch ein Teilgebiet der *Informatik*. Besonders interessant sind durch Transitivität definierte Mehrfachverbindungen, z.B. *neuronale Netzte* sind nicht nur ein Teilgebiet der *künstlichen Intelligenz*, sondern auch der *Biologie*.

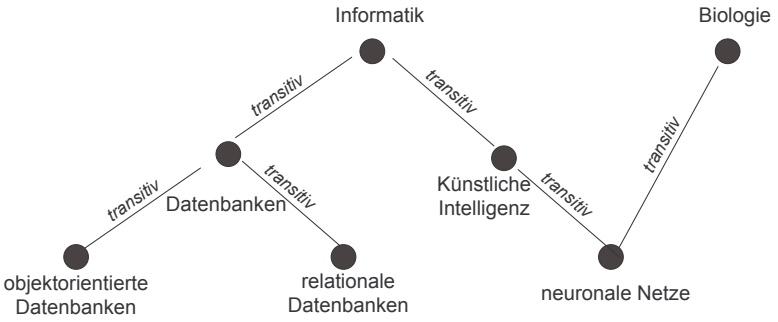


Abbildung 2.1: Transitive Beziehung innerhalb der Fächer

Transitivität lässt sich mittels relationalen Datenbanken nur sehr schwierig realisieren. Innerhalb von Semantischen Netzen, kann mit Hilfe von Transitivität, leicht ein semantisches Netzwerk gewonnen werden.

Des Weiteren ist der Einsatz von Logik sehr hilfreich beim direkten Modellieren von Semantischen Netzen. Wie im Kapitel 1 gezeigt, werden Beziehungen in einer bestimmten Wissensbasis modelliert. Dabei können ungewollte Inkonsistenzen auftreten. Um dies zu vermeiden kann ein so genannter *Reasoner*, z.B. Racer [HMW04], verwendet werden. Dabei handelt es sich um ein Programm, welches in der Lage ist, eine in der OWL-Syntax erzeugte Ontologie, auf logische Fehler zu überprüfen. Durch den Reasoner kann erkannt werden, ob bestimmte Regeln eine Inkonsistenz erzeugen. Des Weiteren können in der Entwicklung befindlich Ontologie-Modelle mit notwendigen Regeln ergänzt und fehlende Beziehungen erkannt werden. Dies kann bei großen Ontologien hilfreich sein.

2.3 Taxonomien

Bei einer Taxonomie handelt es sich um ein Konzept, welches ein wichtiger Bestandteil jeder Ontologie darstellt. Taxonomien kommen in vielen Bereichen zum Einsatz. Die menschliche Natur hat geradezu ein grundlegendes Bedürfnis, welches die Erstellung von Taxonomien betrifft [DOS03].

Allgemein formuliert ist eine Taxonomie eine Klassifizierung einer Sache, d.h. dass etwas einen bestimmten Namen erhält und strukturiert in einer Wissensbasis eingeordnet wird. Dabei erfolgt die Klassifizierung in Form einer einfachen Hierarchie, i.d.R. als Baumstruktur mit Knoten und Zweigen dargestellt. Innerhalb der Hierarchie hat jeder Knoten

nur einen Vorgängerknoten. Weiterhin besitzt jede Taxonomie eine Wurzel, welche den Hauptknoten – auch als *root* bezeichnet – darstellt.

Die folgende Abbildung 2.2 zeigt einen Ausschnitt aus der Taxonomie der deutschen Wikipedia-Webseite [Wik05a], welche aus 21 Hauptkategorien besteht. Unterhalb der Hauptkategorien existiert eine große Anzahl an Unterkategorien, so dass der Baum sehr schnell in Breite und Tiefe wächst. Die eigentlichen Informationen werden an der passenden Stelle innerhalb des Baumes eingeordnet. Falls kein passender Knoten gefunden wird, ist die Taxonomie dank der einfachen Struktur sehr leicht erweiterbar.

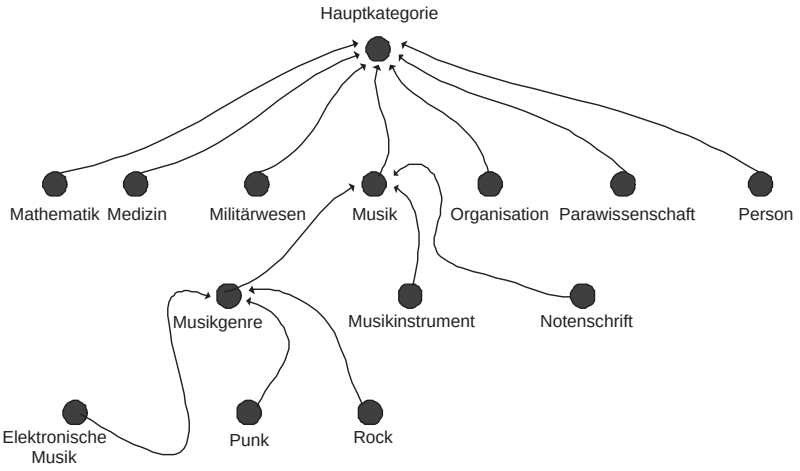


Abbildung 2.2: Taxonomieausschnitt der deutschen Wikipedia

Taxonomien sind sehr hilfreich, um Informationseinheiten semantisch zu klassifizieren. Tiefer verschachtelte Zweige der Taxonomie enthalten spezifischeres Wissen, während Knoten, die näher an der Wurzel liegen allgemeinere Informationen enthalten. Durch diese Klassifizierung von Wissensbereichen innerhalb einer Hierarchie entsteht eine einfache Semantik [DOS03]. Weitere Beispiele für Taxonomien sind die Ordnerstruktur eines Dateisystems, Yahoo! Directory [Tea05g] oder die Fächerklassifikation in Bibliotheken, z.B. Regensburger Verbundklassifikation [Lei05]. Innerhalb der Biologie und Medizin existieren bereits sehr große Taxonomien, welche Lebensformen oder Krankheiten klassifizieren.

Wie bereits erwähnt, besitzen Taxonomien einen einfachen Aufbau. Dadurch ist es nicht möglich, dass ein Knoten mehrere Vorgängerknoten besitzt. In bestimmten Fällen kann es jedoch passieren, dass es die Klassifikation erfordert, einen Knoten in zwei Kategorien einzuordnen, beispielsweise hat der Knoten *Musikinstrumente* in der obigen Abbildung 2.2 den Vorgängerknoten *Musik*, was durchaus logisch erscheint. Jedoch ist auch die Produktion der Instrumente eine besondere Kunst, die dem Bereich *Handwerk* zugeordnet werden müsste. Daher ist eine Taxonomie oftmals nur eine Sammlung von Begriffen [DOS03].

Der normale Einsatzbereich einer Taxonomie liegt darin, dass Anwender innerhalb einer Informationsstruktur navigieren können, vor allem dann, wenn diese nur eine allgemeine Idee haben, nach was gesucht werden soll [DOS03]. Eine deutliche Erweiterung von Taxonomien sind Ontologien welche auf Taxonomien aufbauen.

2.4 Ontologien

Für die Repräsentation von komplexen Wissensbeziehungen hat sich in der Informatik der Begriff Ontologie eingebürgert. Der wohl bekannteste Definitionsversuch stammt von T. Gruber [Gru93]. Dieser bezeichnet Ontologien als

> explizite formale Spezifikation einer gemeinsamen Konzeptualisierung.

Eine Ontologie ist eine begriffliche Abbildung eines bestimmten Gebiets. Der große Unterschied zur Taxonomie ist der, dass die Ontologie ein Netzwerk von Informationen darstellt, während die Taxonomie eine einfache Hierarchie bildet. Des Weiteren enthält eine Ontologie auch logische Relationen, d.h. es werden bestimmte Eigenschaften und Beziehungen von semantisch zusammenhängenden Elementen erzeugt. Ontologien bestehen aus verschiedenen Komponenten. Diese werden im Folgenden erklärt:

Konzepte

Eine Ontologie hat die Aufgabe, Objekte in der Welt zu gruppieren, die bestimmte ähnliche Eigenschaften besitzen, beispielsweise verschiedene Städte. Die Beschreibung dieser gemeinsamen Eigenschaften wird als Konzept definiert. Konzepte werden auch als *Klassen* bezeichnet. Diese können in einer Klassenstruktur mit Über-und Unterklasse angeordnet werden [HS04].

Instanzen

Instanzen repräsentieren Objekte in der Domäne und stellen das zur Verfügung stehende Wissen dar. Diese werden anhand vorher definierter Konzepte erzeugt und auch als *Individuals* bezeichnet [HS04].

Relationen

Instanzen vom gleichen Typ, müssen an verschiedene Gegebenheiten angepasst werden. Dazu werden Relationen verwendet, die Beschreiben, welche Beziehungen zwischen den Instanzen bestehen, beispielsweise „Stadt X liegt in Land Y“. Relationen werden auch als *Eigenschaften* bezeichnet.

Vererbung

Es ist möglich, Relationen und Konzepte zu vererben. Dabei werden alle Eigenschaften an das zu vererbenden Element weitergegeben. Mehrfachvererbung bei Konzepten ist grundsätzlich möglich. Durch den Einsatz von Transitivität, können Instanzen in einer Bottom-Up-Hierachie aufgebaut werden. Dabei spricht man von Delegation.

Axiome

Axiome sind Aussagen innerhalb der Ontologie, die immer wahr sind. Diese werden normalerweise dazu verwendet, um Wissen zu präsentieren, das nicht aus anderen Konzepten abgeleitet werden kann, z.B. „zwischen Amerika und Europa existiert keine Zugverbindung“ [GPFLC04].

2.4.1 Ontologietypen

Grundsätzlich unterteilt man Ontologien in zwei Typen:

1. *lightweight*-Ontologien beinhalten Konzepte, Taxonomien und Beziehungen zwischen Konzepten und Eigenschaften, welche diese beschreiben [GPFLC04].
2. *heavyweight*-Ontologien sind eine Erweiterung von *lightweight*-Ontologien und fügen diesen Axiome und Einschränkungen hinzu, wodurch die beabsichtigte Bedeutung einzelner Aussagen innerhalb der Ontologie klarer werden [GPFLC04].

Eine Ontologie ist abhängig davon, wer diese einsetzt. Beispielsweise kann es bei einer Ontologie über Weine für ein Restaurant wichtig sein, auch Speisen zu den passenden Weinen in der Ontologie aufzunehmen. Ist der Benutzer dagegen ein Weinabfüller, dann dürfte der Bereich der Speisen völlig uninteressant sein. Dagegen ist es für den Abfüller wichtig, welche verschiedenen Glas- und Flaschensorten existieren [NM00].

Eine Ontologie ist vergleichbar mit einem UML-Diagramm. Dieses modelliert nach dem Konzept der objektorientierten Softwareentwicklung einzelne Klassen, deren Eigenschaften sowie die Beziehungen zwischen den verschiedenen Klassen. Ontologien haben die gleiche Aufgabe. Es werden jedoch keine Softwareklassen modelliert, sondern einzelne Konzepte. Diese Konzepte haben – ähnlich einer Softwareklasse – bestimmte Eigenschaften und Beziehungen zu anderen Konzepten. Der Unterschied zur Softwareklasse liegt darin, dass ein Konzept eine einfache Beschreibung über ein bestimmtes Objekt darstellt. Innerhalb des Konzeptes werden ausschließlich Informationen hinterlegt, um dieses zu spezifizieren. Die Softwareklasse enthält ebenfalls Informationen über das zu modellierende Objekt, jedoch besteht der Hauptteil dieser aus Programmcode.

Sollen zwei Programme, z.B. Web-Suchmaschinen oder Software-Agenten, miteinander kommunizieren, so müssen sie entweder selbst die Interpretationsvorschrift für die Daten in sich tragen (sind also datenabhängig) oder aber sie liefern diese in Form von Metadaten aus einer beiden Seiten zugänglichen Ontologie mit. Beim automatischen Schließen können Programme logische Schlüsse schon aufgrund der per Ontologie bekannten Ableitungsregeln ziehen – diese müssen also nicht stets von Neuem übermittelt werden. Der Wert einer Ontologie steht und fällt mit dem Umfang der Anerkennung und Zustimmung, die diese in der betreffenden Fachwelt erfährt. Im Allgemeinen ist diese Zustimmung umso leichter zu erreichen, je mehr Entscheidungsträger und Betroffene am Entwurfsprozess beteiligt sind, andererseits steigt der Aufwand mit der Zahl der am Entwurf beteiligten Personen. Die wichtigsten noch ungelösten Probleme mit Ontologien betreffen aus heutiger Sicht die folgenden Punkte [Hes04]:

1. Wie lassen sich gut verwertbare Metadaten für sehr große Ressourcenbestände erzeugen und konsistent weiterentwickeln? Lässt sich das Annotieren von Dokumenten sinnvoll automatisieren?

2. Können Ressourcen klar und eindeutig klassifiziert werden, z.B. in Dokumente, Daten, Metadaten, physische und virtuelle Aktoren, physische Einheiten?

3. Wie lassen sich Metadaten in (möglicherweise überlappende und inkonsistente) Ontologien einordnen? Wie werden Synonyme, Homonyme und zirkuläre Definitionen behandelt?

4. Macht es Sinn, nach einer gemeinsamen, allen Ontologien unterliegenden Toplevel-, Universal- oder Meta-Ontologie zu suchen – oder führt dies zu ähnlichen Schwierigkeiten wie die Suche nach einem objektiven Weltbild in der Physik?

2.4.2 Beispielontologie

Die folgende Abbildung 2.3 zeigt das Funktionsprinzip einer Ontologie. Das Thema wurde aus der Diplomarbeit [Wle03] entnommen.

Die obere Ebene zeigt die Ontologie, welche Konzepte und Relationen enthält. Konzepte werden durch Ellipsen dargestellt und Relationen durch Pfeile. Die Rechtecke stellen einfache Container für Informationen, so genannte Literale dar, welche keine weiteren Beziehung zu anderen Konzepten besitzen (s. *Literale* in Kapitel 1). Die Relationen verbinden zwei Konzepte miteinander und schränken diese gleichzeitig ein, beispielsweise muss ein *Künstler* ein *Kunstwerk erzeugen*.

Wie bereits erwähnt, können Konzepte vererbt werden. Aus diesem Grund besitzen die *Maler* und *Bildhauer* ebenfalls die Relationen *Name* und *Vorname*. Der etwas dickere Pfeil kennzeichnet die Vererbung. Die Relationen *schlägt* und *malt* sowie *gemaltVon* und *geschlagenVon* sind vererbte Relationen von *erzeugt* und *hergestelltVon*, die jeweils für das entsprechende Konzept eingesetzt werden. Die ursprünglichen Relationseigenschaften bleiben dabei erhalten, können jedoch erweitert werden, z.B. durch eine Regel die besagt, dass das Ziel von *malt*, eine Instanz vom Typ *Bild* sein muss. Die Relationen *malt* und *gemaltVon* besitzen inverse Beziehungen zueinander, wodurch weitere Logik in die Ontologie integriert wird, die es ermöglicht, dass von einem Maler auf seine Kunstwerke und umgekehrt, von einem Bild zum Maler, geschlossen werden kann.

Die untere Ebene der Abbildung 2.3 zeigt Instanzen der Ontologie. Diese werden durch einen schwarzen Punkt dargestellt. Die erste Instanz mit Bezeichnung *I1* ist vom Typ Bild und besagt, dass es eine Person namens *Pablo Picasso* gibt. Das Kürzel *I1* steht dabei für den einmaligen Ressourcennamen der Instanz. Im Semantischen Web wird eine URI zur Kennzeichnung verwendet. Die weiteren Instanzen beschreiben das Werk und die Technik der Künstlers. Eine Besonderheit besitzt die Instanz des Malers *Raffaelo Santi*. Dieser verwendet bereits existierende Instanzen, nämlich *I3* vom Typ *Ölgemälde* und *I6* vom Typ *Galleria dell'Accademia*.

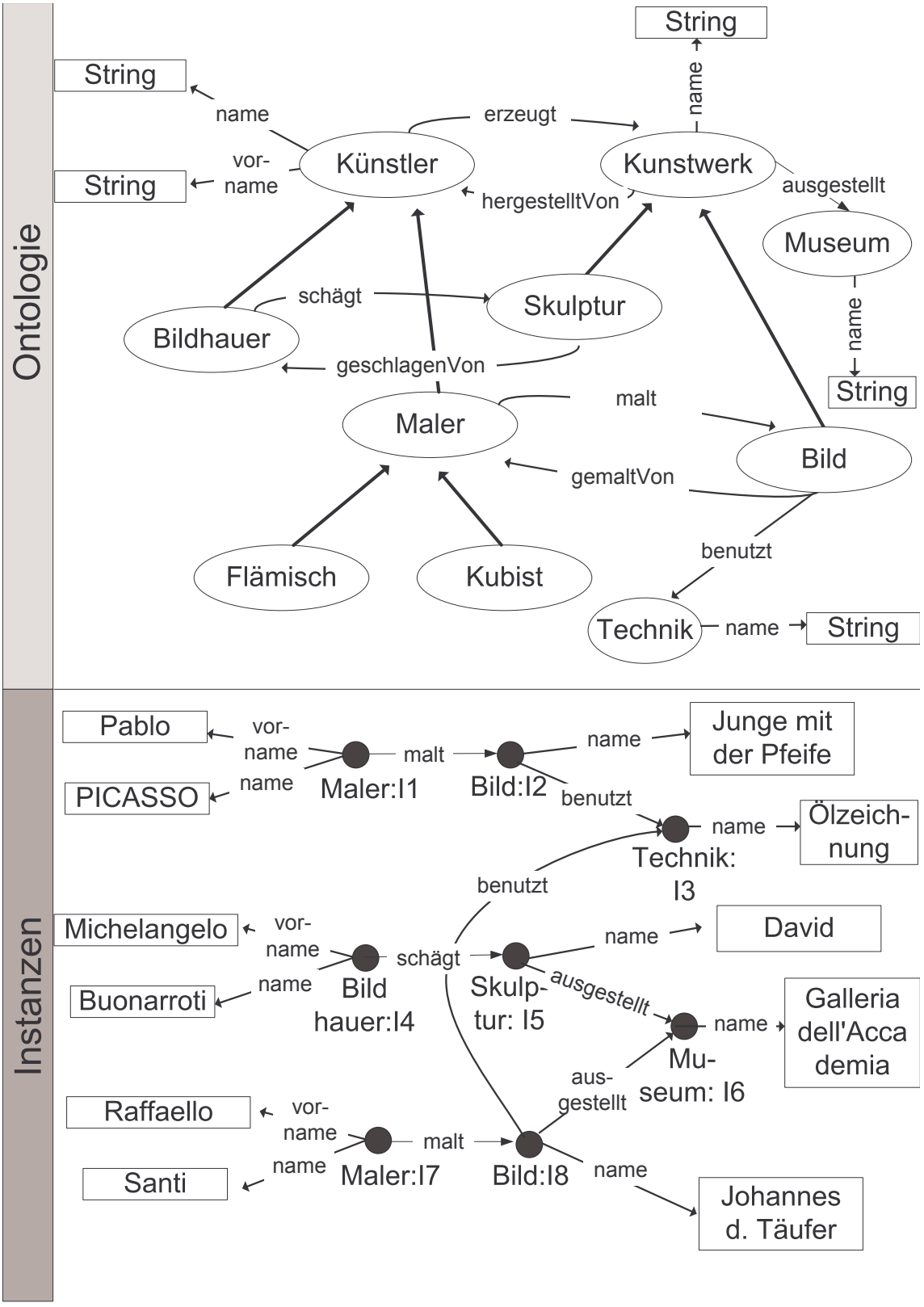


Abbildung 2.3: Schichten der Ontologie

2.4.3 Modellierung von Ontologien

Die Modellierung von Ontologien ist eine komplexe Aufgabe, die sorgfältig durchdacht werden muss. Bei diesem Vorgang spricht man auch von *Ontological Engineering*. Diese Bezeichnung macht nur beim Begriffsverständnis der Informatiker Sinn – in philosophischer Deutung könnten dies die Vorgehensweisen eines Demiurgen, also eines metaphysischen höheren Wesens sein. Ontological Engineering umfasst – analog zum Software-Engineering – alles, was zur Unterstützung des Ontologie-Lebenszyklus dienen kann [Hes04]. Dabei ist es empfehlenswert, folgende grundlegenden Aspekte sorgfältig zu überprüfen [NM00]:

- Welchen Bereich soll die Domäne abdecken?
- Aus welchen Gründen wird die Ontologie erstellt?
- Für welche Fragen muss die Ontologie Antworten bereit halten?
- Wer wird die Ontologie benutzten und pflegen?

Durch diese Fragen wird es möglich, den Bereich der Ontologie einzuschränken um dadurch Zeit und Geld zu sparen. Natürlich ist es auch wichtig die Ontologie auf mögliche Wiederverwendung zu prüfen.

Zurzeit existiert keine allgemeine gültige Methode um Ontologien zu entwerfen. Es gibt lediglich Ansätze aus dem Bereich der Künstlichen Intelligenz. Die beste Möglichkeit liegt immer im Auge des Betrachters und dessen Ansichten. Das Modellieren von Ontologien kann immer als iterativer Prozess angesehen werden, da jede Ontologie im Lauf der Zeit verfeinert wird [NM00]. Im Folgenden werden die wichtigsten Punkte zum Modellieren einer Ontologie aufgezeigt:

Bestimmen der Domäne und deren Bereiche

Am Anfang der Modellierung stellt sich die Frage, was genau abzubilden ist. Dazu können folgende Fragen hilfreich sein:

- Für welche Zwecke wird die Ontologie verwendet werden?
- Für welche Fragen muss die Ontologie Antworten bereithalten?
- Wer wird die Ontologie verwenden?

Die Antworten auf die obigen Fragen können sich im Verlauf des Entwicklungsprozesses verändern. Jedoch wird es am Anfang der Modellierung möglich, den Abzubildenden Bereich einzuschränken [NM00].

Vorhandene Ontologien wiederverwenden

Es ist immer sinnvoll zu prüfen ob bereits eine ähnliche Ontologie entwickelt wurde und ob diese für die eigenen Zwecke wiederverwendet werden kann. Im Internet existieren Ontologiedatenbanken [Med05b] auf denen frei verwendbare Ontologien hinterlegt sind.

Aufzählen der Konzepte

Im nächsten Schritt kann es sehr hilfreich sein, eine Liste der zu modellierenden Konzepte und der damit verbundenen Aussagen anzufertigen [NM00]. Dabei handelt es sich nicht um Relationen zwischen den Konzepten, sondern lediglich um eine Art Aufzählung die darstellt, was abgebildet werden muss, z.B. „jeder Wein soll passende Speisen auflisten“.

Aufbau der Konzepthierarchie

Es existieren verschiedene Möglichkeiten zum Aufbau einer Ontologie [HS04]. Zwei davon werden im Folgenden erläutert:

1. Der *Top-Down*-Entwurf beginnt mit den allgemeineren Konzepten und erzeugt danach detailliertere Unterklassen [NM00].
2. Der *Bottom-Up*-Entwurf modelliert zuerst das spezifischere Wissen. Dieses wird danach den allgemeineren Klassen zugeordnet [NM00].

Unabhängig davon, welche Methode gewählt wird, entsteht an dieser Stelle eine Taxonomie, da die einzelnen Konzepte zuerst ohne Relationen oder logischen Aspekte erzeugt werden.

Definieren der einzelnen Relationen

Konzepte alleine reichen noch nicht aus, um komplexe Antworten zu liefern. Dazu werden Relationen benötigt, welche die Eigenschaften von Klassen beschreiben und erst dadurch vollständige Konzepte bilden. Aus diesem Grund werden an dieser Stelle die einzelnen Relationen definiert.

Bestimmen der Relationsaspekte

Konzepte können bestimmte Aspekte zugewiesen bekommen wodurch diese exaktere Formulierungen innerhalb der Ontologie ermöglichen. Im Folgenden werden einige Aspektmöglichkeiten aufgezeigt:

- Die *Kardinalität* bestimmt die Anzahl der Werte, die eine Relation zugewiesen bekommen kann.

- Der *Datentyp* bestimmt die Möglichkeiten, wie eine Ressource beschrieben werden kann.
- Die *Domäne* bestimmt die Klassen, welche die Relation benutzen kann.
- Der *Bereich* oder *Range* gibt an, mit welchen anderen Klassen die Relation verbunden werden kann.

Integration von logischen Aspekten

Die Integration von logischen Aspekten sollte der letzte Modellierungschritt für die Ontologie darstellen. Darin können Axiome formuliert sowie die bereits im Abschnitt 2.2 aufgezeigten Möglichkeiten zur Integration von prädikatenlogischer Ausdrücke angewendet werden. Dabei ist zu beachten, dass nicht jede Ontologiesprache alle logischen Konstrukte unterstützt.

Erzeugen der Instanzen

Nach dem Erzeugen der eigentlichen Ontologie können Instanzen angelegt werden, welche das eigentliche Wissen repräsentieren, mit welchem später gearbeitet wird [NM00]. Diese Instanzen stammen von einem oder mehreren Konzepten ab und enthalten das eigentliche Wissen.

2.5 Bestandteile des Semantischen Webs

Zusammenfassend folgt als letzter Abschnitt dieses Kapitels die theoretische Beschreibung der vorgestellten Konzepte in Bezug auf das Semantische Web. Die folgende Abbildung 2.4 zeigt den vom W3C vorgeschlagenen Aufbau für das Semantische Web.

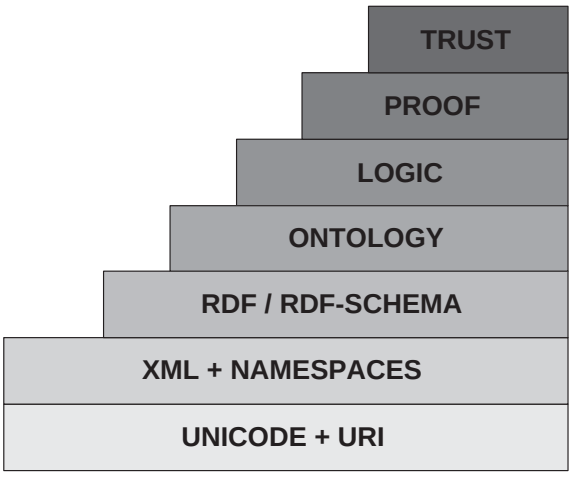


Abbildung 2.4: Schichten des Semantischen Webs [BL04a]

Die einzelnen Schichten haben folgende Bedeutung:

1. Schicht: Unicode und URI
   - *Unicode* umfasst ein universelles Zeichenrepertoire, das alle gängigen Zeichen der Welt umfasst. Damit ist es möglich, dass alle Terraner am Semantischen Web teilnehmen. Der heutige Standard ist als UTF-8 spezifiziert und noch in der Weiterentwicklung.
   - *URI* ist ein universeller Bezeichner, der im Vergleich zur URL nicht zwingend eine Webseite referenzieren muss. Eine URI kann als Identifier angesehen werden.
2. Schicht: XML und Namensräume
   - *XML* ist eine Sprache zur Definition von Auszeichnungssprachen.
   - *Namensräume* sind ein Konzept, welches aus dem Bereich der Programmiersprachen stammt. Dieses hat das Ziel, mehrere spezifizierte XML-Sprachelemente in einem XML-Dokument zu verwenden.
   - *XML-Schema* ist eine Möglichkeit zur Modellierung und Strukturierung von XML-Dokumenten.
3. Schicht: RDF und RDF-Schema
   - *RDF* ist eine Sprache zur Beschreibung von Metadaten.
   - *RDF-Schema* definiert Vokabulare und die Beziehungen, auf denen die in RDF erzeugten Metadaten beruhen dürfen.
4. Schicht: Ontologien
   - *Ontologien* sind Sammlungen von Vokabularen, die einen bestimmten Bereich aus dem Leben abbilden. Innerhalb der Ontologien werden Beziehungen und Möglichkeiten beschrieben, wie sich einzelne Elemente verhalten können.
5. Schicht (in der Entwicklung): Logik
   - *Logik* erweitert das Semantische Web um die Möglichkeit, Regeln zu definieren, die von Programmen verarbeitet werden können.
6. Schicht (in der Entwicklung): Proof
   - *Proof* dient zur Prüfung der erzeugten Ontologien auf Konsistenz.
7. Schicht (in der Entwicklung): Trust
   - *Trust* dient zur Sicherheit. Darin wird spezifiziert, welche Programme – oftmals Web-Services – welche Aktionen, stellvertretend für den Benutzer durchführen dürfen.

Kapitel 3

Konzepte des Semantischen Web

Das folgende Kapitel beschreibt die verschiedenen Sprachen, die zum Aufbau eines Semantischen Webs einsetzbar sind. Dabei werden die grundlegenden Möglichkeiten aufgezeigt und anhand von Beispielen näher erläutert. Im Folgenden kann nicht auf alle Semantischen Web-Sprachen aus Platz- und Zeitgründen eingegangen werden. Deshalb werden ausschließlich die Sprachen RDF/S und OWL erläutert.

3.1 Resource Description Framework

Das *Resource Description Framework (RDF)* ist eine Sprache, um Informationen zu veröffentlichen. RDF ist eine Entwicklung des W3C (World Wide Web Consortium) und steht frei zur Verwendung. Das Framework wurde zur Unterstützung der Aktivitäten im Umfeld des Semantischen Webs entworfen. Das W3C beschreibt RDF folgendermaßen [Pow03]:

> The Resource Description Framework (RDF) is a language designed to support the Semantic Web, in much the same way that HTML is the language that helped initiate the original Web. RDF is a framework for supporting resource description, or metadata (data about data), for the Web. RDF provides common structures that can be used for interoperable XML data exchange.

RDF ist bereits eine ältere Spezifikation des W3C. Eine erste Version erschien im Jahr 1999. Bereits im Jahr 2000 erfolgte die auf RDF aufbauende Spezifikation *Resource Description Framework Schema (RDFS)* [Pow03].

3.1.1 Ziele von RDF

Wie bereits im Kapitel 1 näher erläutert, ist es das Ziel des Semantischen Webs, Informationen zu veröffentlichen, die durch Maschinen besser verarbeitet werden können, als herkömmliche Daten. Aus diesem Grund müssen Informationen in einem für Maschinen

und Menschen lesbarem Format bereitgestellt werden, die von Applikationen, wie Web-Services, Suchmaschinen oder Agenten, verarbeitet aber auch von Menschen verstanden werden können. Ein weiteres Ziel von RDF ist es, einen weit effizienteren Datenaustausch zu realisieren, beispielsweise in Bereichen der Suche, Katalogisierung von Daten oder Klassifizierung [EE04].

3.1.2 Grundlagen von RDF

Mit RDF können Informationen über Web Seiten im World Wide Web zugänglich gemacht werden, die nicht auf der eigentlichen Webseite enthalten sind. Die Metadaten können dabei Informationen wie beispielsweise Titel, Name oder Copyright, enthalten. RDF besteht sowohl aus einem grafischen Modell zur Repräsentation der erzeugten Metadaten, als auch aus einer XML-Syntax mit gleichem Ziel. Beide Konzepte werden im Laufe dieses Kapitels mehrfach verwendet. Im Folgenden wird eine Einführung in die Sprache RDF gegeben. Diese zeigt im Detail, warum RDF eine Möglichkeit darstellt, um ein Semantisches Web umzusetzen. Zum Verständnis von RDF werden im Folgenden die beiden für RDF/XML wichtigen Konzepte URI und Namensräume näher erläutert. Kenntnisse über die Beschreibungssprache XML werden dabei vorausgesetzt.

Uniform Resource Identifier

Ein Uniform Resource Identifier (URI) ist die Standardsyntax, um Objekte im Semantischen Web zu identifizieren. Ungezwungen ausgedrückt, ist eine URI eine generische Benennung eines Objekts, das in der physikalischen Welt existiert und durch eine Adresse und einen Namen beschrieben wird, wodurch dieses Objekt im World Wide Web identifiziert werden kann. [DOS03].

Ein spezielles Protokoll, wie *http://* oder *ftp://*, kann – muss jedoch nicht – angegeben werden. Der Unterschied zwischen einer Uniform Resource Identifier und einer Uniform Resource Locator (URL) ist der, dass durch die URL gezeigt wird, wo ein Objekt, beispielsweise eine Webseite, liegt, während durch die URI einerseits der Name, und andererseits die Lage des Objektes beschrieben wird [Pow03].

Namensräume

Zur Vermeidung von Namensraumkonflikten, d.h. dass bestimmte Bezeichner mehrfach vorkommen, können Namensräume, auch als *Namespaces* bezeichnet, herangezogen werden. Jeder Namensraum wird durch eine URI identifiziert. Namensräume erhalten i.d.R. ein Kürzel, über das der entsprechende Namensraum angesprochen werden kann. Hinter dem Namensraum wird der eigentliche Bezeichner angegeben.

Dabei ist die Idee nicht unbedingt die, dass die Namensraum-URI eine Adresse im Web darstellt, an der eine Datei steht, die Namen und alle Bezeichner enthält. Vielmehr werden

URIs im Zusammenspiel mit Namensräumen dazu eingesetzt, eindeutige Namen anzulegen [EE04]. Die folgende Liste zeigt verschiedene Standard-Namensräume sowie den Namensraum für das Semantische Web der Universitätsbibliothek Heidelberg.

Kürzel:	**Namensraum URI**	**Beschreibung**
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#	RDF
rdfs:	http://www.w3.org/2000/01/rdf-schema#	RDF-Schema
owl:	http://www.w3.org/2002/07/owl#	OWL
xsd:	http://www.w3.org/2001/XMLSchema#	XML-Schema
dc:	http://purl.org/dc/elements/1.1/	Dublin Core
ub:	http://www.ub.uni-heidelberg.de/	UB Heidelberg

Die Namensräume *rdf*, *rdfs* und *owl* sind vom W3C erzeugte Spezifikationen für die jeweilige Semantische Web Sprache. Jeder dieser Namensräume stellt die sprachabhängige Syntax zum Verwenden der entsprechenden Sprache bereit. Das Kürzel *dc* in der Zeile fünf, steht für den Namensraum des *Dublin Cores* (s. 3.1.3) und dient zum Beschreiben von Objekten durch Metadaten. Über diese Namensräume werden so genannte Standardeigenschaften definiert, welche zum Beschreiben der einzelnen Ressourcen verwendet werden sollten.

Die letzte Zeile der obigen Aufzählung zeigt einen Namensraum für die Universitätsbibliothek Heidelberg, welcher während der Masterthesis entstanden ist. Alle Namensräume werden in den folgenden Beispielen verwendet.

3.1.3 Dublin Core

Der Dublin Core ist ein Metadatenformat zur Beschreibung von Objekten. Beschrieben werden können sowohl elektronische als auch reale Objekte. Die Ziele der *Dublin Core Metadata Initiative* (DCMI) sind die Verbreitung von interoperablen domänenunabhängigen Metadaten-Standards, um Ressourcen so zu beschreiben, dass Suchmaschinen entstehen können, die diese Metadaten verarbeiten können [EE04].

Das Metadatenformat beschreibt eine Menge von Haupteigenschaften, anhand derer Dokumente, Datenquellen, Bilder etc. beschrieben werden können. Die Haupteigenschaften sind zusätzlich verfeinert worden, so dass eine exaktere Beschreibung des Objektes ermöglicht wird. Werden nur die Haupteigenschaften der Metadaten-Elemente *Dublin Core Metadata ElementSet* (DCMES) verwendet, spricht man vom einfachen Dublin Core [Boa99]. Wird auch das verfeinerte Metadaten-Schema benutzt, wird dies als der qualifizierte *Dublin Core Qualifiers* (DCQ) [Boa02b] bezeichnet. Im Folgenden werden die einzelnen Elemente des DCMES aufgezählt: contributor, coverage, creator, date, description, format, identifier, language, publisher, relation, rights, source, subject, title, type.

Eine Version des Dublin Cores existiert auch von RDF/XML. Die dazugehörige DTD [Boa02a] sieht vor, dass innerhalb des Tags *rdf:Description* die Dublin-Core-Elemente in beliebiger Reihenfolge beliebig häufig vorkommen können [Boa01]. Beim Dublin Core handelt es sich, genau wie bei wie bei einigen Elementen aus dem *rdfs*-Namensraum um Eigenschaften, die eine Ressource beschreiben. Die Eigenschaften des Dublin Cores sind dabei sehr gut geeignet um die Medien einer Bibliothek mit Metadaten auszuzeichnen.

Wie bereits erwähnt sollte immer, falls möglich, die Standardeigenschaften, stark verbreiteter Namensräume verwendet werden, um Interoperabilität zwischen Semantischen Diensten zu gewährleisten.

Zumindest die führenden Suchmaschinen im Web erkennen die Metaangaben die durch den Dublin-Core gemacht werden können. In HTML-Seiten können Dublin-Core-Metadaten als HTML-Metadaten angegeben werden. Dazu wird das Präfix *DC* oder *DC-TERMS* verwendet.

3.1.4 Das RDF-Modell

Das eigentliche RDF-Modell besteht aus den drei Objekttypen: *Ressourcen, Eigenschaftselementen* und *Objekten*. Jeweils eine Ressource, eine Eigenschaft und ein Objekt bilden zusammen ein so genanntes RDF-Tripel. Durch die Kombination von Subjekt, Prädikat und Objekt wird eine Aussage über ein bestimmtes Objekt innerhalb der Domäne gemacht, was auch als ein so genanntes *Statement* bezeichnet wird. Daher ist ein RDF-Modell, welches eine beliebige Anzahl an Subjekten, Prädikaten und Objekten besitzt, immer eine Sammlung von Statements, wodurch der gewünschte Netzwerkeffekt entsteht [Tea05b].

Innerhalb von RDF-Dokumenten werden Tripel in einer einfachen Art und Weise dargestellt, so dass es sowohl für Computer als auch für Menschen möglich ist, diese zu lesen und zu verstehen. Allgemein dargestellt hat ein RDF-Tripel den folgenden Aufbau:

```
{subjekt, prädikat, objekt}
```

Im Folgenden werden die Bestandteile der RDF-Tripel näher erklärt:

- **Ressourcen** (Subjekte) sind alle Dinge, die durch RDF-Ausdrücke beschrieben werden. Dies können einzelne Web-Seiten, Sammlungen von Web-Seiten, aber auch Objekte, auf die nicht über das Web zugegriffen werden kann, wie z.B. Bücher, Gemälde oder Computer sein. Das Wichtige dabei ist, dass die Ressource eine eindeutige Bezeichnung, beispielsweise durch eine URI, erhält.

  In der grafischen RDF-Modellierung werden Ressourcen durch eine Ellipse symbolisiert. Jede Ressource wird durch ihre Eigenschaften beschrieben [MM04].

- **Eigenschaftselemente** (Prädikate) haben in der sprachlichen Grammatik die Aufgabe, etwas über das Subjekt zu erläutern. Beispielsweise sagt in dem Satz „die Firma verkauft Batterien“ das Prädikat aus, was die Firma – das Subjekt – und damit die Ressource – macht, nämlich verkaufen [DOS03]. Bezogen auf RDF gibt ein Eigenschaftselement Auskunft über die ihm zugeordnete Ressource. Des Weiteren stellt das Eigenschaftselement einen Bezug zum Objekt her, verbindet damit eine Ressource mit einem Objekt.

  Prädikate werden grafisch durch eine benannte Kante dargestellt, die eine Ressource und deren Wert miteinander verbindet [MM04].

- **Objekte** beschreiben den Wert eines Prädikats. Im obigen Beispielsatz ist das Objekt Batterien. Dieses wird durch das Eigenschaftselement näher erläutert [DOS03]. Es existieren mehrere Möglichkeiten zur Darstellung von Objekten. Literale sind dabei die einfachste Art, da sie lediglich einen Wert zugeordnet bekommen. Literale werden in der grafischen Modellierung als Rechtecke dargestellt.

  Ein Objekt kann neben einem Literal auch eine Ressource oder eine leere Ressource sein. Diese Möglichkeiten werden in der folgenden Aufzählung näher beschrieben:

  – **Ressourcen** können auf weitere Ressourcen verweisen, beispielsweise um Redundanzen zu vermeiden.

  – **Leere Knoten** werden verwendet, wenn eine bestimmte Ressource noch nicht existiert oder wenn eine Ressource keinen Namen hat.

  – **Literale** enthalten Werte.

Ein erster großer Unterschied zwischen der semantischen und der syntaktischen Ebene ist derjenige, dass eine Ressource durch eine URI gekennzeichnet wird. Diese URI ist einmalig und kann überall auf der Welt genauestens bestimmt werden, wodurch jedes einzelne Konzept eindeutig identifiziert wird, auch wenn es gleiche Eigenschaften besitzt [DOS03]. Wie bereits erwähnt, wird die Ressource durch einen eindeutigen Bezeichner, meist eine URI präsentiert. Dadurch kann der oben eingeführte Beispielssatz folgendermaßen durch ein RDF-Tripel repräsentiert werden:

```
{http://www.firma.de, verkauft, "Batterien"}
```

3.1.5 Der RDF-Graph

Die RDF-Entwicklungsgruppe am W3C entschied sich dafür, den RDF-Graph als Standardentwicklungsmethode zu verwenden, da ein RDF-Graph, wie in den folgenden Beispielen gezeigt, sehr einfach für Menschen zu lesen ist [Pow03].

Die folgende Abbildung 3.1 zeigt ein einfach modelliertes Tripel in der vorgestellten grafischen Syntax, welches man auch als gerichteter Graph (Directed Graph) bezeichnet.

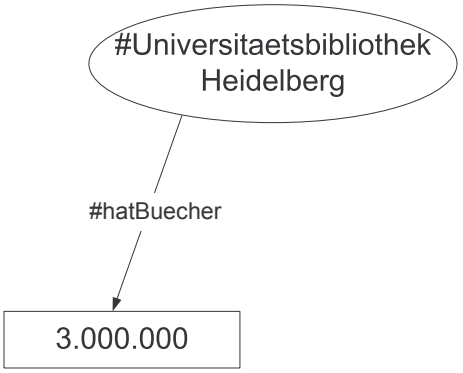


Abbildung 3.1: Einfaches RDF-Tripel

Der RDF-Graph in der obigen Abbildung sagt aus, dass die Ressource *Universitätsbibliothek Heidelberg* im Besitz von 3 Millionen Büchern ist. Zur Modellierung des Objektes wird ein Literal verwendet.

Das Gatter-Zeichen wird vor jedes Element gestellt, um zu zeigen, dass dieses aus dem Standardnamensraum stammt (s. 3.1.6). Ein gerichteter RDF-Graph besteht aus einer beliebigen Anzahl an Knoten, die durch Bögen miteinander verbunden werden und dadurch eine *Knoten-Bogen-Knoten* Struktur formen. Die folgende Abbildung 3.2 zeigt eine Ressource von der zwei gerichtete Graphen ausgehen.

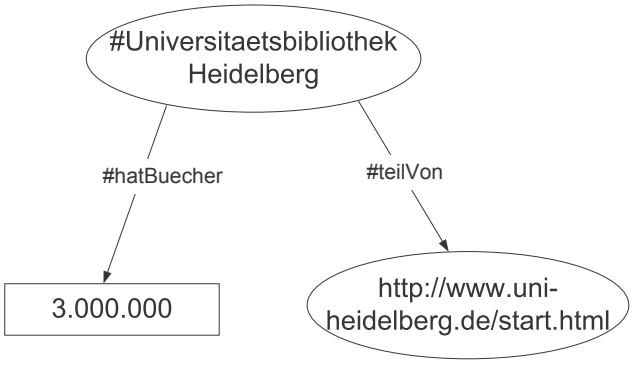


Abbildung 3.2: RDF-Tripel mit 2 Ressourcen

Der erste Graph zeigt auf das in der vorherigen Abbildung beschriebene Literal. Der neue Graph zeigt auf eine weitere Ressource, nämlich die Universität Heidelberg. Da es sich bei der Universität Heidelberg ebenfalls um ein Objekt handelt, das wiederum Eigenschaften und Beziehungen zu anderen Objekten besitzt, muss diese als Ressource und nicht als Literal dargestellt werden. Von der Ressource Universitätsbibliothek Heidelberg können, beispielsweise weitere Eigenschaftselemente wie *hatStudenten* oder *hatFakultäten* ausgehen, die für dieses Modell jedoch nicht benötigt und deshalb nicht modelliert werden.

Die Ressource welche die Universität Heidelberg beschreibt, stammt nicht aus dem Standardnamensraum. Aus diesem Grund wird diese Ressource durch eine vollständige URI beschrieben.

RDF-Anforderungen

Wie bereits erwähnt, wird für die Arbeit mit RDF die Auszeichnungssprache XML eingesetzt. Aus diesem Grund entsteht eine Mischung von RDF-Semantik und XML-Syntax, welche man als *RDF/XML* bezeichnet.

Es existiert eine weitere Möglichkeit, die Anforderungen von RDF zu erfüllen, welche als die N3-Notation bezeichnet wird. Diese kann daran erkannt werden, dass zur Markierung der Ressource die beiden Zeichen < > verwendet werden [Pow03]. In der Masterthesis wird die Standardnotation RDF/XML verwendet. Im Folgenden wird das obige Beispiel 3.1.4 in in N3-Notation angegeben:

```
<http://www.firma.de> verkauft "Batterien"
```

3.1.6 Die RDF-Basis-Syntax

Das folgende Beispiel 3.1 ist eine Einführung in das Konzept von RDF/XML. Das Wurzelelement einer RDF-Instanz ist *RDF*, dem *rdf:* für den Namensraum vorangestellt sein sollte [EE04].

```
1  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
2         xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
3         xmlns:ub="http://www.ub.uni-heidelberg.de/ns#"
4         xml:base="http://www.ub.uni-heidelberg.de">
5
6    <rdf:Description rdf:ID="UniversitaetsbiliothekHeidelberg">
7      <rdfs:label xml:lang="de">Universitätsbibliothek Heidelberg</
           rdfs:label>
8      <ub:hatBuecher>3000000</ub:hatBuecher>
9      <ub:email>mailto:ub@uni-heidelberg.de</ub:email>
10   </rdf:Description>
11 </rdf:RDF>
```

Beispiel 3.1: RDF-Syntax

In den ersten drei Zeilen werden die benötigten Namensräume eingebunden, in diesem Fall der *rdf/rdfs*-Namensraum sowie der selbst spezifizierte Namensraum *ub* für die Universitätsbibliothek Heidelberg (s. 3.1.2). Danach wird mit *xml:base* der Standardnamensraum angegeben.

Wie bereits erwähnt, verwendet RDF so genannte RDF-Tripel, welche aus einer Ressource, einem Eigenschaftselement und einem Objekt besteht. In diesem Beispiel wird die Ressource innerhalb des so genannten *Description*-Elements, durch das Attribut *rdf:id* beschrieben. Die Bezeichnung der Ressource geschieht durch Angabe eines einfachen Namens nach dem Identifier-Attribut, nämlich *UniversitaetsbiliothekHeidelberg*. In Zusammenhang mit dem Standardnamensraum wird daraus automatisch die folgende URI gebildet: *http://www.ub.uni-heidelberg.de#UniversitaetsbiliothekHeidelberg*. Dabei ist zu beachten, dass zwischen dem Standardnamensraum und dem eigentlichen Namen ein Trennzeichen in Form eines Gatters # angegeben ist. Durch das *ID* Attribut können die oftmals sehr langen URIs in gekürzter Weise im RDF-Dokument beschrieben werden [EE04].

Eine weitere Methode kann mit Hilfe des Attributs *rdf:about* erzeugt werden. Dies kommt zum Einsatz, wenn Ressourcen erzeugt werden sollen, die im Web verfügbar oder wenigstens durch eine URL referenzierbar sind. Soll beispielsweise als URI der obigen Ressource, die Startseite der Universitätsbibliothek Heidelberg verwendet werden, dann muss diese mit dem *rdf:about* Attribut geschehen, beispielsweise *rdf:about="http://www.ub.uni-heidelberg.de/start.html"*. Das *rdf:about* Attribut wird immer dann verwendet, wenn es gilt eine Ressource mit einer URL zu beschreiben.

Alternativ kann mit ebenfalls mit dem *rdf:about*-Attribut eine Ressource mit Hilfe einer URIs in Kurzform beschrieben werden. Dazu muss das Gatter-Zeichen mit angegeben werden, z.B. *rdf:about="#UniversitaetsbiliothekHeidelberg"*.

Ein RDF/XML-Dokument kann beliebig viele *Description*-Elemente enthalten, von denen jedes eine Ressource beschreibt. Die Eigenschaften *rdfs:label*, *ub:hatBuecher* und

ub:email stellen die Eigenschaftselement (Prädikate) dar. Diese zeigen Beziehungen zwischen der Ressource und den Literalen oder anderen Ressourcen an. Die entsprechenden Namen der Eigenschaftselemente sind frei definierbar, lassen sich durch den Einsatz von RDFS jedoch einschränken [EE04]. Die Namensraumangabe ist dabei optional, da es sich um den Standardnamensraum handelt, wird jedoch der Übersichtlichkeit wegen angegeben. Gültig wäre genauso die Angabe von beispielsweise *<hatBuecher>3000000</hatBuecher>*.

Wichtig ist es, dass man Eigenschaften verwendet, die bereits in anderen Namensräumen definiert sind. Dies geschieht durch die erste Eigenschaft *rdfs:label*, welche dazu verwendet wird, den Namen jeder Ressource in beliebiger Sprache anzugeben.

Ressource-Attribut

Wie bereits erwähnt, ist es möglich, dass eine Ressource auf eine weitere Ressource verweist, beispielsweise um Redundanzen zu vermeiden. Um die Eigenschaft in RDF/XML zu verwenden, existiert das *rdf:resource*-Attribut. Diese gibt an, dass der Wert des Eigenschaftselements durch eine andere Ressource beschrieben wird, die durch die angegebene URI identifiziert werden kann [MM04]. Das folgende Beispiel 3.2 verdeutlicht diese Möglichkeit. Dies wurde bereits in der Abbildung 3.2 modelliert.

```
1  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
2         xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
3         xmlns:ub="http://www.ub.uni-heidelberg.de/ns#"
4         xml:base="http://www.ub.uni-heidelberg.de">
5
6  <rdf:Description rdf:ID="UniversitaetsbibliothekHeidelberg">
7    <rdfs:label>Universitätsbibliothek Heidelberg<rdfs:label>
8    <ub:hatBuecher>3000000<ub:hatBuecher>
9    <ub:teilVon rdf:resource="http://www.uni-heidelberg.de/start.html"/>
10 </rdf:Description>
11
12 <rdf:Description rdf:about="http://www.uni-heidelberg.de/start.html">
13   <rdfs:label xml:lang="de">Universität Heidelberg<rdfs:label>
14   <rdfs:label xml:lang="en">University Heidelberg<rdfs:label>
15 </rdf:Description>
16 </rdf:RDF>
```

Beispiel 3.2: RDF Ressource-Attribut

Die beiden Elemente *rdfs:label* und *ub:hatBuecher* zeigen auf einfache Literale. Das *ub:teilVon* Eigenschaftselement in Zeile 4 ist auf eine Eigenschaft gerichtet, deren Wert eine andere Ressource ist. Dies wird durch das *rdf:resource*-Attribut angegeben. Diese weitere Ressource wird über die URI *http://www.uni-heidelberg.de* beschrieben. Dabei gilt zu beachten, dass diesmal zur Beschreibung der Ressource, das *rdf:about*-Attribut verwendet wurde, um direkt die URL der Universität Heidelberg zu verwenden.

Ab Zeile 7 wird diese Ressource beschrieben, wobei ein anderer Namensraum *uni* aus Gründen der Modellierung verwendet wird. Sollen mit dem *resource*-Attribut Ressourcen beschrieben werden, die im Standardnamensraum liegen,

genügt die Angabe des Ressourcenames mit vorgestelltem Gatter-Zeichen, z.B. *rdf:resource=“#UniversitaetHeidelberg“*.

Eine andere Möglichkeit, um von einer Ressource auf eine andere zu verweisen, besteht darin, dass die beiden Ressourcen ineinander verbunden werden. Dies wird im Beispiel 3.3 dargestellt, welches gleichwertig zum vorherigen Beispiel 3.2 ist.

```
<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:ub="http://www.ub.uni-heidelberg.de/ns#"
       xml:base="http://www.ub.uni-heidelberg.de">

<rdf:Description rdf:ID="UniversitaetsbibliothekHeidelberg">
  <rdfs:label>Universitätsbibliothek Heidelberg<rdfs:label>
  <ub:hatBuecher>3000000<ub:hatBuecher>
  <ub:teilVon>
    <rdf:Description rdf:about="http://www.uni-heidelberg.de/start.
        html">
      <rdfs:label xml:lang="de">Universität Heidelberg<rdfs:label>
      <rdfs:label xml:lang="en">University Heidelberg<rdfs:label>
    </rdf:Description>
  <ub:teilVon>
</rdf:Description>
</rdf:RDF>
```

Beispiel 3.3: Zwei ineinander verbundene Ressourcen

Durch das Verschachteln der beiden Ressourcen kann die Beziehung zwischen den Ressourcen erkannt werden. Für die Modellierung ist es jedoch sinnvoller, diese getrennt zu halten, da dadurch eine bessere Übersicht bewahrt wird. Außerdem ist es nicht möglich, von einer weiteren Ressource auf die Verschachtelung zu verweisen. Das Beispiel 3.3 zeigt eine fundamentale Eigenschaft von RDF in welcher Subjekte und Prädikate in mehreren Schichten auftreten. Dabei werden die Subjekte von anderen Subjekten durch Prädikate getrennt und Prädikate werden von anderen Prädikaten durch Subjekte getrennt. Niemals werden Subjekte direkt mit Subjekten oder Prädikate direkt mit anderen Prädikaten verbunden [Pow03].

Leere Knoten

Wie bereits im Abschnitt 3.1.5 besprochen, existiert die Möglichkeit, Ressourcen als leere Knoten *(bnodes)*, d.h. ohne eine URI, darzustellen. In der grafischen RDF-Modellierung geschieht dies durch eine leere Ressource. Viele RDF-Werkzeuge erzeugen in diesem Fall eine temporäre Ressource mit einer einmaligen ID, um diese von anderen leeren Knoten zu unterscheiden.

Ein leerer Knoten kann verwendet werden, wenn eine Ressource noch keinen Bezeichner hat. Oftmals auch dann, wenn es nicht sinnvoll ist, eine URI für eine Ressource zu erzeugen. Beispielsweise ist es nicht üblich, einen individuellen Menschen durch eine URI zu repräsentieren [Pow03].

Des Weiteren verwenden RDF-Werkzeuge, wie Jena (s. 5.1.1), leere Knoten für Schnitt- oder Vereinigungsmengen, die anhand von prädikatenlogischen Regeln erzeugt wurden. Die Abbildung 3.3 zeigt einen RDF-Graphen mit einem leeren Knoten.

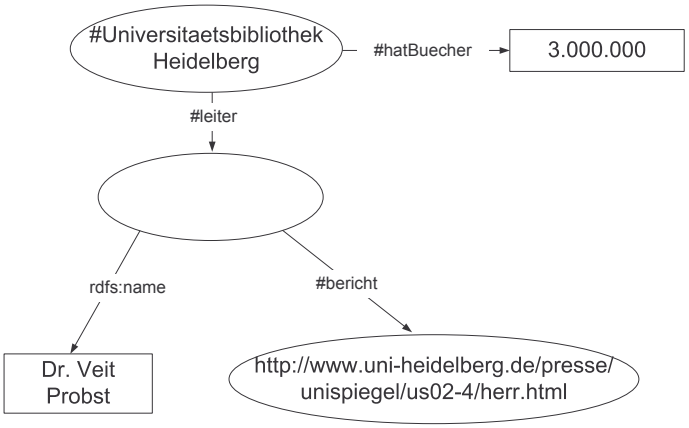


Abbildung 3.3: RDF-Statement mit leerem Knoten

In der Abbildung existiert die bereits bekannte Ressource *http://www.ub.uni-heidelberg/start.html* mit ihrem Literal. Neu in der Abbildung ist jedoch die Information, dass die Universitätsbibliothek Heidelberg einen *Leiter* hat, dessen Name *Dr. Veit Probst* – angegeben durch *rdfs:label* – lautet und der Bericht *http://www.uni-heidelberg.de/presse/unispiegel/us02-4/herr.html* von dieser Person handelt. Da es unüblich ist, Personen durch eine URI zu beschreiben, wird in diesem Beispiel ein leerer Knoten für diese Person verwendet und durch die ihr zugeordneten Objekte beschrieben. Das folgende Beispiel 3.2 zeigt die obige Abbildung 3.3 in RDF/XML-Syntax.

```
1 <rdf:Description rdf:ID="UniversitaetsbiliothekHeidelberg">
2   <ub:leiter>
3     <rdfs:label>Dr. Veit Probst</rdfs:label>
4     <ub:bericht>http://www.uni-heidelberg.de/presse/unispiegel/
          us02-4/herr.html
5     <ub:bericht>
6   </ub:leiter>
7 </rdf:Description>
```

Beispiel 3.4: RDF Statement über Statement

Genau wie das Beispiel 3.3 über die verschachtelten Ressourcen, findet auch im diesem Beispiel eine Verschachtelung statt. Der Unterschied ist der, dass innerhalb der Verschachtelung keine weitere Ressource angelegt wird, sondern direkt weitere Eigenschaftselemente nach dem Subjekt *Leiter* folgen. Aus diesem Grund wird eine leere Ressource ohne Namen erzeugt. Im Folgenden werden einige weitere Konzepte von RDF besprochen.

parseType-Attribut

Wie in den vorherigen Beispielen gezeigt, kann der Wert eines Objektes ein Literal oder eine Ressource sein. Ist das Objekt eine Ressource, wird eine Ellipse gezeichnet, ansons-

ten ist das Objekt als Literal durch ein Rechteck dargestellt. RDF-Parser wissen aus dem Kontext, um was für ein Objekt es sich handelt [Pow03].

Als Alternative gibt es die Möglichkeit, einen speziellen Typ anzugeben. Dies geschieht durch die Angabe eines *parseType*-Attributs mit dazugehörigem Wert. Das folgende Beispiel 3.6 zeigt die explizite Zuordnung des Objekts *Name* als Literal.

```
<rdf:Description rdf:ID="UniversitaetsbiliothekHeidelberg">
 <ub:hatBuecher rdf:parseType="Literal"><h1>3000000<h1></ub:hatBuecher>
 <ub:email>mailto:ub@uni-heidelberg.de</ub:email>
</rdf:Description>
```

Beispiel 3.5: RDF parseType-Beispiel

Im obigen Beispiel werden HTML-Elemente zusätzlich als Objektwerte integriert. Soll XML- oder HTML-Code innerhalb der Objektwerte gespeichert werden, muss vorher *rdf:parseType="Literal"* verwendet werden, um Fehler zu vermeiden.

Eine andere Verwendungsmöglichkeit besteht darin, eine Ressource durch *rdf:parseType="Resource"* anstelle des *ID*- oder *about*-Attributs anzulegen. Dadurch wird ein Objekt als Ressource identifiziert, obwohl keine *ID*- oder *about*-Attribut angegeben wurde. Grafisch würde dieses Objekt als leerer Knoten dargestellt werden.

RDF-Container

Häufig benötigt man eine Möglichkeit, mehrere gleichartige Beschreibungen zusammenzufassen, beispielsweise mehrere Autoren für ein Buch. Hierfür bietet RDF das Konzept der *Container* an, bei dem zwischen Sequenzen, Multimengen und Alternativen unterschieden werden kann. [EE04]. Diese werden in der folgenden Aufzählung erläutert.

- **rdf:seq**: eine *Sequenz* ist eine geordnete Liste von Elementen, wobei Duplikate innerhalb der Liste möglich sind.
- **rdf:bag**: eine *Multimenge* ist eine ungeordnete Liste von Ressourcen oder Literalen, wobei Duplikate innerhalb der Liste möglich sind.
- **rdf:alt**: eine *Alternative* lässt die Auswahl eines Elements zu.

Um eine Ressource als Container zu initialisieren, wird diesem eine *rdf:type*-Eigenschaft übergeben, deren Wert eine der vordefinierten Ressourcen *rdf:bag*, *rdf:seq*, or *rdf:alt* ist [MM04]. Elemente von Containern erhalten einen Namen der durch die folgende Syntax beschrieben wird: *rdf:_1 ... rdf:_n*. Die folgende Abbildung 3.4 zeigt ein Beispiel für einen Container vom Typ Sequenz.

Darin ist eine Ressource für ein Buch mit Namen *bspBuch* gezeigt. Für diese Ressource existiert ein Container, der eine geordnete Liste der Studenten zeigt, die dieses Buch vorbestellt haben. Normalerweise werden keine Personen, sondern Kundenkonten als Ressourcen dargestellt. Der leere Knoten benötigt keine URI, da dieser lediglich weitere

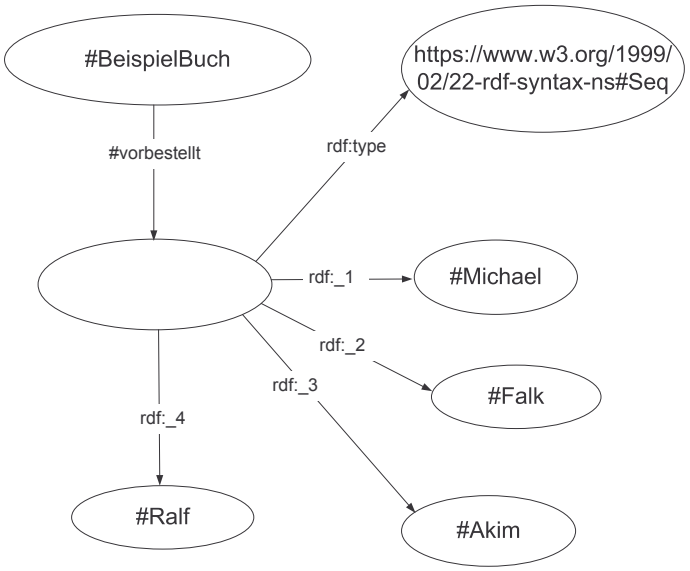


Abbildung 3.4: RDF-Container

Aussagen über andere Ressourcen enthält, selbst jedoch keine nennenswerten Informationen liefert. Theoretisch könnte dieser leere Knoten beispielsweise *Container für Buch - Student* oder ähnlich heißen.

Wie bereits erwähnt, ist der Typ des Containers aus dem RDF-Namensraum entnommen. Ebenso werden die einzelnen Aufzählungseigenschaften für die Inhalte des Containers aus dem RDF-Namensraum gewonnen. Im folgenden Beispiel 3.6 wird die Abbildung 3.4 durch RDF/XML präsentiert.

```
1  <rdf:Description rdf:ID="BeispielBuch">
2    <ub:vorbestellt>
3      <rdf:bag>
4        <rdf:li rdf:resource="#Michael"/>
5        <rdf:li rdf:resource="#Falk"/>
6        <rdf:li rdf:resource="#Akim"/>
7        <rdf:li rdf:resource="#Ralf"/>
8      </rdf:bag>
9    </ub:vorbestellt>
10 </rdf:Description>
```

Beispiel 3.6: RDF-Container

Elemente von Containern besitzen immer den Namen *li*, der sie als solche kennzeichnet. Die Containerbezeichner *rdf:_1 ... rdf:_n* werden aus den *rdf:li* Elementen generiert. Durch die Abkürzung *rdf:bag* werden zwei Elemente, nämlich ein *rdf:description*-Element und ein *rdf:type*-Element durch ein einzelnes Element ersetzt, indem eine Instanz erzeugt wird [MM04].

Es gibt analog zu den Eigenschaften (Ressource oder Literal) zwei Arten von *li*-Elementen. Die erste besitzt ein *resource*-Attribut, das mit einer URI auf eine andere bestehende bzw. definierte Ressource verweist. Die andere Art enthält entweder einen String, eine Beschreibung oder einen weiteren Container [EE04]. Ob es sinnvoll ist die

Bücher sowie die Kunden der Universitätsbibliothek direkt in den Standardnamensraum einzubinden oder dafür eigene Namensräume zu definieren ist wiederum Sache der Modellierung. In diesem Beispiel wurde der Einfachheit halber darauf verzichtet.

Manchmal gibt es eindeutige Alternativen zu RDF-Containern. Beispielsweise kann eine Beziehung zwischen einer bestimmten Ressource und einer Gruppe anderer Ressourcen auch dadurch angedeutet werden, indem die erste Ressource als Subjekt mit mehreren Statements erzeugt wird. Betrachtet man den folgenden Satz:

> Michael ist Autor von "Das Semantisch Web", "Lendplan Portierung nach Cocoon", "Diplomarbeit über BDIS"

kann dieser durch RDF/XML folgendermaßen repräsentiert werden. Der Namensraum *dc* steht für Dublin Core und wird im Abschnitt 3.1.3 näher erläutert. Zur folgenden Modellierung wird eine alternative, leicht zu lesende Methode verwendet, die in den W3C-Dokumentationen häufig benutzt wird.

```
dc:author Michael dc:published dc:title Das Semantichee Web
dc:author Michael dc:published dc:title Lendplan Portierung
dc:author Michael dc:published dc:title Diplomarbeit über BDIS
```

In diesem Beispiel sind die drei Ressourcen unabhängig voneinander, bis auf die Eigenschaft, dass diese den gleichen Autor haben. Daher ist der folgende RDF/XML Ausschnitt völlig gleichwertig zum obigen Beispiel.

```
dc:author Michael dc:published _:z
_:z       rdf:type        rdf:Bag
_:z       rdf:_1          dc:title Das Semantische Web
_:z       rdf:_2          dc:title Lendplan Portierung
_:z       rdf:_3          dc:title Diplomarbeit über BDIS
```

Andererseits besagt der Satz:

> „Der Beschluss wurde von den Mitgliedern des Komitees, bestehend aus Michael, Christine und Rosa, verabschiedet",

dass das Komitee als ganzes den Beschluss verabschiedet hat. Es besagt nicht, dass jedes Mitglied für den Beschluss gestimmt hat. In diesem Fall würde ein falsches Modell entstehen, wenn drei verschiedene Sätze, wie im Folgenden gezeigt, gebildet werden würden:

```
ub:Beschluss ub:hatVerabschiedet ub:Michael
ub:Beschluss ub:hatVerabschiedet ub:Christine
ub:Beschluss ub:hatVerabschiedet ub:Rosa
```

Aus dem oben genannten Grund wäre es besser, den Satz durch ein einzelnes *ub:hatVerabschiedet* Statement darzustellen, dessen Subjekt der Beschluss und das Objekt alle Mitglieder repräsentieren. Die Komitee Ressource kann durch einen Bag-Container wie im Folgenden dargestellt werden [MM04]:

```
ub:Beschluss   ub:hatVerabschiedet ub:Komitee
ub:Komitee     rdf:type        rdf:Bag
ub:Komitee     rdf:_1          ub:Michael
ub:Komitee     rdf:_2          ub:Christine
ub:Komitee     rdf:_3          ub:Rosa
```

Aussagen über Aussagen (Reification)

Manche RDF-Anwendungen müssen Aussagen über andere RDF-Statements machen, z.B. wann und von wem ein Statement erzeugt wurde [MM04]. Im RDF-Fachjargon spricht man dabei von:

> Reification - making Statements about Statements

Der folgende Satz: „Christine sagt, dass diese Kirschen süß sind“, ist eine solche strukturierte Aussage. Die innere Aussage besagt, *dass Kirschen süß sind*. Ob diese Ressource der Wahrheit entspricht, wird durch die äußere Aussage, *Christine sagt*, vermittelt. Das Konzept stellt sich als folgendes Modell dar [Pow03]:

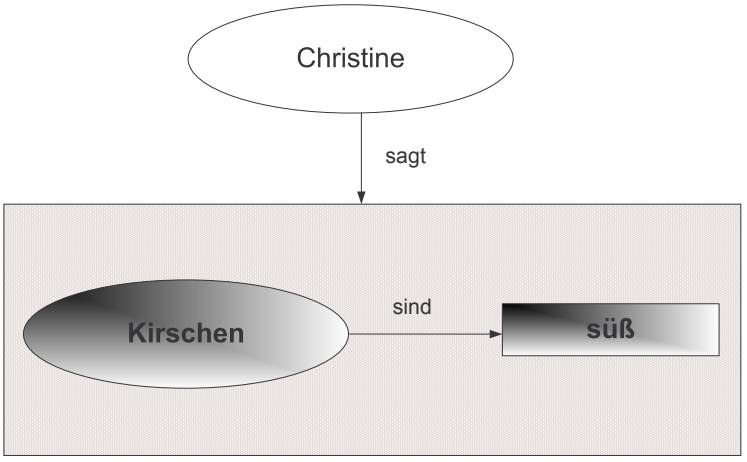


Abbildung 3.5: Statement über Statement Einführung

Zu diesem Zweck beinhaltet das RDF-Vokabular spezielle Konzepte um Aussagen über Aussagen zu treffen. Das RDF-Reification Vokabular besteht aus dem Typ *rdf:statement*, und den Eigenschaften **rdf:subject**, *rdf:predicate* und *rdf:object*.

Bei Reification handelt es sich um ein sehr komplexes Konzept, das beispielsweise in [Pow03] als “the big ugly“ bezeichnet wird. Viele Werkzeuge unterstützen das Konzept nicht. Aus diesen Gründen wird in der Masterthesis nicht näher auf diese Möglichkeit eingegangen.

3.2 Resource Description Framework Schema

Mit dem Resource Description Framework-Modell erhält man die Möglichkeit, einzelne XML-konforme Dokumente zu erzeugen, welche Objekte anhand von Statements beschreiben. Durch die geschickte Wahl der Ressource-Namen (s. 3.1.2) erhält man Informationen über das jeweilige Objekt. Um eine Gruppe von ähnlichen Objekten, z.B. Bücher, alle mit den gleichen Eigenschaften auszuzeichnen, bietet RDF keine Möglichkeit um einen “Rahmen“ für alle diese Objekte zu definieren. Für diese Zwecke wurde die

RDF Beschreibungssprache – RDF Schema (RDFS) definiert [Bri04]. Diese stellt die Möglichkeit bereit, Begriffe und die damit verbundenen Elemente semantisch zueinander in Beziehung zu setzten, z.B. kann mit RDFS festgelegt werden, dass die Eigenschaft *title* dazu verwendet wird, um den Titel eines Buchs zu beschreiben [EE04]. In RDF-Schema wird für jede Eigenschaft festgelegt, welche Werte erlaubt sind, was diese für eine Bedeutung hat, welche Beziehungen zu anderen Eigenschaften bestehen und welche Arten von Ressourcen diese Eigenschaft verwenden darf. Dabei wurde vom W3C nicht ein allgemein gültiges Schema definiert, in dem verschiedene Klassen und Eigenschaften festgelegt werden, sondern es wird in einer "Schema Definition Language" beschrieben, mit deren Hilfe die eigentlichen Schemas definiert werden können. Diese Schemas werden auch als Vokabulare bezeichnet [KK04].

In den letzten Jahren haben sich RDF-Schema-Gemeinschaften gebildet, die die Aufgabe haben, RDF-Schema-Metadatenmodelle zu entwerfen, z.B. der Dublin Core (s. 3.1.3). Durch diese dezentralisierte Vorgehensweise wird eingestanden, dass es unmöglich ist, ein einzelnes Schema zu entwickeln, das für alle Gebrauchsmöglichkeiten passend wäre.

3.2.1 Aufbau von RDF-Schema

Das Vokabular, welches von RDFS zur Verfügung gestellt wird, beschränkt sich auf einige Dutzend Wörter. Mit Hilfe dieses Vokabulars können jedoch eigene Konzepte und deren Eigenschaften spezifiziert und eingeschränkt werden. Die wesentlichen Konzepte von RDFS sind Klasse- und Unterklasse- sowie Eigenschafts- Untereigenschafts-Beziehungen. Klassen fassen Objekte mit gleichen Eigenschaften zusammen, z.B. die Klasse *Buch*. Unterklassen sind spezielle Untermengen der Oberklasse, denen zusätzliche Eigenschaften hinzugefügt wurden, z.B. die Klasse *technischesBuch*. Entsprechendes gilt für Eigenschafts- Untereigenschafts-Beziehungen [EE04].

RDF-Schema-Dokumente müssen der RDF/XML-Syntax genügen und sind auf den ersten Blick nicht von RDF-Dokumenten zu unterscheiden. Charakteristisch für RDF-Schema-Dokumente ist die häufige Verwendung der Vokabulare *rdfs:Class* und *rdfs:Property* sowie der Eigenschaften *rdfs:subClassOf* und *rdfs:subPropertyOf*. Ein einführendes RDF-Schema wird im Beispiel 3.7 gezeigt:

```
1  <rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
2         xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
3         xmlns:ub='http://www.ub.uni-heidelberg.de/ns#'>
4
5    <rdfs:Class rdf:ID="Medium">
6        <rdfs:label xml:lang="de">Medium</rdfs:label>
7        <rdfs:comment>Ein Medium, dass in beliebiger Form vorliegt</
             rdfs:comment>
8    </rdfs:Class>
9
10   <rdfs:Class rdf:ID="Zeitschrift">
11       <rdfs:label xml:lang="de">Zeitschrift</rdfs:label>
12       <rdfs:comment>Regelmäßig erscheinendes Heft.</rdfs:comment>
13       <rdfs:subClassOf rdf:about="#Medium"/>
```

```
14    </rdfs:Class>
15
16    <rdfs:Class rdf:ID="Standort"/>
17
18    <rdf:Property rdf:ID="hatStandort">
19          <rdfs:domain rdf:resource="#Medium"/>
20          <rdfs:range rdf:resource="#Standort"/>
21    </rdf:Property>
22  </rdf:RDF>
```

Beispiel 3.7: RDFS-Klassen

Das RDFS-Beispiel hat drei Namensraum-Deklarationen für die bereits vorgestellten Namensräume von RDF, RDF-Schema und dem selbst definierten Namensraum der Universitätsbibliothek Heidelberg. In den folgenden Zeilen werden zwei Klassen definiert, wobei *ub:Medium* die Oberklasse darstellt und die Klasse *Zeitschrift* davon erbt. Innerhalb der drei Klassen werden durch so genannte *Annotation Properties* (s. 3.3.1) Informationen hinterlegt, die i.d.R. nicht von Maschinen verarbeitet werden und sich direkt auf die Konzepte beziehen. Diese haben die Aufgabe, die Klassen direkt zu beschreiben, so dass ein Anwender weiß, wozu dieses Konzept dient. Durch die dritte Klasse werden Instanzen vom Typ *Standort* erzeugt.

Zuletzt wurde eine Eigenschaft *hatStandort* definiert, welche innerhalb der Instanzen von Medium verwendet werden kann. Eigenschaften sind der Teil eines RDF-Statements, welcher eine Ressource mit einem Objekt verbindet. In RDFS können Eigenschaften mit Einschränkungen spezifiziert werden. Durch *rdfs:domain* wird angegeben, welche Ressource die Eigenschaft verwenden darf und mit *rdfs:range* wird festgelegt, auf welchen Objekttyp die Eigenschaft zeigen kann. Die folgende Abbildung 3.6 visualisiert diese Möglichkeiten:

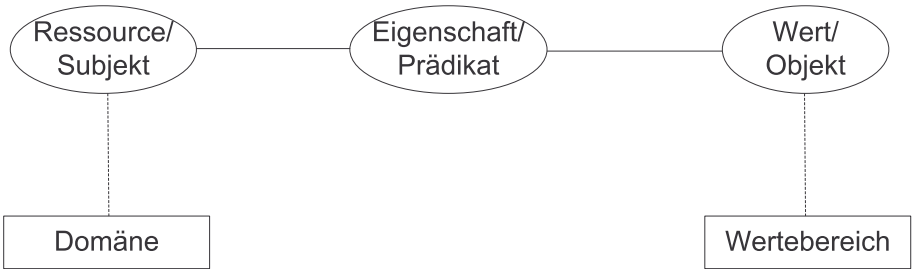


Abbildung 3.6: Domäne und Wertebereich [EE04]

Innerhalb von RDF könnten diese definierten Konzepte, wie im Beispiel 3.8 gezeigt, verwendet werden. Die obere Instanz gibt direkt den Typ der Ressource durch den Zeitschriften-Tag <*zeitschrift*> an. Die einzelnen Eigenschaften werden mit Hilfe des Dublin Cores beschrieben. Die zweite Ressource verwendet das im Abschnitt 3.1.6 eingesetzte *rdf:description*-Attribut. Durch *rdf:type* wird beschrieben, dass es sich um eine Ressource vom Typ *Zeitschrift* handelt. Beide Methoden sind gleichwertig.

In der Praxis wird wegen der kürzeren Schreibweise Variante 1 verwendet. Die Eigenschaft *hatStandort* zeigt auf jeweils eine Ressource vom Typ Standort. Durch das

rdf:resorce-Attribut wird angegeben, dass die Eigenschaft keine Instanz definiert, sondern direkt auf eine bereits definierte Ressource zeigt. Die beiden Ressourcen vom Typ *Standort* werden zuletzt deklariert.

```
<Zeitschrift rdf:ID="zeitschrift1">
   <dc:title>Java Magazin</dc:title>
   <dc:format>Online-Ressource</dc:format>
   <dc:identifier>http://www.ub.uni-heidelberg.de/dok/12306
   </dc:identifier>
   <ub:hatStandort rdf:resource="#Lesesaal"/>
</Zeitschrift>

<rdf:Description rdf:ID="zeitschrift2">
  <rdf:type rdf:resource="#Zeitschrift"/>
  <dc:title>C++ Magazin</dc:title>
  <dc:format>Online-Ressource</dc:format>
  <dc:identifier>http://www.ub.uni-heidelberg.de/dok/12307
  </dc:identifier>
  <ub:hatStandort rdf:resource="#WEB"/>
</rdf:Description>

<Standort rdf:ID="Lesesaal"/>
<Standort rdf:ID="WEB"/>
```

Beispiel 3.8: RDF-Einsatz

Das Vokabular von RDFS ist auf die grundsätzlichen Definitionen von Klassen und Eigenschaften beschränkt. Stärkere Einschränkungen, durch die zusätzliches Wissen gewonnen werden kann, wird erst durch die auf RDFS aufbauende Ontologiesprache OWL(s. unten) ermöglicht. Im Folgenden wird RDFS mit ähnlichen Konzepten verglichen.

3.2.2 RDFS vs. XML-Schema

Das RDF-Schema lässt sich nur bedingt mit dem XML-Schema vergleichen. Das XML-Schema gibt die syntaktische Struktur von wohlgeformten XML-Instanzen vor. Hingegen wird ein RDF-Dokument, welches auf einem RDF-Schema aufbaut, einerseits durch die von der RDF/XML-Syntax vorgegebene Struktur eingeschränkt, andererseits ermöglicht RDF-Schema die semantische Einordnung von Begriffen, die innerhalb der Beschreibungen verwendet werden soll. Insbesondere der zweite Aspekt sowie die semantischen Beschreibungen gehen über das XML-Schema hinaus [EE04].

3.2.3 RDFS vs. Objektorientierung

Das RDF-Schema ähnelt dem Konzept zur objektorientierten Programmierung. Jedoch gibt es vier gravierende Unterschiede, die im folgenden beschrieben werden [Wle03]:

1. **Mehrfachklassifikation**, d.h. die Zuordnung einer Klasse zu mehreren anderen ist in RDF-Schema erlaubt. In der Welt der Objektorientierung wird dieses Konzept

lediglich von C++ unterstützt. Eine Gemeinsamkeit ist jedoch, dass keine Ressource völlig klassenlos sein kann - selbst eine nicht explizit klassifizierte Ressource gehört immer der Klasse *rdfs:Ressource* an.

2. Unter einem **erweiterbaren Metaschema** versteht man ein **Metaschema** mit all jenen Mitteln, die zur Verfügung stehen, um das Schema zu definieren. In der Objektorientierung besteht das Metaschema zumeist aus speziellen Schlüsselwörtern, z.B. class, final oder public, und speziellen syntaktischen Konstrukten, z.B. durch Klammerung {}. Während diese Metaschemas in der Objektorientierung statisch sind, können innerhalb von RDFS neue Metakonstrukte definiert und eingesetzt werden.

3. Hinter dem Begriff **autarke Prädikate** verbirgt sich die Idee, Eigenschaften von Klassen unabhängig von der entsprechenden Klasse zu definieren. Bei Definition des Prädikates kann optional bestimmt werden, auf welche Klassen dieses Prädikat angewendet werden darf (Definitionsbereich), und welche Klassen als Wert verwendet werden dürfen (Wertebereich).

4. **Mehrfache Prädikate**: Es ist möglich, in RDF-Schema **Prädikate** die einer Klasse zugeordnet werden, mehrfach zu verwenden. In der Objektorientierung darf ein Prädikat nur einen Wert erhalten, mit Ausnahme von dynamischen Datentypen oder Arrays.

3.3 Web Ontologie Sprache (OWL)

Die Web Ontology Language (OWL) ist die zurzeit populärste Sprache für die Modellierung von Ontologien und damit zur Entwicklung des Semantischen Webs. OWL wurde vom der W3C Gruppe *Web Ontology Working Group* entwickelt. Eine erste Version erschien im Jahre 2003 [DOS03]. OWL ist von der DAML+OIL Web Ontologie Sprache abgeleitet und baut auf RDF/RDFS auf. Das bedeutet, dass die offizielle Austausch-Syntax RDF ist [Kos].

OWL wird auf dem Semantischen Web Konzept (s. Abbildung 2.4) oberhalb von XML angesiedelt. Mit OWL werden, genau wie mit RDFS, Terme einer Domäne und deren Beziehungen formal beschrieben. Allerdings bietet OWL im Vergleich zu RDFS weitaus komplexere Funktionen zum Beschreiben der Beziehungen. Allgemein liegt der Unterschied zwischen OWL und RDFS darin, dass sich in OWL Konzepte deutlicher spezifizieren lassen, wodurch ein höherer Abstraktionsgrad entsteht. Des Weiteren können mit Hilfe von Reasonern, welche OWL anstelle von RDFS verarbeiten, bessere logische Schlussfolgerungen geschlossen werden, da sich in OWL logische Konstrukte erstellen lassen, die mit RDFS nicht möglich sind.

Die Web Ontology Language existiert in drei verschiedenen Versionen. Dazu wurden die Sprachebenen *OWL LITE*, *OWL DL* und *OWL FULL* definiert. Für den Einsatz von OWL LITE/DL wurden Einschränkungen definiert, welche die Entwicklung von Werkzeugen erleichtern. Das Ziehen von logischen Schlussfolgerungen basiert in OWL FULL auf dem

Konzept des so genannten *Open World Assumption* – kurz OWA. Das Open World Assumption bedeutet, dass ein Reasoner nicht annimmt, dass etwas nicht existiert, solange nicht explizit definiert wurde, dass es nicht existiert. Allgemein ausgedrückt gilt, dass solange etwas nicht als Wahr ausgesagt wurde, ein Reasoner nicht annimmt, dass es Falsch ist – es wird lediglich angenommen, dass das Wissen noch nicht zur Wissensbasis hinzugefügt wurde [HKR+04]. Dadurch kann es in OWL FULL vorkommen, dass keine Rückgabemenge gefunden wird. Dabei besteht die Gefahr, eine unendlich oder zumindest sehr lange dauernde Rechenoperation anzustoßen.

Im Folgenden werden die einzelnen Sprachebenen beschrieben:

1. **OWL LITE** unterstützt die Funktion zum Aufbau von Klassenhierachien oder Taxonomien und einfachen Beziehungen [MH04].

2. **OWL DL** – oder OWL Description Logic – ist für Anwender vorgesehen, die ein Maximum an Ausdrücken formulieren wollen, die alle in einer endlichen Zeit abgearbeitet werden können. Im Vergleich zu OWL FULL existiert bei OWL DL die Garantie, dass jeder formulierte Ausdruck auch ein Ergebnis zurückliefert. Um dies zu gewährleisten, wurden diverse Einschränkungen eingefügt.

3. **OWL FULL** besteht aus den selben Sprachkonstrukten wie OWL DL, verzichtet aber auf die dort vorhandenen Einschränkungen. Dadurch sind die Ontologien unentscheidbar (s. 2.2), können dafür aber prädikatenlogische Ausdrücke höheren Grades ermöglichen [Wik05b].

Jede dieser Sprachen ist eine Erweiterung der jeweiligen vorherigen. Daher ist jede OWL LITE Ontologie auch eine korrekte OWL DL Ontologie. Mathematisch kann das Ganze in der folgenden Formeln dargestellt werden [SH04]:

$$RDF(S) \subset OWLLITE \subset OWLDL \subset OWLFULL \tag{3.1}$$

3.3.1 OWL im Detail

Im Folgenden werden die grundsätzlichen funktionalen Erweiterungsmöglicheiten von OWL beschrieben. OWL verwendet einige Sprachkonzepte aus dem RDF-Namensraum. Lediglich Ausdrücke, die mit RDF nicht möglich sind, werden durch den OWL-Namensraum abgedeckt.

Klassen

Eine OWL-Klasse, welche durch *owl:class* erzeugt wird, hat die gleiche Funktionalität wie eine mit *rdfs:class* erzeugte Klasse. Allerdings bietet der OWL-Namensraum die bereits erwähnte und im Folgenden erklärte Funktionalitätserweiterung. Jede Klasse ist in RDF auch eine Ressource.

Instanzen

In OWL können genau wie in RDFS Instanzen von definierten Konzepten erzeugt werden. Während OWL FULL keine Einschränkungen besitzt, können bei OWL DL und OWL LITE Eigenschaften keine Klassen zugewiesen bekommen. Beispielsweise kann in OWL-DL nicht ausgedrückt werden, dass die Instanz der Klasse Buch “Das Java Buch“ für die Eigenschaft “hatFach“ direkt die Klasse “Java“ zugewiesen bekommt. Stattdessen muss zuerst eine Instanz, der Klasse “Java“ erzeugt werden. Dies hat den Nachteil, dass bei der obigen Fächerhierarchie auf eine Konsistenz zwischen Instanz und Klasse geachtet werden muss [Tea05f].

Instanzen werden genau dann verwendet, wenn spezifische und individuelle Aussagen gemacht werden müssen. Jede Instanz ist in RDF auch eine Ressource.

Eigenschaften

OWL unterscheidet zwischen zwei Eigenschaftstypen, die im Folgenden erklärt werden. Diese Unterscheidung existiert bei RDFS nicht [Tea04c].

1. **owl:ObjectProperty**: Objekt-Eigenschaften verbinden je eine Instanz mit einer anderen Instanz, beispielsweise:

   ```
   (Instanz(Mannheim) -- liegtNaheBei -- Instanz(Heidelberg))
   ```

2. **owl:DatatypeProperty**: Datentyp-Eigenschaften verbinden je eine Instanz mit einem Datenwert, beispielsweise:

   ```
   (Instanz(Bibliothek Heidelberg) -- hatBücher -- [3000000]).
   ```

Des Weiteren kann jede Eigenschaft transitiv **owl:TransitiveProperty**, funktional (nur eine Beziehung) **owl:functionalProperty** oder die Inverse **owl:inverseProperty** einer anderen Eigenschaft sein.

Annotation Properties

Zusätzliche Informationen über eine Ressource können mit so genannten *Annotation Properties* spezifiziert werden. Häufig wird mit Annotation Properties auf normale Texte verwiesen, die nicht von Maschinen interpretiert werden können [EE04].

Des Weiteren dienen Annotation Properties dazu, direkt Klassen mit zusätzlichem Wissen auszustatten. Normale Eigenschaften werden in OWL lediglich Klassen zugewiesen, können aber nur auf Instanzen dieser Klassen verwendet werden, um Wissen zu hinterlegen. Dagegen sind Annotation Properties in der Lage, direkt die Klassen mit Wissen auszustatten. Annotation Properties können genau wie normale Eigenschaften vom Typ Objekt- oder Datentyp-Eigenschaft sein.

Während OWL FULL keine Einschränkungen für Annotation Properties festlegt, erlaubt OWL DL deren Verwendung auf Klassen, Eigenschaften und Instanzen nur unter bestimmten Bedingungen, z.B. muss das Objekt der Annotation entweder vom Typ Literal, URI oder eine Instanz sein [Bec04]. Es existieren fünf vordefinierte Annotation Properties: *owl:versionInfo*, *rdfs:label*, *rdfs:comment*, *rdfs:seeAlso* und *rdfs:isDefinedBy* [Bec04]. Auch die Eigenschaften des Dublin Core können als Annotation Properties verwendet werden [HKR+04].

Kardinalität

Zusätzlich zu den beschriebenen Eigenschaftsmerkmalen ist es möglich, weitere Beschränkungen des Wertebereichs von Eigenschaften, in speziellen Kontexten vorzunehmen. Die Attribute *owl:allValuesFrom* und *owl:some-ValuesFrom*, gelten lokal in ihren umschließenden Klassendefinitionen [Kos]. Wird die *owl:allValuesFrom*-Einschränkung einer bestimmten Eigenschaft zugewiesen, müssen alle Instanzen, die diese Eigenschaft benutzen, ausschließlich vom angegebenen Typ sein. Beispielsweise kann eine Instanz *Bordeaux-Wein* von der Klasse *Wein* abstammen. Innerhalb der Klasse *Wein* existiert eine Eigenschaft *hatHersteller*. Diese Eigenschaft hat die oben beschriebene Einschränkung *owl:allValuesFrom #WeinHersteller*. Durch diese Einschränkung folgt, dass die Instanz *Bordeaux*, und deren *hatHersteller*-Eigenschaft, nur auf Instanzen der Klasse *WeinHersteller* zeigen darf.

Das Attribut *owl:someValuesFrom* ist ähnlich, aber weniger restriktiv. Dies bedeutet, dass mindestens eine der Individuen vom entsprechenden Typ sein muss. [Kos].

Die strengste Kardinalitätsbeschränkung ist *owl:cardinality*, was die Festlegung der exakten Anzahl von Elementen in einer Relation erlaubt [Kos].

Weitere Eigenschaften

In OWL-Klassen kann Äquivalenz mit der Eigenschaft *owl:equivalentClass* bzw. *owl:samesAs* ausgedrückt werden. Des Weiteren kann mit *owl:disjointWith* der Gegensatz zu einer Klasse bzw. zu einer Menge von Klassen spezifiziert werden. Boolsche Kombinationen können mit den Eigenschaften *owl:unionOf* für die Vereinigung, *owl:intersectionOf* für den Durchschnitt und *owl:complementOf* für die Negation verwendet werden, um eine neue Klasse zu bilden [EE04].

Das Attribut *owl:hasValue* ermöglicht es, Klassen basierend auf der Existenz einzelner Eigenschaftswerte zu definieren. Somit ist ein Individuum ein Mitglied einer solchen Klasse, wenn mindestens eines seiner Eigenschaftswerte gleich der *owl:hasValue* Ressource ist [Kos].

Abbildung von Ontologien

Damit Ontologien ihre größte Wirkung entfalten können, müssen sie weit verbreitet werden. Um den Aufwand im Entwickeln von Ontologien zu minimieren, werden diese wie-

derverwendet. Die größte Arbeit beim Entwerfen von Ontologien besteht darin, Klassen und Eigenschaften in der Art zusammenschließen, dass sie die Möglichkeiten zum Schlussfolgern maximieren.

Will jemand eine Menge von Ontologien als Komponente einer eigenen Ontologie aneinander binden, ist es meist nützlich, ausdrücken zu können, dass einzelne Klassen oder Eigenschaften der einen Ontologie äquivalent zu Klassen und Eigenschaften einer anderen Ontologie sind [Kos]. Dies kann beispielsweise bei der Verwendung von Ontologien, die in anderen Sprachen definiert wurden, sehr nützlich sein.

Zum Umsetzen dieser Konzepte stehen in OWL die Attribute *owl:sameClassAs*, *owl:samePropertyAs* und *owl:sameIndividualAs* zur Verfügung.

Kapitel 4

Analyse des Semantischen Webs für die Universitätsbibliothek Heidelberg

Die Kapitel 1 – 3 beschreiben die komplexen Grundlagen zum Aufbau von Semantischen Netzwerken. Das folgende Kapitel analysiert die Anforderungen und Möglichkeiten des Vorhabens, ein Semantisches Web für die Universitätsbibliothek Heidelberg zu realisieren. Im Folgenden wird dieses Projekt als **SemantikUB** bezeichnet werden.

4.1 Konzept von SemantikUB

Die Aufgabe von SemantikUB im bibliothekarischen Umfeld besteht darin, eine bessere Suche nach Medien aller Art, für den Benutzer anzubieten. Darüber hinaus kann SemantikUB auch als eine Art Berater eingesetzt werden, der dem Benutzer bei den komplexen Suchmöglichkeiten, die eine Universitätsbibliothek i.d.R. anbietet, unterstützt. Zuletzt muss es auf der administrativen Seite für die Mitarbeiter der Universitätsbibliothek möglich sein, SemantikUB möglichst automatisiert zu pflegen und darüber hinaus benutzerfreundliche Verwaltungsmasken anzubieten. Im Folgenden werden diese Konzepte näher untersucht und erläutert.

4.1.1 Verbesserte Suche über Fächer

Der folgenden Abschnitt zeigt das Funktionsprinzip von SemantikUB für eine einfache Suche über die Fächerstruktur. Anhand des Sequenzdiagramms 4.1 wird diese Einsatzmöglichkeit grafisch visualisiert und erläutert.

Das Sequenzdiagramm zeigt einen Anwender. Dieser ruft über seinen Browser die SemantikUB Webseite auf, welche verschiedene Suchmöglichkeiten anbietet. Der Benutzer führt eine fachbezogene Suche nach Medien zum Thema Java durch. Die erhaltenen Treffer sind für den Benutzer nicht zufriedenstellend. Aus diesem Grund führt er eine weitere Suche durch, in welcher er sich alle Medien anzeigen lässt, welche innerhalb der Fächerstruktur dem übergeordneten Fach zugeordnet sind –in diesem Fall Objektorientierung.

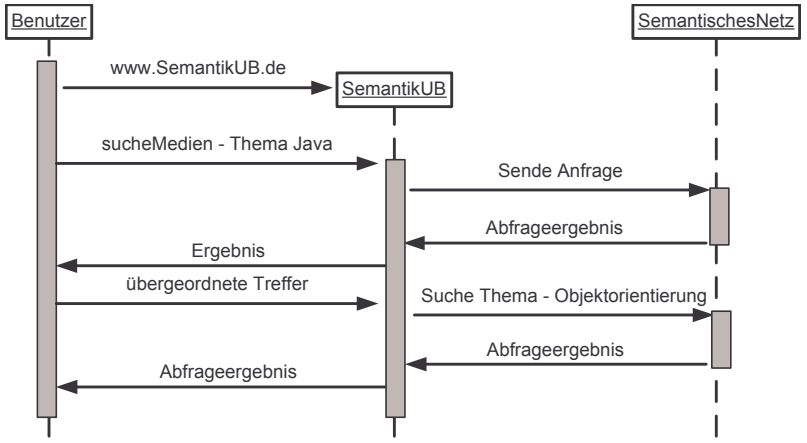


Abbildung 4.1: Suche nach Thema Java und übergeordnete Treffer

4.1.2 Berater

Eine Universitätsbibliothek bietet i.d.R. viele verschiedene Möglichkeiten zum Suchen und Finden von Informationen an. Beispielsweise ist es möglich, auf mehrere Dutzend Kataloge zuzugreifen, um nach Medien zu suchen. Es existieren mehrere hundert Datenbanken, die wiederum unterteilt werden müssen in so genannte Volltextdatenbanken und in Verzeichnisse. Volltextdatenbanken sind Sammlungen von Texten, beispielsweise Lexikas. Verzeichnisse hingegen verfügen über keine Texte im eigentlichen Sinn, sondern stellen eine Art Sammlung von Informationen dar. Die bekannteste und am weitesten verbreitetste Suche ist das Recherchieren im Hauptkatalog, in welchem alle Medien verzeichnet sind. Oftmals ist diese Suche jedoch nicht ausreichend, da im Hauptkatalog beispielsweise keine Zeitschriftenartikel, sondern lediglich die Zeitschrift selbst erfasst wird. Gleiches gilt für die Inhalte in so genannten Volltextdatenbanken.

Viele Benutzer kennen die erweiterten Suchmöglichkeiten nicht. Einer der Gründe besteht darin, dass das Auffinden sowie die Bedienung des Kataloges einen zeitlichen Mehraufwand darstellt. Des Weiteren sind oftmals bibliothekarische Grundkenntnisse zum Arbeiten mit den Katalogen erforderlich. Um beispielsweise nach Artikeln, Büchern, Zeitschriften, Publikationen oder ähnlichem zu suchen, existieren so genannte Bibliographiedatenbanken. Diese liefern i.d.R. sehr ausführliche Listen zu allen Publikationen eines bestimmten Themas. Beim Anwenden solcher Bibliographiedatenbanken stellen sich für unerfahrene Benutzter jedoch folgende Probleme:

- Was liefert eine Bibliographiedatenbank?
- Was kann mit den Treffern gemacht werden?
- Wo ist das eigentliche Dokument hinterlegt?

Aus diesem Grund ist es nicht nur die Aufgabe von SemantikUB bessere Ergebnisse innerhalb der Suche zu erzielen, sondern dem Benutzer beim Finden und Verwenden der

entsprechenden Suchmaschine zu unterstützen. Dabei könnte es sich um eine anwendergeführte Suche handeln, die ähnlich einem ELearing-System den Benutzer durch die verschiedenen Bereiche führt. Hat ein Benutzer beispielsweise im Hauptkatalog keine oder nur unzureichende Treffer gefunden, wäre es hilfreich, wenn SemantikUB Vorschläge für weitere Suchmöglichkeiten aufzeigen könnte.

Das folgende Sequenzdiagramm 4.2 zeigt ein solches Szenario auf. Ein Anwender betritt die Startseite von SemantikUB. Auf dieser wird ihm beispielsweise erklärt, welche Möglichkeiten existieren, einerseits kann er im Hauptkatalog direkt nach vorhandenen Medien zu einem bestimmten Thema suchen oder falls er dort nicht erfolgreich war, in den anderen Datenbanken. SemantikUB kann Vorschläge mit Erklärungen für weitere Datenbanken machen. Die Informationen über weitere Datenbanken sind alle im Semantischen Netz hinterlegt. Eine noch weit in der Zukunft liegende Vision könnte es sein, dass SemantikUB bereits anhand der vorgenommenen Suche, die passenden Datenbanken, z.B. medizinisch- oder technisch-orientierte Datenbanken auswählt und dem Benutzer präsentiert. Im Sequenzdiagramm 4.2 erfolgt eine Suche in einer Bibliographiedatenbank. In diesem zeigt SemantikUB auf, um was es sich bei den gefundenen Treffern handelt und macht weiterhin Vorschläge, was mit diesen Resultaten zu tun ist, z.B. eine Suche im Hauptkatalog der Bibliothek oder in den Verbundkatalogen (s. [Lei05]) mit darauf folgender Fernleihe.

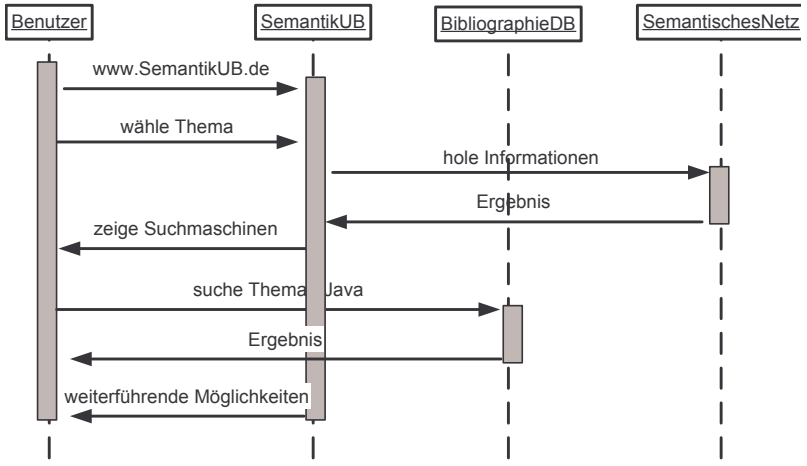


Abbildung 4.2: Unterstütze suche in Bibliographiedatenbank

Im Vergleich zu den im Abschnitt 1.2 vorgestellten virtuellen Agenten, ist die benutzergeführte Suche nicht in der Lage, eine gegenseitige Kommunikation aufzubauen. Lediglich berät das System den Benutzer und zeigt ihm bestimmte Möglichkeiten auf.

4.1.3 Web-Service

Das in Abbildung 4.2 gezeigte Szenario könnte noch weitergeführt werden. Darin werden die von der Bibliographiedatenbank erhaltenen Treffer, selbst auf Vorhandensein in verschiedenen Katalogen nachgeprüft. Als Ergebnis wird eine Bestandsliste zurückgeliefert.

Da es sich hierbei um eine Interaktion zwischen zwei Maschinen handelt, spricht man von Web-Services. Ein Semantischer Web-Service könnte dabei bereits erkennen, in welchen Katalogen es zu suchen gilt. Ist beispielsweise ein Treffer der Bibliographiedatenbank ein Artikel aus einer Zeitschrift, muss der Web-Service Zeitschriftendatenbanken abfragen. Wie bereits im Abschnitt 1.2 erwähnt, bilden Semantische Web-Services ein großes Potential. Jedoch können diese erst eingesetzt werden, wenn alle Kataloge in denen gesucht wird, mit Ontologien ausgestattet sind [BFS05].

4.1.4 Schlagwortkataloge

Ein letztes Szenario dient zur verbesserten Pflege der Treffer und damit der Administration der Daten, wodurch ebenfalls bessere Suchresultate gewährleistet werden können. In Bibliothekskatalogen wird mit so genannten *Schlagwörtern* gearbeitet. Diese ordnen beispielsweise eine Zeitschrift einem oder mehreren Fächern zu. Dadurch kann sich der Benutzer alle Treffer anzeigen lassen, die ebenfalls den entsprechenden Bereichseintrag enthalten. Das ganze funktioniert ähnlich, wie die im ersten Szenario vorgestellte Suche in einem übergeordneten Fach. Der wichtige Unterschied ist jedoch, dass die Anordnung der Schlagwörter keine Hierarchie aufweist. Es existiert lediglich eine sehr große Liste mit Schlagwörtern, mit der man Medien auszeichnen kann. Sucht man nach einem Schlagwort erhält man alle Medien, die das entsprechende Schlagwort enthalten. Oftmals enthalten Medien mehrere Schlagwörter, wodurch es möglich ist, auch in andere Bereiche vorzustoßen.

Was passiert jedoch, wenn ein Medium bestimmte Gebiete behandelt, für die eine spezielle Bezeichnung erst einige Jahre nach der Katalogisierung des entsprechenden Medium erscheint? Die folgende Abbildung 4.3 zeig ein solches Szenario auf, in welchem ein Buch bei der Katalogisierung mit den Schlagworten Mechanik, Elektronik und Informationstechnik ausgezeichnet wird.

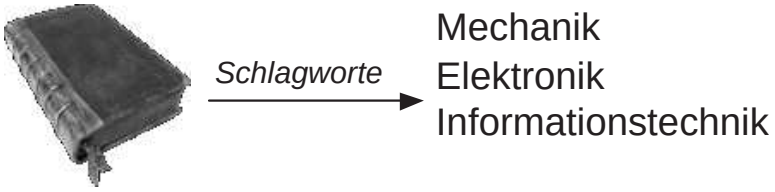


Abbildung 4.3: Schlagwort Szenario

Aus diesen drei Schlagworten ist in erst seit den 90’er Jahren das Gebiet der Mechatronik hervorgegangen. Das Buch in der Abbildung muss, falls dieses auf den aktuellen Stand gebracht werden soll, mit einem weiteren Schlagwort ausgestattet werden. Möglicherweise kann dies durch maschinelle Bearbeitung, beispielsweise durch einen Datenbank-Join, vereinfacht werden.

Im Semantischen Netz hingegen könnte dieses neue Fach definiert werden, indem es die Bedingung hat, dass es mindestens die obigen drei Fächer hat (s. owl:hasValue 3.3.1). Der Logikteil vom SemantikUB wird durch dieses neu definiert Konzept, automatisch

jedes Medium, dass die drei Fächer Mechanik, Elektronik und Informationstechnik enthält, zusätzlich dem Bereich der Mechatronik zuordnen. Auf diese Weise kann innerhalb der Wissensbasis neues Wissen hinzugefügt werden, ohne die eigentliche Metadaten der Medien zu ändern.

4.1.5 Visualisierung

Für die Visualisierung von SemantikUB sind neue Konzepte nötig, da der Benutzer häufig mit bidirektionalen Verbindungen arbeitet. Auch die häufig auftretenden Querverbindungen im Semantischen Netz sind mit normalen unidirektionalen Links nur schwer zu realisieren. Das System muss dem Benutzer intuitiv anzeigen, woher er gekommen ist und wie er dorthin zurückkommen kann. Das Layout und die Anzeige der Daten muss entsprechend intuitiv erfolgen.

4.2 Anforderungen an das Projekt SemantikUB

Im Folgenden werden die grundsätzlichen Punkte aufgezählt, die aktuell eingesetzte Systeme anbieten sowie Anwenderwünsche, die sich mit Semantischen Netzen realisieren lassen. Weiterhin werden Ideen für zukünftige Anforderungen aufgelistet, die bereits bei der Modellierung der Ontologie beachtet werden können.

1. Das System muss als Berater für die verschiedenen Suchmöglichkeiten eingesetzt werden können. Dabei hat es die Aufgabe:
   - passende Suchmöglichkeiten vorzuschlagen,
   - dem Benutzer zu zeigen, wie er die Suchergebnisse verwenden kann.
2. Altdaten müssen automatisiert konvertiert werden können.
3. Neue Daten müssen automatisiert eingetragen werden können.
4. Die Eingabe und Pflege der Daten muss über eine benutzerfreundliche Maske erfolgen.
5. Die Ontologie muss Standards, wie beispielsweise den Dublin Core verwenden.
6. Die Ontologie muss für andere Systeme offen und erweiterbar sein.
7. Zukünftige Konzepte, müssen beim Aufbau der Ontologie beachtet werden.
8. SemantikUB muss über das Internet bedienbar sein.

4.3 Lastenheft

Der folgenden Abschnitt beschreibt die Funktionsanforderungen, die an SemantikUB gestellt werden. Diese wurden im Verlauf der Masterthesis durch die Mitarbeiter der verschiedenen Abteilungen der Universitätsbibliothek Heidelberg gesammelt und zusammengestellt. Das Lastenheft zeigt die zurzeit realisierbaren Punkte. Andere Ideen, welche im vorherigen Abschnitt 4.1 vorgestellt wurden, können nicht in das Lastenheft für diese Masterthesis aufgenommen werden.

4.3.1 Entwurf einer Ontologie

Für SemantikUB wird eine Ontologie benötigt. Aus diesem Grund muss untersucht werden, ob bereits vorhandene Ontologien das zu modellierende Spektrum abdecken kann. Andernfalls erfolgt ein Neuentwurf nach dem in Abschnitt 2.4.3 vorgestellten Schema. Bei der Modellierung ist auf zukünftige Möglichkeiten zur Erweiterbarkeit der Ontologie zu achten. Die Ontologie muss sich an die, durch das W3C, gegebenen Standards halten.

4.3.2 Import von Altdaten

Die bereits vorhandenen Metadaten über elektronische Medien müssen importiert werden können.

4.3.3 Implementierung der Business-Logik

Für SemantikUB wird eine Business-Logik benötigt, welche das Kernstück der Anwendung bildet. Diese hat die folgenden Aufgaben:

- Verarbeiten der Wissensbasis,
- persistente Speicherung der Daten,
- Anbieten von Funktionen zur Suche, Beratung und Pflege der Daten,
- Modellierung und Rückgabe der Daten aus der Wissensbasis, in benutzerkonformer weise.

Im Folgenden werden die einzelnen Funktionen der SemantikUB Business-Logic genauer spezifiziert.

4.3.4 Suche nach Daten

Es muss grundsätzlich möglich sein, nach Daten zu suchen. Dabei muss dem Benutzer die Möglichkeit angeboten werden, nach verschiedenen Kategorien (Fach, Titel, Verlag) innerhalb der Datenbank zu suchen.

4.3.5 Funktion: Über- und untergeordnete Fächer anzeigen

Es muss eine Funktion implementiert werden, mit der es möglich ist, dass die Über- und Unterkategorien eines Fachs angezeigt werden. Nach Auswahl eines Fach erfolgt eine, wie in der Anforderung “Suche nach Daten“, beschriebenen Suche.

4.3.6 Funktion: Alternative Datenbanken zurückliefern

Der im Abschnitt 4.1.2 vorgestellte Berater muss in seiner grundsätzlichen Funktionalität implementiert werden. Dieser soll die Möglichkeit anbieten, einem Benutzer weiteren Datenbanken zu empfehlen.

4.3.7 Integration in ein Web-Framework

Die oben besprochenen Funktionen müssen innerhalb eines Standardbrowsers zugänglich gemacht werden.

Kapitel 5

Design und Realisierung des SemantikUB

Im folgenden Kapitel wird die Umsetzung des Prototypen für das Semantische Web im Bereich der Universitätsbibliothek Heidelberg beschrieben. Der Prototyp hat das Ziel, die verschiedenen Konzepte des Semantischen Webs umzusetzen. Dazu wird zunächst erläutert, welche Komponenten für den Prototypen eingesetzt wurden. Danach erfolgt die Beschreibung der Ontologie. Zuletzt wird angegeben, wie die Business Logik für die Anwendung umgesetzt wurde. Die Resultate werden im Kapitel 6 vorgestellt.

5.1 Die Komponenten des SemantikUB

Für die Entwicklung des Prototypen von SemantikUB boten sich eine Vielzahl im Rahmen von Universitäten entwickelter Werkzeuge an. Bei der Auswahl wurde darauf geachtet, dass die entsprechenden Werkzeuge stabil einzusetzen und die Programme möglichst benutzerfreundlich sind. Dieser Abschnitt beschreibt die Technologien, die für SemantikUB eingesetzt wurden.

5.1.1 Jena-API

Die Jena-API ist ein Java-Framework zum Erstellen von Anwendungen für das Semantische Web. Unter einem Framework, versteht man eine Umgebung, die Funktionen bereitstellt, um dem Entwickler Arbeitsaufwand abzunehmen. Das Jena-Framework wurde von der *Hewlett Packard Labs Semantic Web Research* Gruppe entwickelt. Die aktuelle Version ist Jena 2.2 und steht kostenlos zur Verfügung. Das Jena-Framework besteht aus mehreren Paketen mit denen sowohl RDF- als auch OWL-Dateien verarbeitet werden können. Die folgende Aufzählung beschreibt die fünf wichtigsten Pakete des Jena-Frameworks [Tea05c]:

1. **RDF-API** bietet die grundsätzlichen Funktionalitäten an, die zum Bearbeiten von Ressourcen, Eigenschaften und Literalen notwendig sind. Weiterhin wird mit der

RDF-API das Interface *Model* mitgeliefert. Von diesem Interface gibt es mehrere verschiedene Implementierungen, die alle die Aufgabe haben, die mit RDF oder OWL definierten Ontologien zu verarbeiten. Über das Modell können Operationen, wie Suchen, Hinzufügen oder Löschen, der semantischen Daten durchgeführt werden. Innerhalb von SemantikUB wurde das normale Jena-Modell und das darauf aufbauende *InfModel* verwendet. Letzteres hat die Aufgabe, mit Hilfe eines Reasoners, ein erweitertes Modell anzubieten, das zusätzliches Wissen, z.B. durch Transitivität oder Prädikatenlogik, gewinnen kann.

2. **OWL-API** beinhaltet einen Satz von Klassen, welche die RDF-API, durch die in OWL spezifizierten Möglichkeiten, erweitern. Beispielsweise wird innerhalb der OWL-API zwischen Datentyp- und Objekteigenschaften unterschieden. Die Unterscheidung existiert innerhalb der RDF-API nicht.

3. **Jena-DB** ist ein Paket, welches es ermöglicht, dass die Daten des Jena-Modells persistent, d.h. in einer Datenbank, abgespeichert werden können. Unterstützte Datenbanken sind MySQL, Oracle, PostgreSQL.

4. **RDQL** steht für *Resource Description Query Language* und ist eine Spezifikation des W3C. Das RDQL-Paket setzt diese Spezifikation um und bietet die Möglichkeit an, die in der Wissensbasis gespeicherten Daten, in einer SQL ähnlichen Syntax abzufragen. Ein einfaches Beispiel zeigt die folgende Abfrage 5.1

```
1  SELECT ?ressource ?titel, ?fach
2  WHERE (?ressource, rdf:type, <http://www.ub.uni-heidelberg.
       de#Medium>),
3        (?ressource, dc:title, ?titel)
4        (?ressource, dc:format, ?fach)
```

Beispiel 5.1: Einfache RDQL-Abfrage

Die obige Abfrage sucht nach Ressourcen vom Typ Medium (s. Zeile 2). Das Fragezeichen gibt an, dass es sich um eine Variable handelt. Jeder gefundene Wert wird in der Variablen *?ressource* abgelegt. Des Weiteren wird für jeden gefundenen Wert geprüft, ob die Ressource ein Statement besitzt, das den Titel sowie das Fach angibt. Die Eigenschaften *dc:title* und *dc:subject* müssen innerhalb der jeweiligen Ressource enthalten sein. Die Rückgabemenge könnte folgendermaßen aussehen:

```
ressource       |titel              |fach
=============================================================
ub:Zeitschrift1 |"Java-Magazin"     |ub:Java
ub:Zeitschrift2 |"Medizin-Journal"  |ub:Medizin
ub:Zeitschrift3 |"Technik-Magazin"  |ub:Mechanik
ub:Zeitschrift3 |"Technik-Magazin"  |ub:Elektrotechnik
ub:Zeitschrift3 |"Technik-Magazin"  |ub:Informationstechnik
```

Die erste Spalte der Rückgabemenge gibt den gefundenen Ressourcenamen an, welcher das Subjekt des RDF-Tripels darstellt. Weiterhin wurde für jede gefundene

Ressource der Titel sowie das Fach in Spalte 2 und 3 zurückgeliefert. Die Ressource mit dem Namen *Zeitschrift3* besitzt mehr als ein Fach, weshalb mehrere Zeilen mit jeweils einem Fach zurückgegeben werden.

Besitzt eine Ressource mehrere unterschiedliche Statements, bei denen jeweils die gleiche Eigenschaft auf unterschiedliche Objekte zeigt, kann es schnell passieren, dass die Anzahl der Zeilen für jeden Ressource schnell ansteigt. Die Möglichkeit, mehrmals die gleiche Eigenschaft für eine Ressource zu verwenden, stellt einen großen Unterschied zu relationalen Datenbanken dar, in denen es nicht möglich ist, dass beispielsweise in einem Datensatz eine Spalte mehrere verschiedene Werte enthält.

5. **Reasoner** bieten die Möglichkeit an, zusätzliches Wissen in die Wissensbasis zu integrieren. Diese sind in der Lage, komplexe logische Folgerungen aufgrund vordefinierter Regeln zu schließen. Zum Arbeiten mit einem Reasoner benötigt man immer das Standard Jena-Modell. Diese muss beim beim Erzeugen des Reasoners mit übergeben werden. Der Reasoner liefert nach dem Verarbeiten des Jena-Modells ein erweitertes Modell vom Typ *InfModel* zurück. In SemantikUB wurden die folgenden beiden Reasoner eingesetzt:

   - **OWLReasoner** liest automatisch aus einer Ontologie alle definierten Regeln ein. Anhand dieser logischen Konstrukte wird das Jena-Modell um Statements erweitert. Beispielsweise kann innerhalb von OWL eine transitive Eigenschaft definiert werden. Wird diese Eigenschaft auf eine bestimmte Ressource angewendet, werden wie im Kapitel 2.2 in Abbildung 2.1 gezeigt, automatisch die transitiven Beziehungen für die entsprechende Ressource gesetzt.
   - **Generic Rule Reasoner** bietet eine eigene Regel-Syntax. Die Syntax hat gewisse Ähnlichkeit mit der in RDQL verwendeten Syntax. Mit dem Generic Rule Reasoner ist es möglich, alle in OWL definierbaren Regeln nachzubilden. Allerdings können auch darüber hinaus sehr komplexe und rechenintensive Regeln definiert werden. Im Abschnitt 5.2.2 wird darauf näher eingegangen. Das folgende Beispiel zeigt eine einfache Regel, welche das Kommutativgesetz umsetzt:

     ```
     [(?A liegtBei ?B) -> (?B liegtBei ?A)]
     ```

     Variablen werden in einer RDQL-Abfrage immer durch ein Fragezeichen markiert. Die obige Regel wird für alle Ressourcen in der Wissensbasis durchgeführt. Existiert eine Ressource – dargestellt als Variable *?A* – und verfügt diese über die Eigenschaft *liegtBei*, dann wird das Ziel der Eigenschaft – das Objekt – in *?B* abgespeichert. Als logische Schlussfolgerung wird festgelegt, dass die Variable *?B*, welches auch eine Ressource darstellt, die Eigenschaft *liegtBei* Ressource *?A* zugewiesen bekommt. Damit wird die Ressource *?B* um ein zusätzliches Statement erweitert.

Das Jena-Framework hat sich während der Implementierung des Prototypen sehr gut bewährt und macht einen stabilen, durchdachten und ausgereiften Eindruck. Es wird bereits in vielen Open-Source Projekten, wie beispielsweise Tomcat, standardmäßig mitgeliefert.

Eine ausführliche Dokumentationsseite [Tea05d] sowie ein gut geführtes Forum helfen bei der täglichen Arbeit mit Jena und tragen wesentlich zum grundlegenden Verständnis von Semantischen Web Konzepten bei. Aus diesen Gründen wird das Jena-Framework für das Projekt SemantikUB verwendet werden, um die im Hintergrund laufende Business Logik (s. Lastenheft 4.3.3) zu implementieren.

5.1.2 Modellierungswerkzeug Protégé

Zur Modellierung von Ontologien existiert bereits eine große Auswahl an Programmen. Im Folgenden wird das Modellierungswerkzeug Protégé [Med05a] vorgestellt. Protégé ist in der Lage, den Benutzer bei der Entwicklung von Ontologien zu unterstützen. Es besteht die Möglichkeit, Instanzen zu erzeugen und zu pflegen. Protégé unterstützt OWL DL oder RDF/S. Ein für Protégé vorhandene Java-API ermöglicht es, die modellierte Ontologie wiederzuverwenden und in eigene Java-Programme oder Web-Applikationen zu integrieren. Zum einfacheren Erzeugen von komplexen Klassen, stehen Wizards zur Verfügung, die so genannte Design Patterns umsetzten und für mehrere Objekte bestimmte Aktionen gleichzeitig durchführen. Werden Design Pattern beim Ontologieentwurf eingesetzt, bieten diese Lösungen für häufig auftretende Modellierungsprobleme an [HKR+04].

Protégé verwendet die Jena-API zum Erzeugen von Modellen. Dadurch sind mit Protégé erzeugte Ontologien vollständig kompatibel mit Jena-Projekten. Nicht unterstützt werden RDF-Container und Reifications (s. 3.1.6). Die Möglichkeit zur Integration von Plugins, die Verwendung eines Reasoners und verschiedene Varianten zum Ex- und Import, z.B. HTML, RDF und OWL, machen das Programm zu einem guten Werkzeug für Ontologieentwicklungen.

Protégé unterscheidet zwischen drei verschiedenen Eigenschaftstypen:

1. **Objekteigenschaften** verbinden eine Instanz mit einer anderen Instanz (s. 3.3.1). Da in OWL alles eine Ressource sein kann, ist es möglich, eine Instanz direkt mit einer Klasse zu verbinden. Dazu muss allerdings OWL FULL verwendet werden.
2. **Dateneigenschaften** verbinden eine Instanz mit einem Datenwert (s. 3.3.1).
3. **Annotation Properties** werden dazu verwendet, um Klassen, Eigenschaften oder Instanzen der Ontologie mit verschiedenen zusätzlichen Informationen/Metadaten auszuzeichnen. Eigene Annotation Properties können definiert werden, die entweder Daten- oder Objekteigenschaften sind.

Die folgende Abbildung 5.1 zeigt einen Teil der Oberfläche von Protégé. Darin wird die Konzepthierarchie der für SemantikUB entwickelten Ontologie dargestellt.

Im Feld 1 – Asserted Hierarchy – wird ein Baum gezeigt, welcher die Klassen *Fach* und *Medium* inklusive Unterklassen darstellt.

Die Eigenschaften, die eine einzelne Klasse besitzt, werden im Feld 2 - *Annotation Properties* auf der rechten Seite angezeigt. Im Unterschied zu Properties beschreiben die Annotation Properties direkt eine Klasse.

Die im Feld 3 – Properties – aufgelisteten Eigenschaften, stellen die Eigenschaft dar, die eine Instanz der Klasse aufweist.

In Feld 4 – Asserted Conditions – werden logische Einschränkungen definiert. In diesem Beispiel wird die Klasse, *ElektronischesMedium* so eingeschränkt, dass eine Instanz einerseits vom Typ *Medium* sein muss, und andererseits die Eigenschaft *hatVerfuegbarkeit* auf eine Instanz vom Typ *ElektronischerStandort* zeigen muss (s. 5.2.2). Die im blauen Fenster stehende Information stellt eine Hilfe für diese Bedingung dar.

Feld 5 – Disjoint – gibt an, dass die aktuell markierte Klasse *ElektronischesMedium* verschieden ist von *PhysikalischesMedium*. Eine Inkonsistenz würde beispielsweise entstehen, wenn eine Instanz von Medium gleichzeitig von beiden Klassen *ElektronischesMedium* und *PhysikalischesMedium* abstammen würde.

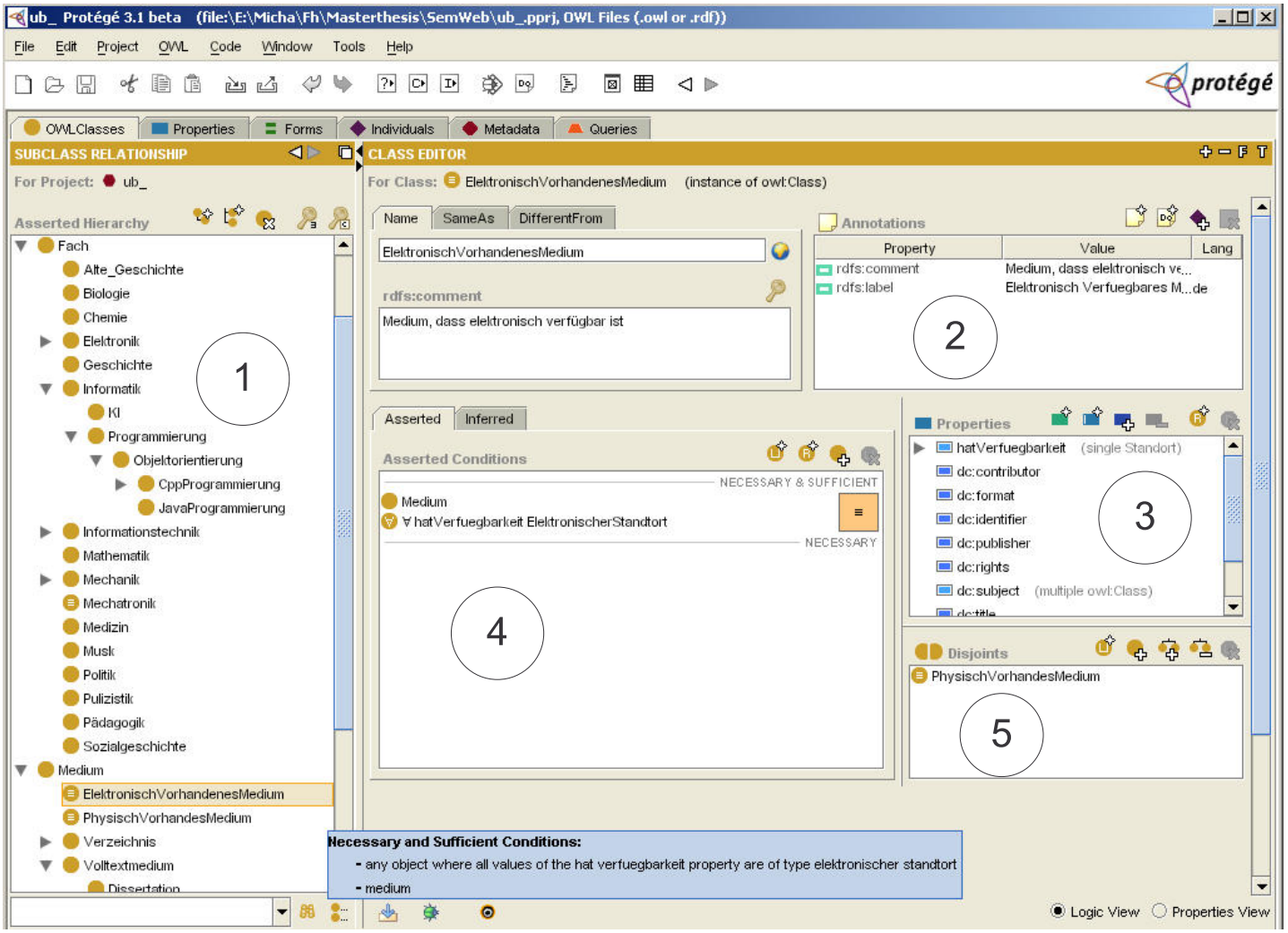


Abbildung 5.1: Modellierungswerkzeug Protégé

Protégé macht einen sehr ausgereiften Eindruck. Auch die Bedienbarkeit des Programms, z.B. Drag & Drop, automatische Fehleranzeige oder der Aufbau der Masken sind sehr

gut gestaltet. Nach der Bearbeitung kann die Ontologie als XML-Datei exportiert werden. Durch die Kompatibilität mit der Jena-API ist gewährleistet, dass die exportierten Daten im nächsten Schritt in Jena verarbeitet werden können. Aus diesen Gründen wird Protégé zur Modellierung der Ontologie verwendet werden.

5.1.3 Cocoon

Cocoon ist ein Projekt der Apache Software Foundation, bei dem es sich um ein in Java geschriebenes Publishing-Framework für Web-Inhalte handelt. Ein Web-Publishing-Framework ist ein spezielles Framework zur Entwicklung von Webseiten. Cocoon wurde im Rahmen des Apache XML-Projekts entwickelt und wird von vielen Programmierern weltweit ständig weiterentwickelt. Zurzeit liegt Cocoon in der Version 2.1.7 vor.

Das Konzept von Cocoon baut im Vergleich zu anderen webbasierten-Frameworks auf einem neuen Ansatz auf. Bei HTML-Dokumenten werden i.d.R. die Schichten Inhalt, Layout und Programmierlogik fest miteinander verbunden. Cocoon geht einen anderen Weg zur Veröffentlichung der Informationen. Die drei Schichten Inhalt, Layout und Programmierlogik werden strikt voneinander getrennt und müssen in separaten Dateien bearbeitet werden. Um das Konzept der Trennung der einzelnen Schichten in Cocoon umzusetzen, wurde XML gewählt, da XML diese Anforderungen konsequent erfüllt.

Eine weit verbreitete und bereits in Cocoon 1.x eingesetzte Möglichkeit, um Cocoon Applikationen zu entwickeln, ist das Konzept der *eXtensible-Server-Pages* (XSP). Eine eXtensible-Server-Page ist ein normales und gültiges XML-Dokument. Dieses enthält i.d.R. einen statischen Inhalt oder bestimmte Funktionen, die einen dynamischen Inhalt aus einer Datenquelle (Datenbank, Hash-Tabelle, Datei) einlesen.

Im Gegensatz zu einer normalen Server Page liefert die XSP-Datei kein HTML-, sondern ein XML-Dokument zurück. Die XSP-Seite besitzt zusätzlich zu den bereits erklärten Möglichkeiten einen optionalen Logik-Bestandteil ,in welchem Java-Code integriert werden kann [Mil04].

Da Cocoon sehr leicht die mit Jena entwickelte Business Logik integrieren und verarbeiten kann, wird für die Präsentationsschicht, welche die Semantischen Web Daten darstellt, Cocoon verwendet werden.

5.1.4 OWL

Die Ontologiesprache OWL wurde bereits ausführlich im Kapitel 3.3 besprochen. Diese bietet im Vergleich zu RDFS Erweiterungen an, durch welche eine stärkere Logik integriert werden kann. Verschiedene Tests mit Reasonern haben gezeigt, dass diese durchaus in der Lage sind, aus den in OWL definierten logischen Konzepten, Rückschlüsse zu ziehen und weiterführende Informationen zu liefern, die sonst nur durch menschliche Intelligenz gefolgert werden könnten.

Zur vollen Nutzung der Ontologie wird OWL FULL verwendet, welches den Vorteil bietet, dass auch Konzepte und nicht nur Instanzen als Ressourcen im Jena-Modell verwendet werden können.

5.2 Ontologieentwurf von SemantikUB

Dieser Abschnitt stellt das entwickelte Ontologiemodell vor. Die Ontologie für SemantikUB wurde nach den Kriterien, die im Abschnitt 2.4.3 beschrieben wurden, entworfen. Im Folgenden werden die wichtigsten Schritte und Begründungen, die zur Entwicklungen der Ontologie notwendig waren, angegeben.

Die Ontologie hat die Aufgabe, elektronische Medien zu beschreiben und dem Benutzer eine Suchfunktion anzubieten. Die Ontologie wird weiterhin von den Mitarbeitern der Universitätsbibliothek Heidelberg verwendet werden, um beispielsweise neue Daten einzugeben oder zu pflegen. Keine der vorhanden Ontologien konnte die gestellten Anforderungen abdecken. Untersucht wurden die Ontologiedatenbanken Schemaweb [Tea05e] und DAML Ontology Library [Tea04a]. Aus diesem Grund wird eine neue Ontologie entwickelt. Die vorhandenen Möglichkeiten zum Auszeichnen von Konzepten und Instanzen durch den *rdfs*- und Dublin Core-Namensraum werden eingesetzt. Der Dublin Core eignet sich hervorragend zum Beschreiben von Medien. Dieser wird bereits von einigen großen amerikanischen Bibliotheken verwendet [RC04]. Für die Entwicklung der Konzepthierarchie wird die Top-Down Entwurfsmethode eingesetzt, weil während der Ontologieentwicklung zusätzliches Wissen aus den verschiedenen bibliothekarischen Bereichen, integriert werden mussten. Die folgenden Punkte beschreiben die Konzepte und Eigenschaften und begründen die Art und Weise, auf die ein bestimmtes Konzept in die Ontologie integriert wurde.

Medium

Medium ist das wichtigste Konzept in der Ontologie. Alle elektronischen Medien werden in diesem Konzept erfasst. Mögliche Erweiterungen für physikalische Medien, z.B. Bücher, werden bereits beachtet. Dazu wurde *Medium* unterteilt in zwei Unterklassen *ElektronischesMedium* und *PhysikalischesMedium*. Jedes Medium wird mit Hilfe des Reasoners einer dieser beiden Kategorien zugewiesen. Die verschieden Medientypen werden anhand der Katalogliste [MBL05a] zusammengefasst.

Fach

Fach stellt eine Taxonomie dar, welche alle vorhandenen Fächer beschreibt mit denen ein Medium ausgezeichnet werden kann. Um die Fächerhierachie zu bilden, wurde die Möglichkeit untersucht, den Schlagwortkatalog (s. 4.1.4) als Taxonomie in der Ontologie zu integrieren. Die Schlagwörterliste bietet eine große Menge (mind. 1.000.000) von Begriffen an. Diese Wörter sind nicht hierarchisch angeordnet und besitzen nur eine schwache semantische Beziehung zueinander. Deshalb kann mit Schlagworten, die im Lastenheft (s. 4.3.5) geforderte Möglichkeit, zum Anfordern von über- und untergeordneten Fächern, nicht realisiert werden. Aus diesem Grund wird eine eigene Fächerhierachie entwickelt, um zu erkennen, welche Anforderungen an eine Integration des Schlagworkataloges in die Wissensbasis gestellt werden.

Jedes Fach steht durch die *rdfs:subClassOf*-Beziehung mit den über- und untergeordneten Fächern in Verbindung. Allerdings jeweils nur mit dem direkt über- und untergeordneten Fach. Deshalb wurde eine Annotation Property namens *istTeilgebiet* erzeugt. Diese Eigenschaft wird jedem Fach zugewiesen und zeigt auf die übergeordnete Klasse. Die Eigenschaft ist Transitiv wodurch ein Fach nicht nur Teilgebiet des direkt übergeordneten Fachs, sondern auch weiterer übergeordneter Fächer ist (s. Abbildung 2.1).

Des Weiteren wurde eine inverse Eigenschaft *hatTeilgebiet* von *istTeilgebiet* erzeugt. Auf diese Weise kann ein Reasoner sehr leicht alle untergeordneten Fächer angeben und damit auch die dazugehörigen Medien. Sowohl Fächer-Klassen als auch Instanzen von *Medium* zeigen direkt auf Unterklassen von *Fach*. Lediglich OWL FULL erlaubt das direkte verweisen auf Klassen und nicht auf deren Instanzen. Aus diesem Grund wird, wie bereits erwähnt, OWL FULL als Ontologiesprache verwendet.

Zur Internationalisierung wurden die Eigenschaften des Dublin Cores in der eigenen Ontologie um ein Annotation Statements erweitert, welches die deutsche *rdfs:label*-Bezeichnung enthält. Des Weiteren wurde *dc:subject* in eine Objekteigenschaft umgewandelt, da diese auf Ressourcen vom Typ *Fach* zeigen muss. Ob solche Erweiterungen bereits spezifizierter und weit verbreiteter RDF-Dokumente zu Problemen führen können, z.B. beim Einsatz von mehreren Web-Services, konnte in Rahmen der Masterthesis nicht geklärt werden. Auf der W3C-OWL-Webseite [Tea05f] wird im Abschnitt 5 jedoch ebenfalls eine Dublin Core Eigenschaft für die eigenen Zwecke modifiziert. Innerhalb von SemantikUB führte die Erweiterung zu keinen Problemen.

Standort

Standort beschreibt die verschiedenen Standorte innerhalb der Universitätsbibliothek, an denen sich ein Medium befinden kann. Es werden lediglich zwei Unterklassen – *PhysikalischerStandort* und *EletronischerStandort* – definiert. Von diesen werden Instanzen wie beispielsweise `Lesesaal`, `Magazin` oder `WEB` erzeugt.

Instanzen

Zum Erzeugen von Instanzen wurde auf vorhandene Altdaten zurückgegriffen. Für den Import von Altdaten stehen lediglich die Daten des Verzeichnisses der elektronische Zeitschriftenbibliothek [Tea05a] zur Verfügung. Dieses bietet eine Liste aller elektronischer Zeitschriften, die von der Universitätsbibliothek Heidelberg angeboten werden. Jedoch fehlt eine Angabe des zugehörigen Fachs. Für die elektronischen Datenbanken existieren bisher nur Informationen, welche direkt in entsprechenden Webseite statisch integriert wurden, beispielsweise auf der Katalogseite der Universitätsbibliothek Heidelberg [MBL05b]. Daher kann für diesen Bereich kein Export aus einer Datenbank erfolgen. Für Datenimport wurde eine Java-Klasse geschrieben, welche die Zeitschriftendaten in Form einer Textdatei einliest und verarbeitet. Jeder Datensatz wurde im Jena Model als Ressource mit den dazugehörigen Eigenschaften angelegt. Da jede Ressource eine einmalige ID benötigt wurde eine Funktion geschrieben, welche eine sehr lange Zufallszahl erzeugt. Diese wird als ID für die jeweilige Ressource verwendet.

5.2.1 SemantikUB-Ontologie

Die folgende Abbildung 5.2 zeigt die grundsätzliche Funktionalität der Ontologie, unabhängig von logischen Aspekten. Darin sind bereits einige Instanzen erzeugt worden, welche die konkreten Beziehungen zueinander aufzeigen.

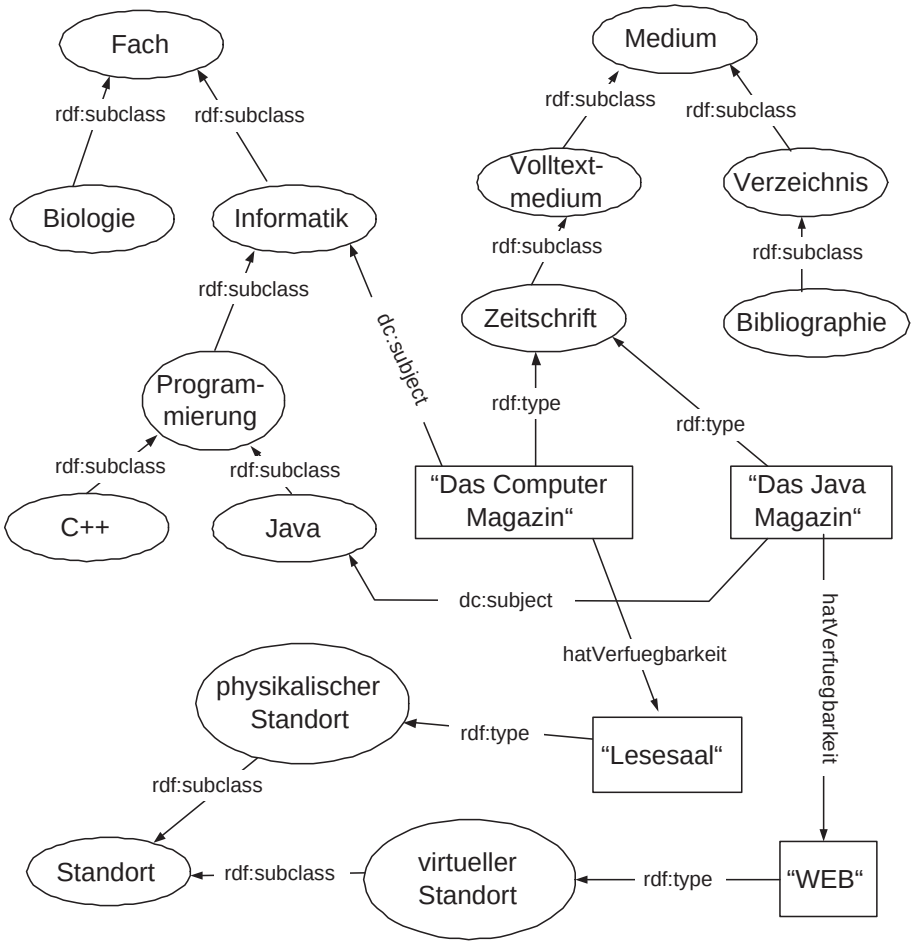


Abbildung 5.2: SemantikUB Ontologie

5.2.2 Integration von logischen Aspekten

Die logischen Aspekte innerhalb eines Semantischen Webs bieten den Vorteil an, dass zusätzliches Wissen in der Ontologie erzeugt werden kann, ohne dabei die eigentlichen Daten verändern oder erweitern zu müssen. Das zusätzliche Wissen wird stattdessen durch Bedingungen und Einschränkungen definiert, von einem Reasoner verarbeitet und auf die Daten der Ontologie angewendet. Die folgenden Beschreibungen zeigen zwei Möglichkeiten, wie in SemantikUB zusätzliches Wissen, anhand der in der Ontologie hinterlegten logischen Einschränkungen, erschlossen wird.

Virtueller Berater

Der virtuelle Berater gibt Informationen an den Benutzer weiter, wie dieser Kataloge oder Datenbanken verwenden kann. Dazu muss der Berater wissen, ob das Medium in elektronischer oder physikalischer Form existiert. Dadurch kann er dem Benutzer Auskunft geben, dass dieser beispielsweise lediglich im Internet Zugriff auf das Medium hat. Aus diesem Grund wurden zwei neue Konzepte in die Ontologie integriert, die beschreiben, ob es sich um ein elektronisches- oder physikalisches Medium handelt (s. Konzeptspezifikation 5.2.3).

Im Folgenden wird das Konzept des virtuellen Beraters anhand der Ressource *ElektronischesMedium* erläutert. Analoges gilt für *PhysikalischesMedium*. Die folgende Abbildung 5.3 zeigt einen Ausschnitt der Ontologie. Diese wurde um eine Klasse *ElektronischesMedium* erweitert. Diese Klasse sowie deren Eigenschaften sind besonders markiert, da eine Instanz von Medium nur unter bestimmten Bedingungen vom Typ *ElektronischesMedium* sein kann.

In der Abbildung 5.3 ist beispielsweise eine Instanz von `Zeitschrift` namens “Das Computer Magazin“ auch vom Typ *ElektronischesMedium*, weil der Standort der Zeitschrift eine Instanz von *ElektronischerStandort* ist.

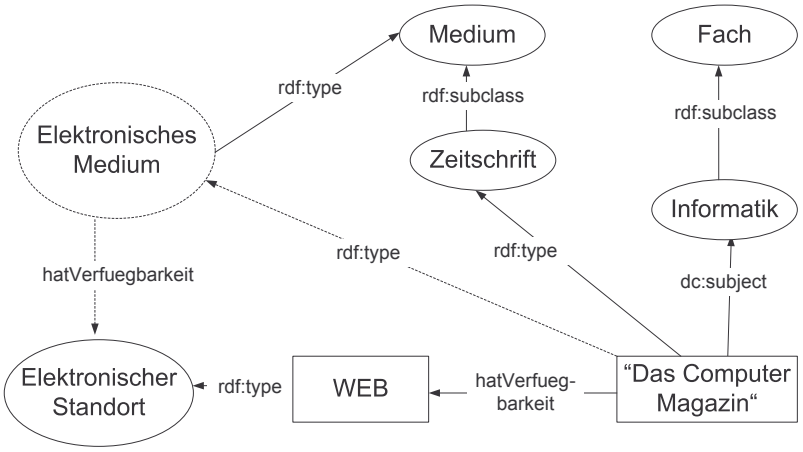


Abbildung 5.3: Erkennen des Medientyps - abhängig vom Standort

Allgemein ausgedrückt, ist eine Instanz von Medium nicht nur vom Typ Medium, sondern auch vom Typ *ElektronischesMedium*, wenn sich diese Instanz ausschließlich an Standorten befindet, die vom Typ *ElektronischerStandort* sind. Die Eigenschaft *hatVerfuegbarkeit*, welche jede Instanz von Medium mit einem Standort verbindet, wurde als *functional* definiert, wodurch ein Medium nur einen Standort haben darf. Durch diese Information kann der virtuelle Berater sehr leicht erkennen, um was für ein Medium es sich handelt und den Anwender über die weitere Vorgehensweise zur Beschaffung des gewünschten Mediums beraten.

Das Beispiel 5.2 verwendet die Semantische Web Sprache OWL um eine Klasse *ElektronischesMedium*, wie in Abbildung 5.3 gezeigt, zu definieren und diese mit der oben beschriebenen Einschränkung zu belegen.

```
<owl:Class rdf:ID="ElektronischesMedium">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hatVerfuegbarkeit"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#ElektronischerStandort"/>
      </owl:allValuesFrom>
    </owl:restriction>
  </owl:intersectionOf>
</owl:Class>
```

Beispiel 5.2: OWL-Definition für ElektronischesMedium

Durch das Attribut *owl:intersection* im obigen Beispiel, wird eine Schnittmenge über die passenden Ressourcen gebildet. Der Parse Typ der Vereinigungsmenge gibt an, dass es sich um eine Collection, d.h. eine Sammlung von verschiedenen Ressourcen handelt.

Die eigentliche Einschränkung wird durch das Attribut *owl:restriction* definiert. Innerhalb dieser wird die oben beschriebene Regel definiert. Das *owl:onProperty*-Attribut besagt, dass die Objekteigenschaft *hatVerfuegbarkeit* so eingeschränkt wird, dass **alle** Objekte der Eigenschaft – *owl:allValuesFrom* – vom Typ *ElektronischerStandort* sein müssen. Trifft diese Bedingung zu, wird die entsprechende Ressource in die Schnittmenge aufgenommen und ist damit auch vom Typ *ElektronischesMedium*.

Neue Fächer definieren

Eine Anforderung im Lastenheft besagt, dass ein Medium mit weiteren Fächern ausgezeichnet werden kann, ohne dass dieses Medium mit zusätzlichen Informationen versehen werden muss. Die nächste Abbildung 5.4 stellt diese Anforderung grafisch dar.

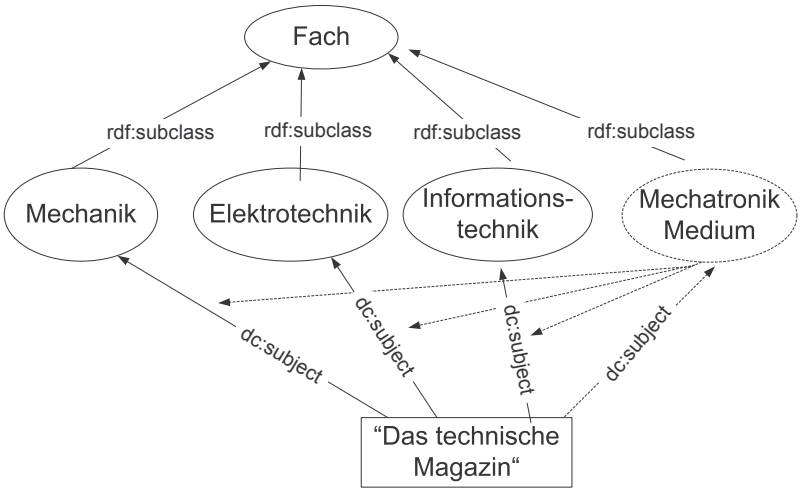


Abbildung 5.4: Neue Fächer definieren

Das Fach Mechatronik und dessen Eigenschaften wurden in der Abbildung 5.4 besonders gekennzeichnet, da dieses Wissen anhand von logischen Schlussfolgerungen aufgebaut wird. Dabei wird das bereits in 4.1.4 vorgestellte Szenario wieder aufgegriffen, in welchem ein Medium die Fächer Mechanik, Elektronik und Informationstechnik besitzt. Ein weiteres Fach Mechatronik muss diesem und allen anderen Medien zugeordnet werden, welche mit den drei Fächern bereits ausgezeichnet worden sind. Durch die im Lastenheft geforderte Bedingung, muss es möglich sein, alle Medien, welche die Bedingung erfüllen, mit dem zusätzlichen Fach auszuzeichnen.

Es wurde keine Möglichkeit gefunden um die Regel direkt mit Protégé zu realisieren, da es in Protégé nur möglich ist, Instanzen so zu definieren, dass diese auch vom Typ einer anderen Klassen sind. Dabei wird die Eigenschaft *rdf:type* der entsprechenden Instanz zugewiesen. Es scheint jedoch nicht möglich zu sein, eine Regel zu definieren, bei welcher beispielsweise eine Instanz die Eigenschaft *dc:subject* zugewiesen bekommt.

Aus diesem Grund wurde der von Jena zur Verfügung gestellte Generic Rule Reasoner (s. 5.1.1) eingesetzt. Dieser bietet eine ähnliche, aber doch von OWL unterschiedliche Syntax. Mit Hilfe des Generic Rule Reasoners wurde eine allgemeine Regel erstellt, welche für alle zutreffenden Instanzen Schnittmengen erstellt. Die folgende Abbildung 5.5 verdeutlicht das Konzept.

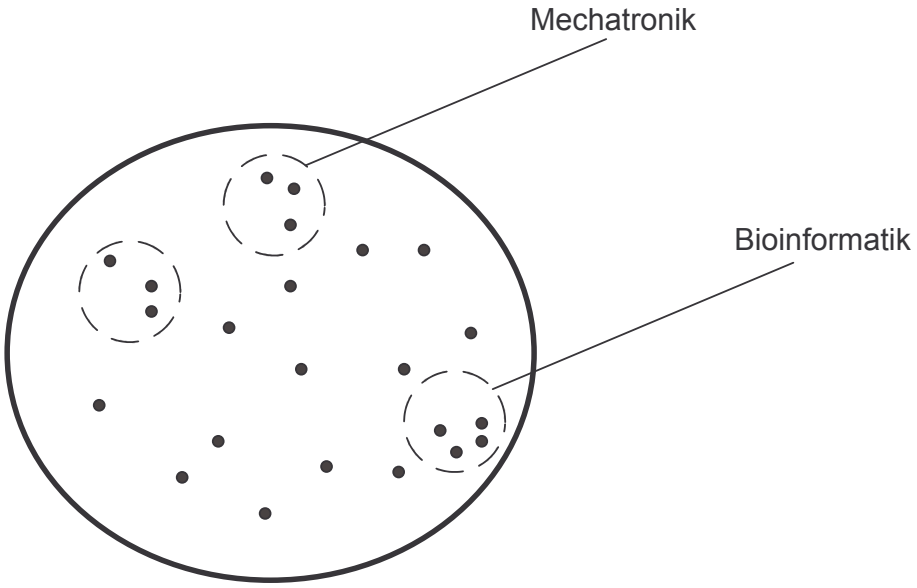


Abbildung 5.5: Vereinigungsmengen

Der große Kreis stellt die Wissensbasis mit allen Instanzen (kleine Punkte) dar. Durch den Generic Rule Reasoner wurde eine Regel definiert, welche die Aufgabe hat, lediglich allgemeine Vereinigungsmengen zu bilden. Die Einschränkung für jede Vereinigungsmenge wurde so festgelegt, dass Medien, welche auf die gleichen Fächerkonzepte zeigen, in einer Vereinigungsmenge gesammelt werden. Die einzelnen Vereinigungsmengen werden durch die kleinen Kreise dargestellt. Es können beliebig viele Vereinigungsmengen entstehen. Wie bereits erwähnt, werden ausschließlich allgemeine Vereinigungsmengen gebildet. Diese besitzen noch keine Ressourcenamen, sondern werden durch einen leeren Knoten (s. 3.1.6) gekennzeichnet.

Erst durch einen zusätzlichen Eintrag innerhalb der Ontologie werden die Vereinigungsmengen zu konkreten Ressourcen, d.h. sie erhalten Ressourcenamen und werden damit spezialisiert. Die Abbildung 5.5 zeigt zwei solcher Spezialisierungen, nämlich *Mechatronik* und *Bioinformatik*, wobei letztere die Kombination von nur zwei Fächern darstellt. Die dritte Vereinigungsmenge besitzt noch keine Bezeichnung, da hierfür in der Ontologie keine Bezeichnung definiert wurde. Möglicherweise wird irgendwann ein Fach entstehen, welches die drei Fächer, ähnlich dem Mechatronik Beispiel, zusammenfasst und einen Mitarbeiter der Bibliothek dazu veranlasst, ein solches Fach zu definieren.

Zusammenfassend kann gesagt werden, dass der Generic Rule Reasoner alle möglichen Vereinigungsmengen bildet. Spezifiziert werden diese jedoch erst durch eine in weiteres Konzept, das in der Ontologie definiert wird. Im folgenden Beispiel wird die Regel, welche der Generic Rule Reasoner übergeben bekommt, beschrieben.

```
[Rule1: (?s owl:intersectionOf ?menge)
      (?menge rdf:first ?s1) (?menge rdf:rest ?menge1)
      (?menge1 rdf:first ?s2) (?menge1 rdf:rest ?menge2)
      (?menge2 rdf:first ?s3) (?menge2 rdf:rest rdf:nil)
      (?medium dc:subject ?s1)
      (?medium dc:subject ?s2)
      (?medium dc:subject ?s3)
    ->
      (?medium dc:subject ?q)]
```

Beispiel 5.3: Allgemeine Generic Rule Reasoner Regel

Die Regel ist eine allgemeine Definition, welche innerhalb der Wissensbasis Schnittmengen bildet. Die erste Variable namens *?s* steht für das gesamte Konzept der Schnittmenge. Die Schnittmenge setzt sich aus den einzelnen Variablen *?s1*, *?s2* und *?s3* zusammen. Durch *rdf:first* wird immer eine Ressource aus der Gesamtmenge *menge* entnommen, durch *rdf:rest* wird eine neue Restmenge *menge1* und *menge2* ohne die entnommene Ressource gebildet. An dieser Stelle werden nicht wirklich Ressourcen aus der Gesamtmenge entnommen, sondern es wird lediglich deklariert, dass *?s1*, *?s2* und *?s3* jeweils eine Ressource aus der Gesamtmenge darstellen. Welche Ressourcen das konkret sind, wird erst im nächsten Schritt festgelegt.

Der zweite Teil der Regel deklariert eine Variable *?medium*, welche beispielsweise eine Ressource vom Typ Medium sein kann. Im Folgenden wird auch die eigentliche Bedingung der Regel definiert. Diese besagt, dass *?medium* eine Eigenschaft namens *dc:subject* haben muss, welche auf die Variablen *?s1*, *?s2* und *?s3* zeigt. Dadurch das *?s1*, *?s2* und *?s3* ganz allgemein definiert wurden, werden alle möglichen Schnittmengen gebildet. Alle Ressourcen – dargestellt durch *?medium* – kommen in die gleiche Schnittmenge, die dreimal die Eigenschaft *dc:subject* aufweisen und auf die gleichen drei *?s1*, *?s2* und *?s3* zeigen. Als Folgerung für jede Schnittmenge gilt, dass alle *?medium* in einer Schnittmenge auch vom Typ ?s sind. Für jede Schnittmenge existiert ein separates *?s*, repräsentiert durch einen leeren Knoten.

Analog dazu muss eine Regel für zwei und möglicherweise vier Fächer implementiert werden. Für ein Fach genügt es mit dem OWL-Attribut owl:sameAs eine Äquivalenz zwischen zwei Fächern auszudrücken. Die Regel kann höchstwahrscheinlich nicht rekursiv,

für alle möglichen Anzahlen an Fächern, implementiert werden, weil dadurch die Gefahr besteht, dass es unendlich viele Schnittmengen geben kann.

Im nächsten Schritt müssen den Variablen *?s*, *?s1*, *?s2* und *?s3* konkrete Bezeichnungen gegeben werden, um bestimmte Muster zu spezifizieren. Diese Definitionen können mit Hilfe der allgemeinen Regel in Protégé von Mitarbeiten erzeugt werden. Diese wird in OWL folgendermaßen definiert:

```
<owl:Class rdf:ID="Mechatronik">
   <rdfs:subClassOf rdf:resource="#Fach"/>
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Mechanik"/>
      <owl:Class rdf:about="#Elektronik"/>
      <owl:Class rdf:about="#Informationstechnik"/>
   </owl:intersectionOf>
</owl:Class>
```

Beispiel 5.4: OWL Intersection

Es wird wiederum wie im Beispiel 5.2 eine Vereinigungsmenge gebildet, die diesmal jedoch nur Klassen und keine Eigenschaften in die Schnittmenge aufnimmt. Die einzelnen Klassen *Mechanik*, *Elektronik* und *Informationstechnik* repräsentieren die allgemeinen Variablen *?s1*, *?s2* und *?s3*. Durch diese OWL-Definition wird eine allgemeine Schnittmenge, welche die Generic Rule Reasoner erstellt hat, spezifiziert.

Im Vergleich zur obigen Reasoner-Regel könnte auch eine spezifizierte Regel definiert werden, welche im folgenden Beispiel gezeigt wird. Darin wird nicht mit den Variablen *?s1*, *?s2*, *?s3* und *?s* gearbeitet. Diese werden nicht benötigt, da es sich bei dieser Regel bereits um eine Spezifikation handelt, d.h. es wird von von Anfang an festgelegt was *?s1*, *?s2*, *?s3* und *?s* sind. Dadurch wird nur eine Schnittmenge gebildet. Der große Nachteil bei diesem Konzept ist es, dass die Regel auf diese Weise nicht mit Protégé definierbar ist. Stattdessen muss die Regel direkt in der Java Klasse für den Generic Rule Reasoner implementiert werden. Eine weitere Regel würde dadurch ein erneutes Kompilieren bestimmter Klassen mit sich ziehen. Aus diesem Grund wurde das komplexe aber allgemeine Konzept bevorzugt.

```
[Regel: (ub:Mechatronik owl:intersectionOf ?menge)
      (?medium dc:subject ub:Mechanik)
      (?medium dc:subject ub:Elektronik)
      (?medium dc:subject ub:Informationstechnik)
     ->
      (?medium dc:subject ub:Mechantronik)]
```

Beispiel 5.5: Spezialisierte Generic Rule Reasoner Regel

5.2.3 Konzeptsepezifikation

Im folgenden werden die genauen Relationen jedes Konzeptes, das für SemantikUB definiert wurde beschrieben. Die folgenden Tabellen enthalten Angaben zu den Eigenschaften, Unterklassen und logischen Konstrukten des jeweiligen Konzeptes.

Konzept **ub:Standort**		
Beschreibung	Konzept beschreibt die verschiedenen Standorte, an denen ein Medium hinterlegt sein kann.	
Eigenschaften		
Name	Bereich	Bemerkung
dc:title	xsd:string	modified Property
ub:hatLage	xsd:string	functional
Unterklassen		
Name	Beschreibung	
Elektronischer-Standort	Standort, der nicht physikalisch vorhanden ist, sondern ausschließlich mit Hilfe elektronischer Geräte, wie Computer oder DVD-Abspielgeräten erreicht werden kann.	
Physikalischer-Standort	Standort, der physikalisch vorhanden ist.	

Konzept **ub:Fach**		
Beschreibung	Konzept beschreibt eine Fächertaxonomie. Es existieren so genannte Grundfächer welche direkt unterhalb von `Fach` angeordnet sind. Diese werden auf den nächst tieferen Ebenen genauer spezifiziert.	
Eigenschaften		
Name	Bereich	Bemerkung
rdfs:subClassOf	owl:class `Fach`	Ober- und Unterklassenbeziehung
istTeilgebiet	owl:class `Fach`	transitive Property
hatTeilgebiet	owl:class `Fach`	inverse Property `istTeilgebiet`
Unterklassen		
Anmerkung	Es existieren sehr viele Unterklassen. Diese stellen jeweils eine spezifischere Fächerbeschreibung dar (s. Protégé-Projekt).	

Konzept **ub:Medium**	
Beschreibung	Allgemeines Medium, welches in der UB vorhanden sein kann. Dient zur Definition der Eigenschaften, die jedes Medium besitzt.

Eigenschaften

Name	Bereich	Bemerkung
dc:contributor	xsd:string	modified Property
dc:format	xsd:string	modified Property
dc:identifier	xsd:string	modified Property
dc:publisher	xsd:string	modified Property
dc:rights	xsd:string	modified Property
dc:subject	ub:Fach	modified Property; classes allowed
dc:title	xsd:string	modified Property
ub:hatStandort	ub:Standort	only individual, functional

Unterklassen

Name	Beschreibung

Volltextmedium	Medium, das ganze Texte enthält. Mögliche Unterklassen:
	`Dissertation`
	`Musiknoten`
	`Urkunden`
	`Volltextsammlung`
	`Zeitschrift`
	`Zeitung`

Verzeichnis	Medium, das nur Verzeichnisse enthält. Mögliche Unterklassen:
	`Bibliographie`
	`Biographieverzeichnis`

Physikalisches-Medium	Klasse beschreibt Medium, das physikalisch verfügbar ist durch logische definierte Einschränkung: $\forall$ ub:hatVerfuegbarkeit (PhysikalischerStandort)
Elektronisches-Medium	Klasse beschreibt Medium, das elektronisch verfügbar ist durch logische definierte Einschränkung: $\forall$ ub:hatVerfuegbarkeit (ElektronischerStandort)

5.3 Implementierung von SemantikUB

Dieser Abschnitt befasst sich mit der Implementierung von SemantikUB. Dabei wird aufgezeigt, wie weit fortgeschritten die eingesetzte Semantische Web Technologie bereits ist und welcher Aufwand für die Umsetzung notwendig war. Der erste Teil dieses Abschnitts stellt die implementierten Java-Klassen vor, welche die im Lastenheft beschriebenen Anforderungen umsetzt. Danach folgt eine Beschreibung, wie diese Klassen in Cocoon integriert wurden.

5.3.1 Allgemeine Beschreibung der Implementierung

Das in Abbildung 5.6 gezeigte UML-Klassendiagramm stellt den Kern von SemantikUB dar. Zum Erhalt der Übersichtlichkeit sind die Parameter der einzelnen Methoden nicht aufgelistet. Im Folgenden wird eine einführende Beschreibung der Aufgaben jeder einzelnen Klasse gegeben. Eine Spezifikation findet sich im nächsten Abschnitt.

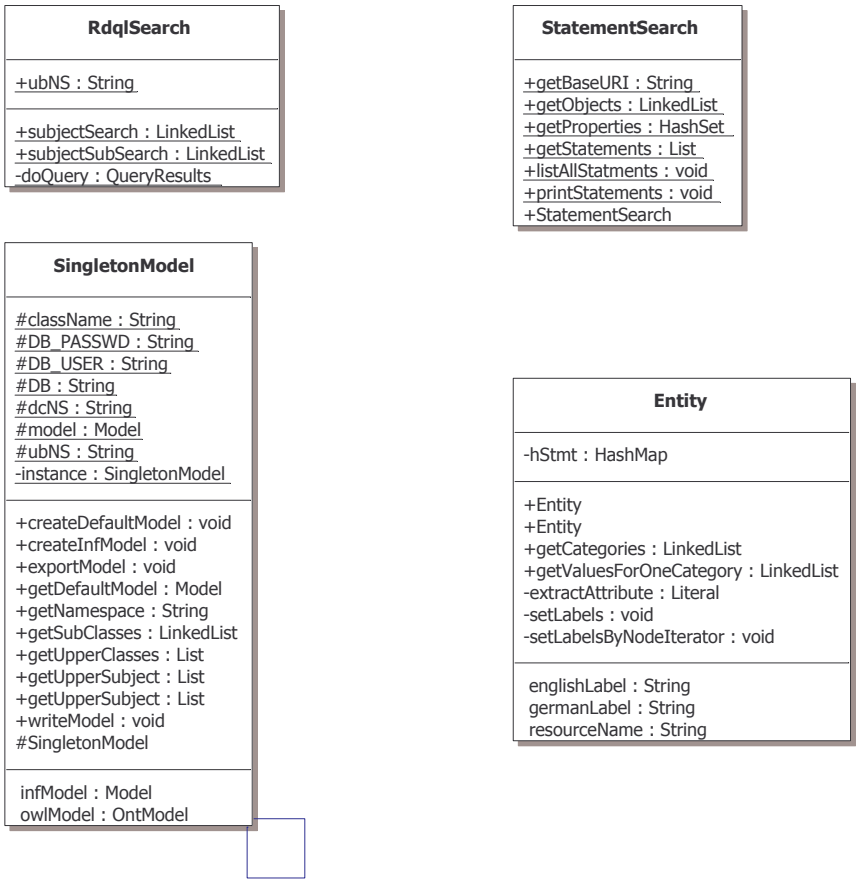


Abbildung 5.6: Business Logik Modell

Die Klassen *SingletonModel*, *RDQL-Search* und *StatementSearch* sind statische Klassen. Von diesen wird nur eine Instanz in der ganze Anwendung benötigt.

Die Klasse *SingletonModel* bildet die Hauptklasse und dient als Schnittstelle, um auf die *model*- und *infModel*-Variable zugreifen zu können. Innerhalb dieser beiden Variablen

werden die Ressourcen, Eigenschaften und Objekte abgelegt. Durch die Jena-API können diese manipuliert werden.

Wie bereits erwähnt, handelt es sich beim *SingletonModel* um eine statische Klasse. Dazu wurde das so genannte *Singleton*-Design Pattern verwendet [Gea03].

Es existieren zwei Möglichkeiten, um nach Daten zu suchen. Einerseits kann eine so genannte Statement-Suche durchgeführt werden, andererseits wird die erweiterte Suchfunktionalität der RDQL-Suche angeboten. Die Statement-Suche hat die Aufgabe, alle oder nur bestimmte Statements für eine oder mehrere Ressourcen zu suchen. Bei der RDQL-Suche wird anhand von übergebenen Parametern ausschließlich nach Medien und deren Eigenschaften, z.B. Titel oder Fach, gesucht.

Die von der RDQL-Klasse durchgeführte Suche enthält viele Rückgabewerte, die es in einem entsprechenden Format an die Präsentationsschicht zurückgeben muss. Aufgrund der zusätzlich durchgeführten Statement-Suche sind immer einige Statements in der Lösungsmenge enthalten, die für eine SemantikUB-Suche überflüssig sind. Beispielsweise ist ein häufig vorkommendes überflüssiges Statement, dass eine Instanz von der Klasse *owl:thing* abstammt. Da alles von *owl:thing* abstammt, wird diese Information nicht benötigt.

Des Weiteren werden Statements mit leeren Knoten zurückgeliefert. Diese enthalten eine temporäre ID und dienen lediglich dem Jena Model für bestimmte Zwischenspeicherungen von Informationen, z.B. Schnittmengen oder Vereinigungen. Die nicht benötigten Ergebnisse einer Suche müssen ausgefiltert werden, da diese keine Relevanz oder sogar störend für einen Anwender von SemantikUB sind. Aus diesem Grund wurden verschiedene Methoden geprüft, um die Daten einheitlich an den Anwender zurückzuliefern. Dabei soll das Ziel sein, dass innerhalb der Präsentationsschicht nicht mehr auf die Jena-API zugegriffen werden muss. Stattdessen müssen die Daten einheitlich zurückgeliefert werden und leicht auszulesen sein.

Factory

Eine Variante zur Realisierung dieser Aufgabe wäre die Möglichkeit, so genannte *Factories* zu verwenden. Eine Factory hat dabei die Aufgabe für jede benötigte Instanz, ein entsprechendes Java-Objekt zu generieren [Gar04]. Diese bietet den großen Vorteil, dass man einfach die gefundenen Ressourcen an die Factory übergeben muss und diese generiert automatisch, beispielsweise Zeitschriften-, Monographie- und Promotionsobjekte. Der Entwickler muss sich nicht um die einzelnen Konzepttypen kümmern. Für den Einsatz von Factories muss jedoch für jedes Konzept eine Java-Klasse implementiert werden. Protégé bietet diese Möglichkeit an, direkt aus den Konzepten heraus Java-Klassen zu erzeugen. Durch diese automatisch generierten Klassen wird es möglich, dass man an die Präsentationsschicht nur Objekte im objektorientierten Sinn zurückliefert.

Das Konzept der Factory bietet den Vorteil, jedes einzelne Semantische Web Konzept als objektorientierte Klasse anzusprechen und auf die Inhalte zugreifen zu können. Ein Problem, das bei der Verarbeitung der erhaltenen Daten vom Jena-Modell auftrat war, dass

bei Ontologien mit Graphen gearbeitet wird, weshalb es häufig passiert, dass eine Eigenschaft auf mehrere Objekte zeigen kann (s. RDQL-Beschreibung in 5.1.1). Aus diesen Grund müssten die Attribute der generierten Java-Klassen *Collections* und nicht als einfache Datentypen repräsentiert werden, da man niemals sicher sein kann, wie viele Elemente eines Typs existieren. Da für jedes Konzept eine eigene Klasse existieren muss, kann mit einem großen Pflegeaufwand gerechnet werden, weil die Ontologie noch oft geändert wird. Ein bereits entwickeltes Factory-Konzept, z.B. durch Jena, konnte nicht gefundenen werden, weshalb eine Eigenentwicklung einen zusätzlichen Aufwand darstellen würde. Aus diesen Gründen wird im Augenblick auf diese Möglichkeit verzichtet werden.

XML-Export

Eine weitere Idee zur Realisierung der oben beschriebenen Zwischenschicht zwischen dem Jena-Model und Präsentationsschicht war, die Daten direkt nach XML zu konvertieren. Ein XML-Export der gewünschten Daten aus dem Jena-Model wird nicht angeboten. Weiterhin liefert Jena die Daten nicht in einem XML-Format zurück, sondern als Statement-Objekte. Daher müssten die Daten ins XML-Format konvertiert werden. Es existieren bereits Werkzeuge (s. [Gro05]), die diese Aufgabe übernehmen. Cocoon ist darüber hinaus in der Lage, aus Java-Objekten XML-Code zu erzeugen, weshalb vorerst die XML-Transformation auf die Seite von Cocoon verschoben werden.

Universelle Klasse

Da das Jena Model bereits alle Ressourcen als Objekte behandelt, ist eine Abbildung in den objektorientierten Raum, wie mit den Factory-Klassen, nicht zwingend notwendig. Aus diesem Grund wurde ein Konzept entwickelt, welches eine universelle Klasse als Schnittstelle zwischen Jena-Modell und Präsentationsschicht verwendet. Diese Klasse wird in der Abbildung 5.6 als *Entity* bezeichnet.

Entity ist in der Lage, eine Suchanfrage für eine Ressource und deren Eigenschaften an das Jena-Model weiterzuleiten und die erhaltenen Daten in einer Hashtabelle abzuspeichern. Dieses Konzept bietet den Vorteil, dass die Präsentationsschicht die erhaltenen Daten mit Hilfe der Standard Java-API verarbeiten kann. Ein Zugriff auf die Jena-API von dieser Stelle muss nicht erfolgen. Die Entscheidung für eine solche universelle Schnittstelle wird damit begründet, dass für den Prototypen keine umfangreiche Ontologie mit vielen verschiedenen Konzepten entwickelt wurde. Aus diesem Grund wurde ein universelles und abstraktes Konzept gewählt. Die Idee wurde bereits erfolgreich für die Semantik Web Seite *Alliknow* eingesetzt.

Für eine umfangreiche Ontologie erscheint die Abbildung der einzelnen Ressourcen in den objektorientierten Raum mittels des oben beschriebenen Factory-Konzepts sinnvoller, da dadurch direkte Kenntnis über den entsprechenden Ressourcetyp gewonnen werden kann. Mittels der Entity-Klasse ist dies nicht möglich. Da jedoch zurzeit lediglich drei Hauptkonzepte existieren, ist i.d.R. bekannt, um was für eine Ressourcetyp es sich handelt, weshalb dieses Kriterium nicht weiter beachtet wird.

5.3.2 SemantikUB-Klassenbeschreibung

Im Folgenden werden die einzelnen Klassen im Detail beschrieben. Dabei wird für jede Klasse in Form eines Klassenkopfes aufgezeigt, welche Methoden diese anbietet.

Klasse SingletonModel

```
public class SingletonModel {
  protected static ModelRDB model;
  protected static ModelRDB infModel;
  protected SingletonModel()
  public static SingletonModel getInstance()
  public void writeModel()
  public Model getDefaultModel(){
  public InfModel getInfModel()
  public LinkedList getUpperSubject(Resource subject)
  public LinkedList getSubClasses(String subject)
}
```

Die Klasse *SingletonModel* repräsentiert den eigentlichen Kern von SemantikUB. Mit Hilfe dieser Klasse kann auf die beiden Modelle *model* und *infModel* zugegriffen werden. Diese haben folgende Funktionalität:

- Die Variable *model* repräsentiert das normale Jena-Model. Die einzelnen Statements aus denen das Modell besteht, sind in einer MySQL-Datenbank namens *jenamodel* abgespeichert. Änderungen in der Ontologie werden in diesem Modell erfasst und sofort in die Datenbank geschrieben. Nach jeder Änderung wird das *infModel* synchronisiert. Dies geschieht durch Aufruf der Funktion *writeModel*. Durch die Funktion *createModel* kann die Datenbank initialisiert werden.
- Die Variable *infModel* steht für das Modell, welches mit logischen Schlussfolgerungen erweitert wurde. Das Model besteht damit aus des Jena-Modell und zusätzlichen Statements, welche mit Hilfe von mehrerer Reasoner erzeugt wurden. Das *infModel* kann als zweite Sicht auf die Wissensbasis angesehen werden. Es enthält beispielsweise Vereinigungsmengen oder integriert die transitiven Eigenschaften. Je nach Bedarf können beide Modelle abgefragt werden. Die Datenbank heißt *infmodel*.

Da es nicht notwendig ist, für jeden Zugriff – möglicherweise von verschiedenen Objekten – auf das Jena-Modell, immer wieder neue Instanzen von *SingletonModel* zu erzeugen, wurde diese Klasse nach dem Singelton Design Pattern [Gea03] entworfen. Dieses Design Pattern bietet die Möglichkeit an, genau ein Objekt der entsprechenden Klasse während der gesamten Projektlaufzeit zu erzeugen. Es enthält lediglich einen *protected*-Konstruktor, weshalb dieser von Objekten anderen Typs nicht verwendet werden kann. Das Erzeugen des *SingletonModel*-Objektes geschieht beim Aufruf der Funktion *getInstance()*.

Im folgenden Beispiel 5.6 wird die Funktion *getInstance()* beschrieben. Diese hat die Aufgabe zuerst ein *SingletonModel*-Objekt zu erzeugen. Danach werden die Variablen *model* (s. Zeilen 12 – 15) und *infModel* (s. Zeilen 18 – 21) mit den dazu passenden Datenbank zu verbunden. Dadurch haben diese vollen Zugriffe auf alle Statements und damit allen Ressource der jeweiligen Datenbank.

```
1  /**
2     * Funktion erzeugt eine Singleton Modell Instanz. Beim Erzeugen,
3          * werden auch die Datenbankverbindungen zu den beiden Modells aufgebaut
4  */
5  public static SingletonModel getInstance() {
6
7     try {
8        //Wird nur einmal ausgeführt!
9        if (instance == null) {
10          //verwenden des Konstruktors
11          instance = new SingletonModel();
12
13          //Datenbanktreiber      laden
14          Class.forName ("com.mysql.jdbc.Driver");
15
16          //Angabe des Datenbanknamens -- für das Jena-Default Model
17          String DB_URL = "jdbc:mysql://localhost/jenamodel";
18          IDBConnection conn = new DBConnection (DB_URL, "User", "
               Passwort", "MySQL" );
19          //Datenbank öffen -- model hat jetzt Zugriff auf die Datenbank jenamodel
20          model = ModelRDB.open(conn);
21
22          //Einlesen des Dublin Cores. Dieser wird der lokalen Ontologie hinzugefügt.
23          //Da der Dublin Core bereits unter einer URL erreichbar muss er nicht in
24          //der lokalen Ontologie gespeichert werden.
25          model.read("http://purl.org/dc/elements/1.1/")
26
27          //Angabe des Datenbanknamens -- für das InfModel Model
28          DB_URL = "jdbc:mysql://localhost/infmodel";
29          conn = new DBConnection (DB_URL, DB_USER, DB_PASSWD, DB );
30          //Datenbank öffen -- infModel hat jetzt Zugriff auf die Datenbank infmodel
31          infModel = ModelRDB.open(conn);
32    }
33       // Rückgabe des Singleton Objekts
34       return instance;
35
36    } catch(ClassNotFoundException e){
37       System.out.println("ClassNotFound: " + e.getMessage());
38    }
39 }
```

Beispiel 5.6: Modell erzeugen

Wie im obigen Beispiel gezeigt, wird innerhalb des Prototypen noch mit verschiedenen Modellen gearbeitet. Dies sind das Jena-Standard-Modell, das *InfModel* sowie ein Modell welches das *InfModel* in einer Datenbank abspeichert. In der nächsten Version wird

voraussichtlich noch das Datenbankmodell verwendet werden. Zurzeit ist dies aus Testgründen nicht gewünscht.

Klasse Entity

```
public class Entity {
  private HashMap hStmt = new HashMap();
  private String englishLabel;
  private String germanLabel;
  public Entity(String sURI, LinkedList sProperty )
  public Entity(String sURI)
  public String getGermanLabel()
  public String getEnglishLabel()
  public LinkedList getCategories()
  public LinkedList getValuesForOneCategory(String
     category)
}
```

Die Klasse *Entity* hat die Aufgabe, die vom Jena-Model zurückgelieferten Daten in einem einheitlichen Format zu verarbeiten. Diese wird von der Klasse *RDQL-Search* dazu verwendet, die erhaltenen Treffer einheitlich darzustellen. Die Entity-Klasse speichert für genau eine Ressource alle oder eine gewünschte Liste an Statements in einer Hashtabelle ab. Da es möglich ist, dass für eine Eigenschaft mehrere Objekte zurückgeliefert werden können (s. RDQL in 5.1.1), wird innerhalb der Hashtabelle, die Eigenschaft als Schlüssel abgelegt. Die dazugehörigen Objekte, werden als Wert in Form einer Liste zum passenden Schlüssel gespeichert. Die Hashtable hat den folgenden Aufbau:

Schlüssel	*Wert*
Property 1	$List(1 \cdots n)$
Property 2	$List(1 \cdots n)$
Property n	$List(1 \cdots n)$

Innerhalb jedes *Entity*-Objekts existieren drei globale Variablen, welche einerseits die deutsche und englische Bezeichnung (*rdfs:label*) der Ressource sowie den Ressourcenamen (*rdf:ID*) enthalten. Da es sich hierbei um Eigenschaften handelt, die jede Ressource hat, werden diese nicht nur in der Hashtabelle, sondern auch direkt in einfach zugänglichen Variablen abgespeichert.

Ein *Entity*-Objekt kann direkt mit Hilfe des Konstruktors erzeugt werden. Möchte man beispielsweise auf die *rdfs:label*-Eigenschaft einer Dublin Core Eigenschaft zugreifen, kann ein *Entity*-Objekt über diese Ressource erzeugt werden. Dazu muss die vollständige URI angegeben werden (s. Methode 1 im Beispiel 5.7).

Die zweite Möglichkeit zum Erzeugen eines *Entity*-Objekts, wird i.d.R. von der RDQL-Suche benutzt, um nur bestimmte Statements im *Entity*-Objekt abzulegen. Die Methode 2 im folgenden Beispiel 5.7 verdeutlicht dies.

```
1  // Methode 1: direkte Erzeugung des Ressourceninhalts
2  Entity ent = new Entity("http://purl.org/dc/elements/1.1/title");
3  // hole deutsches Label
4  String sLabel = ent.getGermanLabel();
5
6  // Methode 2: Erzeugung mehrerer Entity Objekte durch RDQL-Suche.
7  // Gesuchte Eigenschaften in Liste abspeichern.
8  LinkedList listProp = new LinkedList();
9  listProp.add("http://purl.org/dc/elements/1.1/title");
10 listProp.add("http://purl.org/dc/elements/1.1/subject");
11 listProp.add("http://purl.org/dc/elements/1.1/publisher");
12 listProp.add("http://purl.org/dc/elements/1.1/contributor");
13 listProp.add("http://purl.org/dc/elements/1.1/format");
14
15 //Es soll für diese Ressource die Statments -- abgespeichert in listProp -- geholt
       werden
16 String sRessourceName = "http://www.ub.uni-heidelberg.de#
       zeitschrift10"
17 //Erzeuge Entity
18 Entity ent1 = new Entity(sRessourceName,listProp);
```

Beispiel 5.7: Entity-Objekt erzeugen

Die Entscheidung, eine allgemeine Darstellung der Ressourcen zu verwenden, bietet den Vorteil, dass die Pflege für mehrere Klassen entfällt. Eine Funktion, welche den Inhalt jedes *Entity*-Objektes nach XML exportiert, wurde in die ToDo-Liste aufgenommen, wird jedoch innerhalb der Masterthesis nicht mehr realisiert werden können.

Klasse Statement-Search

```
public class StatementSearch{
  public static void printStatements(Resource s,
     Property p, Resource o)
  public static List getStatements(Resource s,
     Property p, Resource o)
  public static LinkedList getObjects(Resource s,
     Property p)
  public static HashSet getProperties(Resource s)
}
```

Die Klasse *StatmentSearch* hat die Aufgabe, Statements aus dem Jena Model für eine Ressource anzufordern. Dazu existiert die Funktion *printStatements*, welche lediglich eine Ausgabe erzeugt und die Funktion *getStatements*, welche eine Statement-Liste zurückliefert. Beide Funktionen können ein Subjekt als *Resource*, eine Eigenschaft als *Property* und ein Objekt als *Resource* übergeben bekommen. Dabei bleibt es offen, welche der Übergabeparameter leer bleiben. Kein Übergabeparameter würde dazu führen, dass alle Statements des Modells ausgegeben werden. Wird ausschließlich ein Subjekt angegeben, werden alle Statements von genau diesem Subjekt angezeigt. Würde hingegen nur eine Ressource als Objekt an die Funktion übergeben, dann erhält man alle Statements, die als Objekt die entsprechende Ressource aufweisen können.

Die Funktion *getObjects* liefert für eine Eigenschaft einer Ressource alle Objekte zurück. Dabei muss zwischen einfachen Literalen und Ressourcen unterschieden werden. Der Rückgabewert der Funktion ist eine Liste von Strings welche die Objektnamen repräsentieren. Die Funktion dient als Hilfsfunktion zum Erzeugen von *Entity*-Objekten.

Die Funktion *getProperties* liefert eine Liste aller Eigenschaften einer Ressource zurück. Dazu werden einfach alle Statements, die eine Ressource aufweist, in ein HashSet geschrieben. Innerhalb des HashSet kann der gleiche Werte niemals doppelt vorkommen, weshalb garantiert ist, dass jede Eigenschaft einmalig im HashSet abgelegt wird. Die Funktion dient als Hilfsfunktion zum Erzeugen von *Entity*-Objekten.

Klasse RDQL-Search

```
public class RdqlSearch {
   public static LinkedList subjectSearch(String
      subject, String publisher, String title, String
      standort)
   public static LinkedList subSubjectSearch(String
      subject)
}
```

Die Klasse RDQL-Search, verwendet zur Abfrage des Jena-Modells die SQL-ähnliche Sprache RDQL. Diese hat den Zweck, nicht nur einzelne Statements zu suchen. Stattdessen können komplexere Suchanfragen über mehrere Statements und sogar mehrere Domänen hinweg aufgebaut werden. Die Funktion *subjectSearch* hat die Aufgabe, nach Medien zu suchen. Dabei kann anhand der an die Funktion übergebenen Parameter *subject*, *publisher*, *standort* und *title* die Suche eingeschränkt werden. Die RDQL-Abfrage lautet folgendermaßen:

```
SELECT ?fname
WHERE (?fname, rdf:type, <http://www.ub.uni-heidelberg.de#Medium>),
      (?fname, dc:subject ,?sub),
      (?fname, dc:publisher ,?pub),
      (?fname, dc:title ,?tit),
      (?fname, ub:hatVerfuegbarkeit,?stand)
AND   ?sub =~ /"+subject+"/i,
      ?pub =~ /"+publisher+"/i,
      ?tit =~ /"+title+"/i,
      ?stand =~ /"+standort+"/i
USING dc
FOR   <http://purl.org/dc/elements/1.1/>
      ub
FOR   <http://www.ub.uni-heidelberg.de#>
```

Beispiel 5.8: RDQL-Abfrage

Diese RDQL-Abfrage hat die Aufgabe, alle Ressourcen vom Typ Medium zu suchen. Die an die Funktion mit übergebenen Variablen *subject*, *publisher*, *standort* und *title* können leer sein.

Weiterhin werden Reguläre Ausdrücke verwendet. Die Reguläre Ausdrücke werden mit Hilfe des Jakarta ORO-Pakets [Sav04] als Perl-Syntax in die RDQL-Abfrage integriert. Die Zeichenfolge = steht dafür, dass der Text enthalten sein muss, wobei Groß- und Kleinschreibung nicht beachtet wird (Parameter: /i).

Die von dieser Abfrage erhaltenen Werte, stellen die Ressourcenamen der gefundenen Medien dar, welche im nächsten Schritt, als Entity-Objekte mit bestimmten Eigenschaften (s. Methode 2 im Beispiel 5.7) an die aufrufenden Funktion zurückgegeben werden.

Die Funktion *subSubjectSearch* findet auch Medien welche in einer Unterkategorie des angegebenen Fachs enthalten sind.

5.3.3 Integration von SemantikUB nach Cocoon

Zur Bedienung von SemantikUB wurde Cocoon verwendet. Diese ist als XML-orientiertes Web-Publishing-Framework sehr gut dazu geeignet, die Business Logik von SemantikUB zu steuern und die erhaltenen Daten zu präsentieren. Eine ausführliche Beschreibung zu Cocoon findet sich in der Seminararbeit [Mil04].

Suche

Im Prototypen von SemantikUB wird die Möglichkeit angeboten, eine Suche nach Medien durchzuführen. Dabei besteht die Möglichkeit, sich nur bestimmte Medien mit Hilfe von Filtern anzeigen zu lassen. Die einzelnen Medien werden von Cocoon in einer HTML-Tabelle ausgegeben. Dazu wurde die Business Logik der SemantikUB direkt in Cocoon integriert. Nach erstmaligem Aufruf der Business Logik durch Cocoon, werden die Klassen verarbeitet und bleiben danach im Speicher bestehen. Dadurch kann eine sehr schnelle Suche innerhalb der Wissensbasis erfolgen.

Um innerhalb der Wissensbasis von SemantikUB zu suchen, wird eine RDQL-Suche (s. Klasse RDQL-Search) durchgeführt. Um die entsprechenden Klassen von SemantikUB in Cocoon zu integrieren, wird ein XSP-Dokument verwendet. Diese enthält einen Logik-Teil in welchem Java-Befehle verarbeitet werden können. Das XSP-Dokument wird nach der Verarbeitung von Cocoon in eine Java-Server-Page umgewandelt. Dieses Java-Dokument kann innerhalb des “work“-Verzeichnisses im Tomcat-Ordner eingesehen werden.

Das folgende Beispiel 5.9 zeigt einen Ausschnitt aus der XSP-Seite, welche eine RDQL-Suche durchführt. Dabei wurde eine etwas vereinfachte Darstellung gewählt. Zum durchführen der RDQL-Suche werden die mit Jena entwickelten und im Abschnitt 5.3.2 beschriebenen Klassen verwendet. Das XSP-Dokument hat die Aufgabe, die von der RDQL-Suche erhaltenen Daten nach XML zu konvertieren.

```
1 <xsp:logic>
2   // Stelle RDQL-Suche mit Parametern (können optional leer bleiben!
3   LinkedList list = rdql.subjectSearch( sSubjekt, sVerlag, sTitel,
      sStandort);
```

```
4   //Diese Liste muss jetzt nach XML verarbeitet werden
5   for (int i = 0; i &lt; list.size(); ++i) {
6           //hole Entity Objekt aus der Entity-Liste
7     Entity e = (Entity) list.get(i);
8     //XML-Tag für jedes gefundene Medium
9     <rdfRessource>
10      //neuer logic-Abschnitt -- Syntax notwendig -- da sonst Fehlermeldung
11      <xsp:logic>
12          //Gehe über Eigenschaften der Entity -- gespeichert in lProperty
13          for (int j = 0; j &lt; lProperty.size(); ++j) {
14          //Hole Objekte für Eigenschaft -- das können mehrere sein!
15                //Ablegen des Wertes
16          <xsp:content>
17             <xsp:element>
18                   <xsp:param name="name">
19                     //Weiteres XML-Tag mit dem Eigenschaftsnamen des Medium
20                     //Tagbezeichnung wird dynamisch durchgeführt
21                     <xsp:expr>entity.getName().toString()</xsp:expr>
22                   </xsp:param>
23                //Speichere den Wert zur Eigenschaft ab
24                <xsp:expr>entity.getValues().toString()</xsp:expr>
25             </xsp:element>
26          </xsp:content>
27          }
28      </xsp:logic>
29    </rdfwert>
30  }
31 </xsp:logic>
```

Beispiel 5.9: XSP

In Zeile 3 wird die eigentliche Suchanfrage an SemantikUB gestellt. Die Instanz *rdql* ist vom Typ *Rdql-Search*. Die Parameter können dabei leere Strings enthalten, wodurch alle Treffer zurückgeliefert werden würden. Der Einsatz von Regular Expressions ist möglich. Als Rückgabewert liefert die RDQL-Suche eine Liste mit Entity-Objekten. Jedes Entity-Objekt stellt dabei ein durch die RDQL-Suche gefundenes Medium dar.

Das Framework Cocoon besteht aus XML-Konzepten. Aus diesem Grund muss im nächsten Schritt jedes Entity-Objekt nach XML konvertiert werden. Dieser erzeugte XML-Baum wird mit Hilfe eines XSL-Templates in eine Webseite umgewandelt.

Um den XML-Baum zu erzeugen, wird in Zeile 8 ein Tag namens *<rdfRessource>* gesetzt. Für jedes gefundene Medium wird ein solches Tag erzeugt. In den Zeilen 15 – 20 wird für jede Eigenschaft des Mediums ein Tag erzeugt, das dem Namen der Eigenschaft angibt.

Dadurch wird ein Konzept verwendet, welches mit dynamischen Tagbezeichnungen arbeitet, da man niemals im Voraus genau weiß, wie die entsprechende Eigenschaft heißen wird. Um mit dynamischen Tagbezeichnungen zu arbeiten, müssen die zusätzlichen xsp-Attribute *xsp:content, xsp:element* und *xsp:param* angegeben werden. In Zeile 19 wird der eigentliche Tagname abgelegt. Zuletzt wird der Wert des Tag in Zeile 22 gespeichert.

Es ist möglich, dass für eine Eigenschaft mehrere Werte zurückgeliefert werden. Aus diesem Grund müsste im Beispiel 5.9 mit einer weiteren *For*-Schleife gearbeitet werden, um

alle Werte im XML-Baum zu integrieren. Da dieses Beispiel nur die allgemeine Funktionsweise zur Verarbeitung von Entity-Objekten aufzeigt, wird das Beispiel ohne diese *For*-Schleife dargestellt.

Dieses XSP-Dokument erzeugt einen XML-Baum, welcher Beispielsweise folgendes Aussehen haben könnte.

```
<rdfRessource>
  <dc:title>Das Java Magazin</dc:title>
  <dc:subject>ub:Java</dc:subject>
  <dc:subject>ub:Informatik</dc:subject>
  <dc:publisher>Der Java Verlag</dc:publisher>
</rdfRessource>
<rdfRessource>
  <dc:title>Das Computer Magazin</dc:title>
  <dc:subject>ub:Informatik</dc:subject>
  ....
</rdfRessource>
....
```

Im nächsten Schritt wird der XML-Baum durch eine XSLT-Datei verarbeitet und als Webseite mit einer Tabelle ausgegeben. Dazu wurde ein Template definiert, welches in der Lage ist, die obigen Attribute zu verarbeiten.

Da eine RDQL-Suche ausschließlich Medien zurückliefert, sind auch die einzelne Eigenschaften jeder Ressource bekannt und können im XSL-Template verarbeitet werden. Ein kleiner Ausschnitt aus der XSLT-Datei wird im Beispiel 5.10 gegeben.

```
<table>
  <xsl:apply-templates select="/page/rdfRessource"/>
</table>
....
<xsl:template match="/page/rdfRessource">
  <tr align="left" valign="top">
    <td>
      <xsl:for-each select="dc:title">
        <table>
          <tr valign="top">
            <xsl:value-of select="."/>
          </tr>
        </table>
      </xsl:for-each>
    </td>
    ....
    <!--Dieser Schritt wird für alle zurückgelieferten Eigenschaften einer Ressource
        wiederholt. Dadurch entstehen die Werte für eine Zeile -->
  <tr>
```

Beispiel 5.10: XSLT

Der Inhalt jeder Ressource wird innerhalb eines *rdfRessource*-Tag abgelegt. Aus diesem Grund wird ein Template erzeugt, welches für jedes *rdfRessource*-Tag aufgerufen wird.

Das Template beginnt in Zeile 5. Nach jedem Aufruf des Templates wird eine neue Tabellenzeile erzeugt (s. Zeile 6). Im nächsten Schritt wird das Tag namens *<dc:title>* gesucht. Da pro Eigenschaft mehrere Werte zurückgeliefert werden können, ist es notwendig einerseits mit einer *xsl:for-each*-Schleife zu arbeiten, andererseits innerhalb der eigentlichen Tabelle eine weitere Tabelle zu erzeugen. Diese enthält den Inhalt jeder Eigenschaft. Dieser beschriebene Prozess wird für jede Eigenschaft durchgeführt.

Internationalisierung

Das RDF-Konzept bietet gute Möglichkeiten zur Internationalisierung an, weshalb diese in den Prototyp integriert wurde. Wie bereits erwähnt, besitzt jede Ressource ein so genanntes *rdfs:label*-Attribut, welches die Aufgabe hat die Bezeichnung einer Ressource anzugeben. In Abhängigkeit von *xml:lang="de"* kann die Bezeichnung beispielsweise in deutscher Sprache hinterlegt werden. Wie bereits in der Klassenbeschreibung von Entity angegeben, werden die englische und deutsche Beschreibung, in jedem Entity-Objekt als einfach zugängliche Variablen hinterlegt.

Da jedes Medium fast ausschließlich Eigenschaften des Dublin Cores zum Beschreiben verwendet, wird der Tabellenkopf durch die Label-Angaben, der jeweiligen Dublin Core Eigenschaft, aufgebaut. Der Offizielle Dublin Core enthält ausschließlich englische Label-Beschreibung. Wie bereits erwähnt, wurde aus diesem Grund, die SemantikUB-Ontologie mit zusätzlichen Dublin Core Informationen ausgestattet, welche nur in der lokalen Ontologie vorhanden sind. Durch diese Konzept kann der Tabellenkopf internationalisiert werden. Dazu wird bei jeder Anfrage der Ländercode des Browsers ausgelesen und in der so genannten *locale*-Variablen abgelegt. In Abhängigkeit vom Inhalt dieser Variablen folgt danach eine dynamische Tabellenbeschriftung. Das folgende Beispiel 5.11 verdeutlicht dieses Verfahren. Dazu wird wiederum XSP und Java verwendet.

```
1  Entity eDC = new Entity(myModel.getDefaultModel(), "http://purl.org
       /dc/elements/1.1/title");
2  if (sLocale.startsWith("de")){
3    <xsp:content>
4      <xsp:element>
5        <xsp:param name="name">dctitle</xsp:param>
6          <xsp:expr> eDC.getGermanLabel()</xsp:expr>
7      </xsp:element>
8    </xsp:content>
9  }
10 else{
11   <xsp:content>
12     <xsp:element>
13       <xsp:param name="name">dctitle</xsp:param>
14       <xsp:expr> eDC.getEnglishLabel()</xsp:expr>
15     </xsp:element>
16   </xsp:content>
17 }
```

Beispiel 5.11: Internationalisierung

In Zeile 1 wird ein Entity-Objekt für die Ressource *dc:title* erzeugt. Innerhalb dieses Entity-Objektes kann auf die deutsche und englische Bezeichnung zugegriffen werden. Die Variable *sLocale* in Zeile2 ist vom Typ String und wird mit dem Inhalt der Ländereinstellung an die XSP-Seite übergeben. Je nach Inhalt wird entweder die deutsche oder englische Bezeichnung als Tag im XML-Baum abgespeichert.

Innerhalb der XSLT-Seite wurde ein Template definiert, welches den Tabellenkopf anhand der im XSP-Dokument definierten Tags aufbaut. Die Resultate können in der Abbildung 6.2 im Kapitel 6 eingesehen werden.

Kapitel 6

SemantikUB im Einsatz

Das folgende Kapitel beschreibt, wie SemantikUB eingesetzt werden kann. Dabei wird anhand von einigen Abbildungen ein ersten Eindruck gegeben. Die im Lastenheft geforderte Funktionen werden auf Vollständigkeit untersucht.

6.1 Ontologie

Die entwickelte Ontologie wurde bereits ausführlich in 5.2 beschrieben. Protégé konnte bei der Modellierung sehr hilfreiche Dienste leisten. Erweiterungen der Ontologie sind jederzeit möglich. Der im Lastenheft geforderte Punkt 4.3.1 konnte damit vollständig umgesetzt werden.

6.2 Datenimport

Der Datenimport konnte ausschließlich für Medien vom Typ elektronische Zeitschriften erfolgen (s. 5.2). In der Ontologie befinden sich zurzeit fast 3000 elektronische Zeitschriften, wobei die Metadaten der meisten Zeitschriften unvollständig sind. Dies liegt daran, dass die Datenquelle für den Import nicht alle Informationen bereitstellen konnte. Des Weiteren gab es bei der verwendeten Datenquelle Probleme mit Sonderzeichen, was durch die Inkompatibilität zwischen ISO-8859 und UTF-8 entstanden ist.

Aus diesem Grund wurden mit Protégé die importierten Instanzen soweit notwendig vervollständigt. Die fehlerhaften Sonderzeichen wurden nicht ausgebessert, da dies für die Entwicklung des Prototypen nicht zwingend notwendig war. Des Weiteren wurden weitere Instanzen anderer Konzepte mit Protégé erzeugt. Damit konnte die im Lastenheft unter Punkt 4.3.2 gestellte Anforderung teilweise umgesetzt werden.

6.3 Datenpflege

Protégé ist ein sehr komplexes Werkzeug, welches viel Vorwissen benötigt. Es ist möglich, dass Mitarbeiter mit Protégé die Daten pflegen und erweitern. Die folgende Abbildung zeigt die Eingabemaske von Protégé, welche zur Pflege der Instanzen verwendet werden kann.

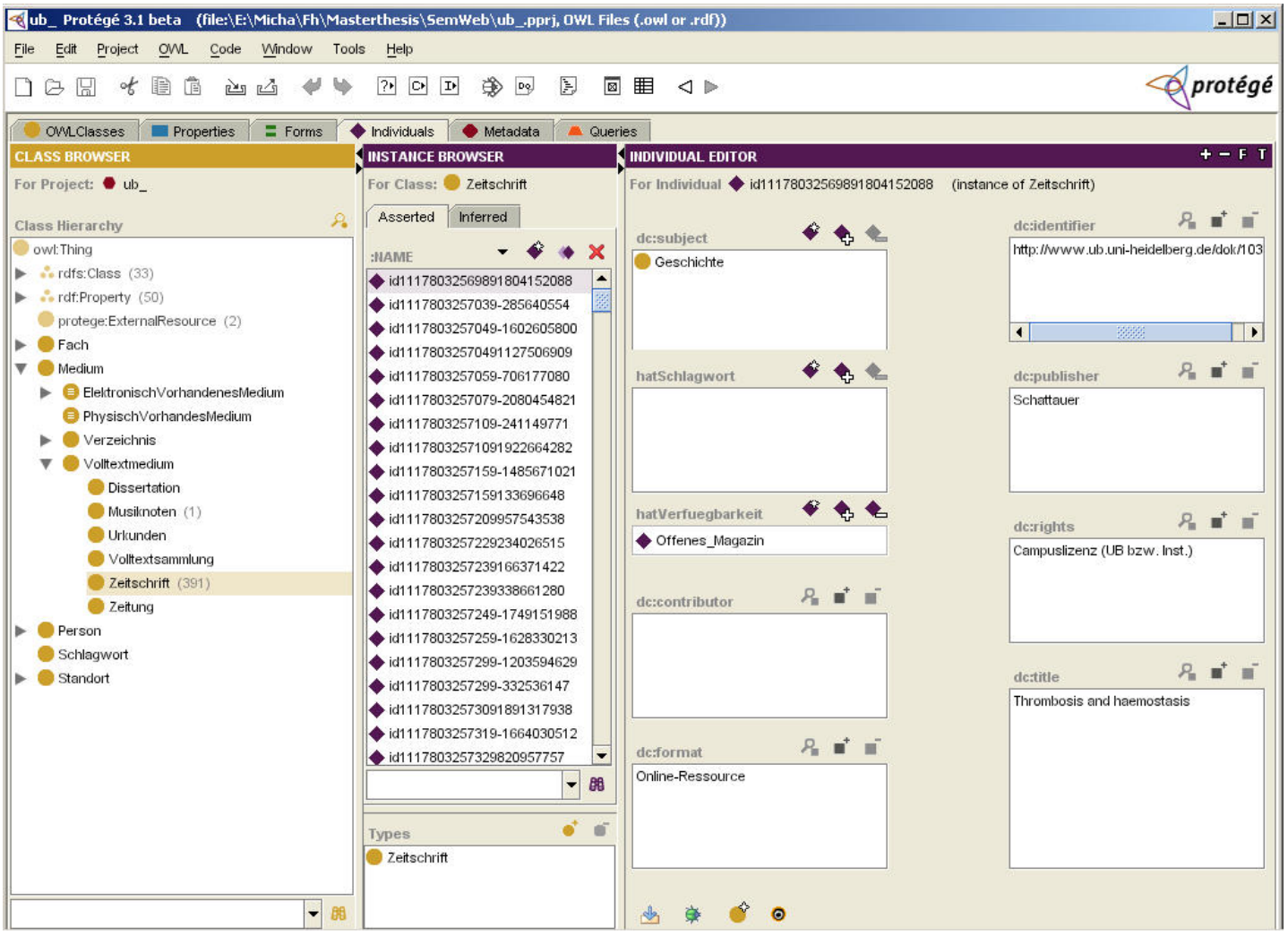


Abbildung 6.1: Datenpflege mit Protégé

Das linke Fenster zeigt die einzelnen Konzepte von denen sich Instanzen erzeugen lassen. In der Abbildung 6.1 wurde das Konzept *Zeitschrift* ausgewählt und ist entsprechend markiert.

Das mittlere Fenster zeigt die einzelnen Instanznamen des ausgewählten Konzeptes an. Die Namen sichern die Einmaligkeit einer Ressource innerhalb der Ontologie. Da diese Darstellung sehr unübersichtlich ist, existiert eine gute Suchfunktion. Zum Benutzen

der Suche sind grundlegende Kenntnisse in RDF notwendig. Es wurde die erste Instanz innerhalb der Liste ausgewählt und entsprechend markiert.

Das rechte Fenster der Abbildung zeigt einzelne Eingabefelder zum Beschreiben der ausgewählten Instanz. Dabei existieren verschiedene Felder, z.B. vom Typ Literal oder Ressource. Wird beispielsweise das erste Eingabefeld *dc:subject* ausgewählt, erscheint ein Auswahlfenster, in welchem alle Fächer angezeigt werden. Der Grund dafür liegt darin, dass die Eigenschaft *dc:subject* vom Typ Objekt-Eigenschaft ist und deshalb auf Ressourcen zeigen muss. Die Auswahl des Feldes *dc:title* hingegen ist vom Typ Literal. Daher kann dort ausschließlich Text eingegeben werden.

Die mit Protégé eingegeben Daten müssen per RDF-Export in eine XML-Datei geschrieben werden. Im nächsten Schritt wird diese Datei durch die Jena-API in die Datenbank integriert. Dadurch ist die direkte Bearbeitung der Datenbank nicht möglich. Der Import-Vorgang ist im Augenblick noch nicht automatisiert.

Die Fähigkeiten von Protégé gehen jedoch weit über die Pflege der Daten hinaus. Aus diesem Grund hätte ein Mitarbeiter immer Zugriff auf die direkten Konzepte der Ontologie (s. Abbildung 5.1 im Kapitel 5) und könnte diese nach belieben verändern.

Aus diesen Gründen ist die Datenpflege mit Protégé nicht sinnvoll und benutzerfreundlich. Diese kann als Übergangslösung angesehen werden, ist jedoch für die Anforderungen des SemantikUB Prototyps, durchaus ausreichend.

Für den produktiven Einsatz von SemantikUB könnte eine passwortgeschützte Eingabemaske im Web eingerichtet werden. Autorisierte Mitarbeiter bekämen über diesen Bereich die Möglichkeit, die Daten direkt in der Datenbank zu bearbeiten. Diese Funktion könnte mit Cocoon und der Jena-API umgesetzt werden. Die Jena-API bietet zum Manipulieren der Ressourcen alle gewünschten Funktionen an.

Ein besonderer Teil der Datenpflege stellt die Möglichkeit dar, neues Wissen auf Grundlage des bereits Vorhandenen zu erzeugen. Dazu wurde SemantikUB mit logischen Regeln erweitert, welche im Abschnitt 5.2.2 beschrieben sind. Durch Protégé ist es möglich, neue Fächer zu erzeugen, welche jedes Medium zugewiesen bekommt, das bereits bestimmte Fächer aufweisen kann (s. Mechatronik Beispiel). Dieser Teil der Datenpflege, stellt einen komplexen Vorgang dar, welcher sehr gut mit Protégé umgesetzt werden kann. Werden logische Konstrukte erstellt, geschieht dies durch direkte Bearbeitung der Ontologie. Aus diesem Grund kann auch in zukünftigen Versionen nicht vollständig auf Protégé und den damit verbundenen Ex- und Importvorgang verzichtet werden. Möglicherweise wird jedoch in naher Zukunft eine Schnittstelle zwischen Protégé und verschiedenen Datenbanken angeboten, so dass eine direkte Bearbeitung der in der Datenbank gespeicherten Ontologie vorgenommen werden kann.

6.4 Datenbankanbindung

Die Datenbankanbindung – gefordert im Lastenheft unter Punkt 4.3.3 – funktioniert hervorragend. Es wurden zwei Modelle in der Datenbank gespeichert (s. 5.3.2). Jena erzeugt

dabei vollautomatisch die notwendigen Tabellen und legt alle Statements in einer Tabelle mit drei Spalten (Subjekt, Prädikat, Objekt) als URIs ab.

Die Semantische Suche auf Grundlage einer Datenbank ist sehr effizient. Die gemessene Zeit, die benötigt wird, um die vollständige Datenbank (ca. 3000 Medien) zu durchsuchen, dauert i.d.R. nur wenige Sekunden. Genaue Benchmarks wurden nicht vorgenommen.

Der vollständige Import aller Daten in die Datenbank ist eine zeitaufwändige Aktion. Das Jena-Default-Modell, ist sehr schnell aufgebaut und besteht aus ca. 20.000 Statements. Im nächsten Schritt müssen jedoch anhand dieses Modells alle logischen Schlussfolgerungen gezogen werden um das *infModel* aufzubauen. Für den Aufbau des *infModel* ist es weiterhin notwendig, mit zwei verschiedenen Reasonern und damit zwei Modellen zu arbeiten. Der Aufbau dieser Modelle dauerte über zwei Stunden. Die Tabelle enthielt nach Abschluss über 100.000 Statements.

6.5 Medienrecherche

Das wichtigste Kriterium von SemantikUB ist es, eine leistungsfähige Medienrecherche zu ermöglichen. Aus diesem Grund musste diese Funktion vollständig implementiert werden.

Die eigentliche Medienrecherche geschieht über eine RDQL-Suche (s. 5.3.2). Diese RDQL-Suche kann mit Hilfe von Cocoon und einem Webbrowser gesteuert werden. Das Ergebnis wird auf einer von Cocoon generierten Webseite, in Tabellenform, angezeigt.

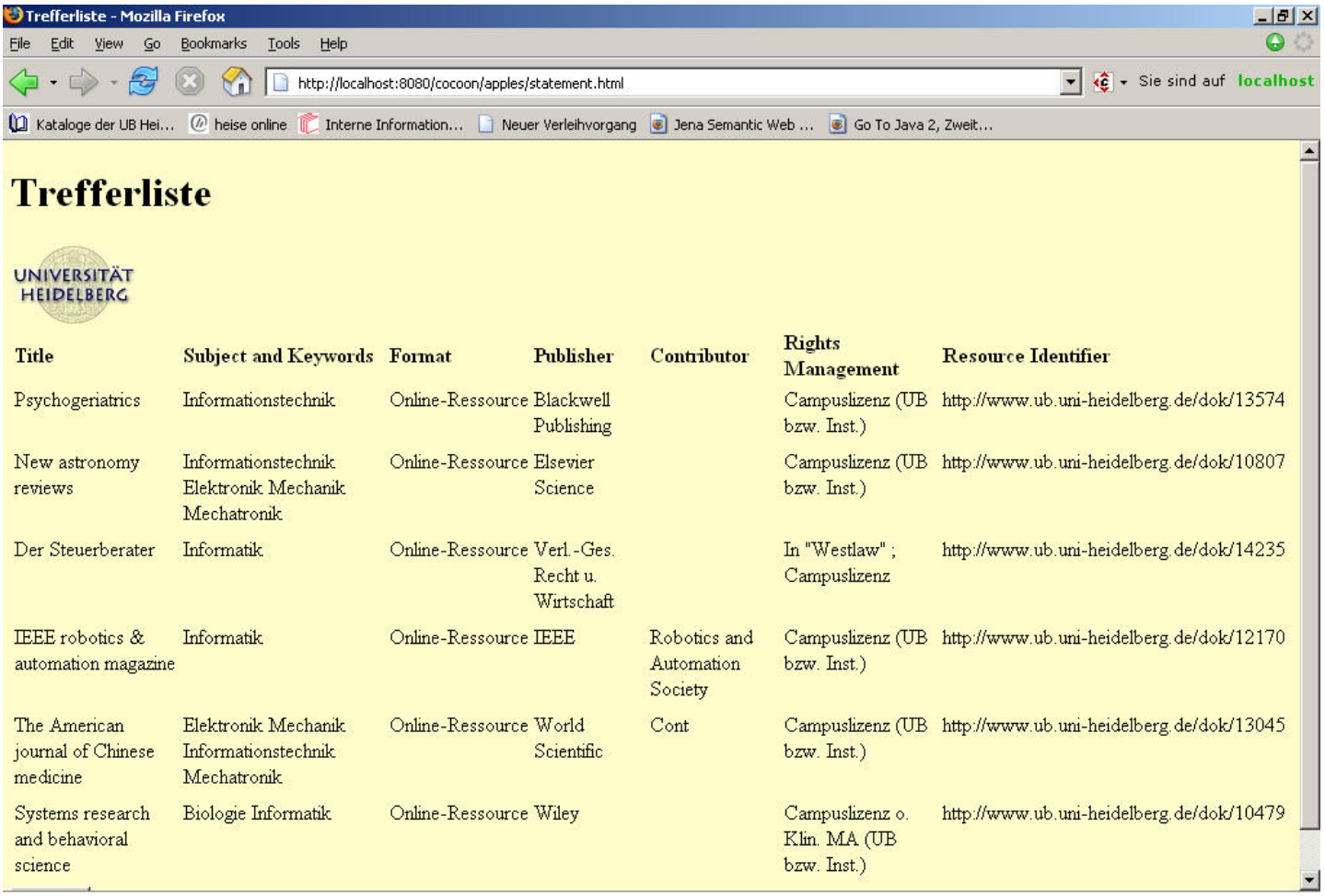
Die Suchfunktion bietet die Möglichkeit an, nach Medien anhand der Kriterien Titel, Fach, Verlag und Standort zu suchen. Dabei ist es möglich, ein oder mehrere Felder miteinander zu kombinieren. Falls alle Felder leer bleiben, erhält der Benutzer alle vorhandenen Medien zurück. Eine Suche mit Hilfe von Regulären Ausdrücken ist möglich. Diese Option ist automatisch aktiviert, d.h. es müssen keine Metazeichen wie der Stern (*) eingegeben werden, um einen Regulären Ausdruck zu kennzeichnen.

Eine weitere Funktion, um nach Medien zu suchen, besteht darin, alle Medien eines Fachs zu finden. Die Besonderheit dieser Funktion ist es, dass das Ergebnis nicht nur alle Medien beinhaltet, welche dem entsprechenden Fach zugeordnet sind, sondern auch alle Medien, welche in einem untergeordnetem Gebiet diese Fachs enthalten sind (s. Abbildung 2.1 im Kapitel 2). Diese Funktion konnte mit Hilfe der transitiven Verbindungen zwischen den einzelnen Fächern realisiert werden. Dabei wird eine RDQL-Suche durchgeführt.

Für die Visualisierung der Rückgabemenge wurde die Tabellenform gewählt. Der Tabellenkopf wurde internationalisiert, d.h. dass dieser länderspezifisch aufgebaut wird. Zurzeit sind die Sprachen Englisch und Deutsch vorhanden. Weitere Sprachen können jederzeit hinzugefügt werden. Ein netzwerkartige Darstellung der zurückgelieferten Medien, welche die Beziehungen der Medien hervorhebt, konnte nicht umgesetzt werden. Für diese Konzepte könnten beispielsweise Mind-Map-Technologien eingesetzt werden.

Die Integration der Business Logik von SemantikUB innerhalb von Cocoon funktioniert sehr gut. Es mussten einige spezifische Probleme gelösten werden. Eine genaue Beschreibung findet sich im Abschnitt 5.3.3.

Die folgende Abbildung 6.2 zeigt die Rückgabemenge einer Medienrecherche dargestellt durch einen Webbrowser.



Title	Subject and Keywords	Format	Publisher	Contributor	Rights Management	Resource Identifier
Psychogeriatrics	Informationstechnik	Online-Ressource	Blackwell Publishing		Campuslizenz (UB bzw. Inst.)	http://www.ub.uni-heidelberg.de/dok/13574
New astronomy reviews	Informationstechnik Elektronik Mechanik Mechatronik	Online-Ressource	Elsevier Science		Campuslizenz (UB bzw. Inst.)	http://www.ub.uni-heidelberg.de/dok/10807
Der Steuerberater	Informatik	Online-Ressource	Verl.-Ges. Recht u. Wirtschaft		In "Westlaw" ; Campuslizenz	http://www.ub.uni-heidelberg.de/dok/14235
IEEE robotics & automation magazine	Informatik	Online-Ressource	IEEE	Robotics and Automation Society	Campuslizenz (UB bzw. Inst.)	http://www.ub.uni-heidelberg.de/dok/12170
The American journal of Chinese medicine	Elektronik Mechanik Informationstechnik Mechatronik	Online-Ressource	World Scientific	Cont	Campuslizenz (UB bzw. Inst.)	http://www.ub.uni-heidelberg.de/dok/13045
Systems research and behavioral science	Biologie Informatik	Online-Ressource	Wiley		Campuslizenz o. Klin. MA (UB bzw. Inst.)	http://www.ub.uni-heidelberg.de/dok/10479

Abbildung 6.2: Rückgabewerte einer Mediensuche

6.6 Funktionalität der logischen Aspekte

Im Folgenden werden mehrere Aufrufe beschrieben, welche eine Statement Suche durchführen. Das Ziel der Aufrufe ist es zu zeigen, wie welche zusätzlichen Informationen durch die logischen Konzepte gewonnen werden können. Dazu wird direkt die Business Logik von SemantikUB benutzt und die Funktionsaufrufe sowie deren Rückgabemengen beschrieben. Das erste Beispiel zeigt eine Statement Suche für eine Ressource.

```
StatemenSearch("id82", null, null)
```

Die obige Abfrage liefert alle Statements zurück, welche als Subjekt die Zeichenfolge *id82* besitzen. Bei dieser Ressource handelt es sich um eine elektronische Zeitschrift. Die

Bezeichnung *id82* steht dabei für eine verkürzte URI. Die volle Länge einer URI kann beispielsweise in der Abbildung 6.1 eingesehen werden.

Der Rückgabewerte der obigen Statement Suche wird im nächsten Beispiel gezeigt. Dabei wurde die Information aus dem Modell geholt, welches zusätzliches Wissen, anhand logischer Schlussfolgerungen aufweisen kann.

Die ersten Zeilen des Beispiels stellen normale Statements dar, welche zur Beschreibung der Ressource dienen. Die erste logische Schlussfolgerung wird durch das Statement mit der Eigenschaft *subject* beschrieben. Das Beispiel zeigt vier Statements welche mit der Eigenschaft *subject*. Drei davon sind direkt in der Wissensbasis hinterlegt. Das vierte Fach *http://www.ub.uni-heidelberg.de#Mechatronik* wird mit Hilfe des Generic Rule Reasoners erzeugt (s. 5.2.2).

Die *type*-Eigenschaften zeigen, von wem die Instanz abstammt. Zwei *type*-Bezeichnungen sind temporäre IDs – leere Knoten – und dienen Jena für interne Informationen.

Die zweite logische Schlussfolgerung gibt das letzte Statement mit der Eigenschaft *type* wieder. Der OWL-Reasoner erzeugt dieses Wissen indem er erkannt hat, um was für einen Medientyp es sich handelt (s. 5.2.2).

```
id82  title    New astronomy reviews
id82  format   Online-Ressource
id82  subject  http://www.ub.uni-heidelberg.de#Informationstechnik
id82  subject  http://www.ub.uni-heidelberg.de#Elektronik
id82  subject  http://www.ub.uni-heidelberg.de#Mechanik
id82  subject  http://www.ub.uni-heidelberg.de#Mechatronik
id82  type     http://www.ub.uni-heidelberg.de#Volltextmedium
id82  type     http://www.w3.org/2000/01/rdf-schema#Resource
id82  type     http://www.ub.uni-heidelberg.de#Medium
id82  type     1f4689e:104b2b7d4d6:-7fe4
id82  type     53fb57:104b2cc5797:-7ff4
id82  type     http://www.ub.uni-heidelberg.de#PhysischesMedium
```

Das zweite Beispiel gibt das Konzept der transitiven Fächerhierarchie wieder. Im Folgenden wird zuerst eine Statement-Suche im Jena-Default-Model und danach im *infModel* durchgeführt. Die Statement-Abfrage hat folgenden Aufbau und muss alle Statements finden, die als Subjekt *Informatik* aufweisen und eine Eigenschaft *hatTeilgebiet* besitzen.

```
StatemenSearch("Informatik", "hatTeilgebiet", null)
```

Die Lösungsmenge sieht folgendermaßen aus:

```
//Jena-Default-Model
Informatik    hatTeilgebiet     KuenstlicheIntelligenz
Informatik    hatTeilgebiet     Programmierung
```

```
//InfModell
Informatik    hatTeilgebiet     KuenstlicheIntelligenz
Informatik    hatTeilgebiet     Programmierung
Informatik    hatTeilgebiet     Objektorientierung
Informatik    hatTeilgebiet     JavaProgrammierung
Informatik    hatTeilgebiet     CppProgrammierung
Informatik    hatTeilgebiet     Mehrfachvererbung
```

Das Jena-Default-Model liefert nur zwei Werte zurück, weil innerhalb der Ontologie lediglich die Fächer *Programmierung* und *KuenstlicheIntelligenz* als Teilgebiete der *Informatik* definiert wurden.

Da die Eigenschaft *hatTeilgebiet* jedoch transitiv ist, enthält das *infModel* zusätzliches Wissen, welches anhand der Transitivität entsteht. Aus diesem Grund enthält die zweite Abfrage mehre Rückgabewerte.

6.7 Virtueller Berater

Der virtuelle Berater und der damit verbundene Punkt 4.3.6 des Lastenhefts, konnte nicht umgesetzt werden. Dieser stellt ein eigenes oder erweitertes Konzept dar, welches auf der semantischen Suche aufbauen könnte. Jedoch wurde ein Konzept zur Unterscheidung des Medientyps in die Ontologie integriert. Darin wird untersucht ob das Medium nur in elektronischer Form vorliegt. Anhand dieser Information und den Daten zum Standort kann bereits eine aussagekräftige Information gewonnen werden, anhand derer es möglich ist, wichtige Informationen unerfahrenen Benutzern zukommen zu lassen.

Zusammenfassung

Die Aufgabe der Masterthesis bestand darin, einen Prototypen für die semantische Suche über alle elektronischen Medien der Universitätsbibliothek Heidelberg zu implementieren. Das Ziel des Prototypen war es, zu verdeutlichen, welcher Mehrwert durch den Einsatz von Semantischen Web Technologien gewonnen werden kann.

Zur Realisierung dieses Projektes war es notwendig, sich in diese anspruchsvolle Materie, mit der ein Semantisches Web aufgebaut wird, eingehend einzuarbeiten. Dabei handelt es sich um größtenteils neue Konzepte. Viele der Ideen sind noch im Entstehen, d.h. dass vor allem an Universitäten intensiv an deren Umsetzung und Optimierung geforscht wird.

Aus diesem Grund konnte nur auf wenig Informationen zurückgegriffen werden. Im Internet existieren bereits einige Dokumente zum Thema. Weiterhin wird eine große Menge an Fachliteratur angeboten. Die meiste Literatur zu diesem Thema behandeln jedoch ausschließlich allgemeine Konzepte, welche für die Einarbeitung geeignet sind. Viele Bücher zeigen die weitreichenden Möglichkeiten auf, die ein vollständiges semantisches Netzwerk anbieten könnte, z.B. die Vision der semantischen Web-Services, jedoch fehlen allumfassende Einführungen und praktische Hinweise hierzu.

Die Webseiten hingegen, welche das Thema Semantisches Web behandeln, sind oftmals spezifiziert, d.h. sie beschreiben ausschließlich ein bestimmtes Gebiet. Um jedoch das ganze Konzept des Semantischen Webs zu erfassen, war es deshalb notwendig, viel Zeit in die Einarbeitung zu investieren.

Bei der Realisierung von SemantikUB wurden ausgewählte Werkzeuge eingesetzt, welche das Ziel hatten, die Entwicklung zu erleichtern. Das Modellierungswerkzeug Protégé half bei der Umsetzung der Ontologie. Die fertige Ontologie konnte direkt in Jena importiert werden. Die Interoperabilität zwischen beiden Programmen funktioniert problemlos.

Mit Hilfe von Jena war es möglich, die Ontologie und das darin enthaltene Wissen abzufragen. Jena bietet dazu sehr weitreichende Möglichkeiten. Durch den Einsatz von Reasonern war es möglich, zusätzliches Wissen in der Wissensbasis abzulegen. Dabei wurde mit logischen Funktionalitäten, wie Transitivität oder komplexen Vereinigungsmengen gearbeitet. Diese Funktionalitäten lassen sich mit relationalen Datenbanken nur äußerst schwierig umsetzen.

Jena liefert dazu verschiedene Reasoner mit. Wie im Verlauf der Masterthesis festgestellt wurde, existieren bereits ein Dutzend unterschiedlicher Reasoner. Wie genau jeder einzelne Reasoner funktioniert, wo dessen Vor- und Nachteile und Leistungsgrenzen liegen,

konnte in der Masterthesis nicht genauer untersucht werden. Für den Einsatz von SemantikUB genügten die von Jena mitgelieferten Reasoner, um die benötigten logischen Schlussfolgerungen zu ziehen.

Bei der Einarbeitung über das Semantische Web wurde außer Acht gelassen, dass es sich als schwierig herausstellte, passende Anwendungsbeispiele für SemantikUB zu finden. Durch die ausführlichen Dokumentationen des W3C über RDF und OWL konnte man in der Ontologie transitive Eigenschaften sowie deren Inverse deklarieren, Schnittmengen mit beliebig komplexen Einschränkungen einsetzen oder große Taxonomien aufbauen. Es stellte sich jedoch als problematisch heraus, diese Möglichkeiten sinnvoll einzusetzen, so dass ein Benutzer der Universitätsbibliothek Heidelberg einen Vorteil erhalten kann.

Der Mangel an Einsatzmöglichkeiten liegt vermutlich daran, dass sich unser Wissen auf die heute machbaren Konzepte beschränkt. Die Möglichkeiten, die durch Semantische Web Technologien durchführbar sind, müssen erst verstanden und akzeptiert werden. In SemantikUB konnten zwei Beispiele umgesetzt werden, die mit Hilfe von Semantischen Web Technologien einen Vorsprung gegenüber aktuellen Technologien erzielen.

Die Modellierung des eigentlichen Wissens stellte eine große Herausforderung dar. Betrachtet man eine Ontologie nur oberflächlich, kann der Eindruck entstehen, dass es sich dabei lediglich um eine Sammlung von Begriffen und Eigenschaften handelt. Bereits ohne logische Konzepte ist eine Ontologie i.d.R. ein schwierig zu modellierendes Gebilde. Für den Entwicklungsprozess wird sehr viel Erfahrung benötigt. Es müssen Experten aus dem Gebiet, welches es zu Modellieren gilt, vorhanden sein. Des Weiteren ist es notwendig, dass immer wieder Vergleiche zwischen den Anwenderwünschen und der umgesetzten Ontologie erfolgen. Der Entwickler einer Ontologie benötigt viel Erfahrung auf diesem Gebiet, da jede Änderung in einem fortgeschrittenen Entwicklungsstadium viel Zeit und Geld kosten kann. Der Entwickler muss deshalb in der Lage sein, eine oftmals komplexe und fachbezogene Materie sehr schnell zu durchschauen, diese zu abstrahieren und in der Ontologie umzusetzen. Möglicherweise wird in naher Zukunft der Beruf des Ontologen entstehen.

Bei der Ontologie von SemantikUB war es mehrmals notwendig, diese mit zusätzlichen Funktionalitäten zu erweitern. Die Veränderungen konnten oftmals erst zu einem späteren Zeitpunkt erkannt werden, da es anfangs unmöglich war, die gesamte Materie des Semantischen Webs zu durchschauen. Im Verlauf der Entwicklung konnte jedoch viel Wissen über die benötigten Gebiete gesammelt und entsprechend umgesetzt werden.

Ein sehr interessanter Unterschied zwischen Ontologien und der Objektorientierung besteht in der Verwendung von Klassen und Instanzen. In RDF wird alles durch eine Ressource dargestellt, d.h. es existiert kein direkter Unterschied zwischen Klassen und Instanzen. In OWL können Klassen und Instanzen gleichermaßen verwendet werden. Die Entscheidung, wann eine Klasse oder Instanz verwendet wird, besteht in der Granularität der zu beschreibenden Sache. Instanzen sind normalerweise die am meisten spezifizierten Objekte einer Wissensbasis [NM00].

Es existieren viele verschiedene Ontologiesprachen, die sich in der Verwendung von logischen Konstrukten unterscheiden. Aus diesem Grund ist es empfehlenswert, genau darüber Bescheid zu wissen, welche Vor- und Nachteile der Einsatz der jeweiligen Sprache

mit sich bringt. Dies kann verglichen werden, mit der Auswahl einer Programmiersprache für ein bestimmtes Softwareprojekt.

Ein wichtiger Teil von SemantikUB bestand in der Visualisierung des Wissens. Dabei galt es, die Daten, welche aus einem komplexen Netzwerk von Verknüpfungen stammen, entsprechend zu modellieren. Dieses Ziel konnte nicht umgesetzt werden. Stattdessen wurde mit bekannten Mitteln eine Webseite erzeugt, welche die Ergebnisse beinhaltet. Leider ging dadurch jeglicher Netzwerkeffekt verloren.

Während der Entwicklung von SemantikUB, wurde jedoch immer wieder die Frage gestellt, wie die netzwerkartigen Beziehungen sinnvoll im Browser abgebildet werden können oder wie sich transitive Eigenschaften über mehrere Ebenen hinweg darstellen lassen. Mit Ideen, wie bidirektionalen Links oder Mind-Map Visualisierungstechniken, könnten sich diese Beziehungen darstellen lassen. Jedoch besteht an dieser Stelle noch viel Forschungsbedarf.

Zur Implementierung der Business Logik wurde die Jena-API eingesetzt. Diese stellt ein komplexes Semantisches Web Werkzeug dar. Die angebotene Datenbankunterstützung funktioniert sehr effizient, wodurch sich Möglichkeiten für den produktiven Einsatz mit großen Datenmengen ergeben. Die Jena-API unterstützt alle Möglichkeiten von RDF und stellt darüber hinaus noch hilfreiche Pakete zur Bearbeitung der Ontologie zur Verfügung. Dank der guten Dokumentation des Jena-Teams konnten hier schnelle und gute Resultate erzielt werden.

Eine Frage, die sich während der Entwicklung der Business Logik ergab, konnte nicht ganz geklärt werden. Dabei ging es darum, wie die Übertragung der Daten zwischen dem Jena Modell und Präsentationsschicht umgesetzt werden muss. Einige Möglichkeiten wurden im Abschnitt 5.3.2 vorgestellt. Vermutlich wird erst anhand umfassender praktischer Erfahrungen gezeigt werden können, welches die effektivste Methode ist.

Für die Entwicklung eines Projekts wie SemantikUB wäre es wünschenswert, in einem Team mit weiteren Mitarbeiten oder Studenten zu arbeiten. Eine sinnvolle Aufteilung könnte darin bestehen, jeweils eine Person im Bereich Entwicklung der Ontologie, Implementierung der Business Logik sowie der Visualisierung einzusetzen. Dabei würde eine sehr starke gebietsübergreifende Entwicklung stattfinden, d.h. dass alles vom Aufbau der Ontologie abhängt. Sehr wichtig bei einer solchen Teamarbeit wäre es daher, den aktuellen Entwicklungsstand stetig abzugleichen. Durch den Austausch im Team können vermutlich auch Fragen oder Probleme schneller und besser geklärt werden, als bei einem Ein-Mann Projekt.

Zusammenfassend hat das Projekt SemantikUB gezeigt, dass die Semantik Web Technologie sich bereits in einem fortgeschrittenen Stadium befindet, welche durchaus für den produktiven Einsatz geeignet ist.

Durch die persistente Speicherung des Wissens in einer MySQL-Datenbank kann eine effiziente Arbeitsweise garantiert werden. Die Werkzeuge zum Erstellen von Ontologien sind anspruchsvoll, aber sehr gut zu bedienen. Die Möglichkeiten, um Wissen zu beschreiben, kann sich mit aktuellen Technologien messen und ist diesen sogar überlegen. Reasoner stellen dabei ein Werkzeug dar, um zusätzliche Informationen auf relativ einfache Weise in die Wissensbasis zu integrieren.

Durch den Einsatz von Ontologien können Informationen besser dargestellt werden. Ähnlich wie man vor einigen Jahren von der funktionalen Programmierung zur Objektorientierung übergegangen ist, um Programmcode besser zu bearbeiten, visualisieren und wiederverwerten zu können, versucht das Semantische Web das gleiche Prinzip mit Informationen umzusetzen. Dabei werden keine Softwareklassen, sondern ausschließlich Wissen modelliert. Durch die verbesserte Darstellung der Daten, können diese von Maschinen effizienter verarbeiten werden.

Die Kehrseite beim Einsatz von Semantischen Web Technologien liegt in deren Komplexität. Zum Verstehen und Arbeiten mit diesen Technologien ist ein großer Einarbeitungsaufwand notwendig. Das Konzept stellt sehr hohe Anforderungen an den einzelnen Anwender. Allgemein gesehen ist das Semantische Web nur eine weitere Möglichkeit, um Wissen zu speichern, wobei eine Datenbank eingesetzt und das Wissen anhand von Klassen und Instanzen hinterlegt wird. Befasst man sich jedoch genauer mit diesem Thema, stößt man sehr schnell an eine Grenze, die man mit dem persönlichen Wissensstand, vermittelt durch das Studium und Beruf, nicht überschreiten kann.

Vermutlich liegt darin ein Grund, warum bisher sehr wenig Applikationen existieren, die auf dem Semantischen Web aufbauen. Möglicherweise wird sich ein vereinfachtes Konzept durchsetzten, welches für jeden Anwender einsetzbar und leicht zu verstehen ist.

Ausblick auf die weitere Entwicklung von SemantikUB

Würde ein Semantisches Web in der Universitätsbibliothek Heidelberg verwendet werden, würde dies ein enormer technologischer Vorteil gegenüber anderen Bibliotheken darstellen. Für die vollständige Umsetzung von SemantikUB, wäre es notwendig im Team zu arbeiten. Da jedoch in der Universitätsbibliothek Heidelberg lediglich ein halbes Dutzend Computerspezialisten tätig sind, wird sich dieser Wunsch nur schwierig realisieren lassen.

Innerhalb der IT-Abteilung der Universitätsbibliothek Heidelberg werden zwar kleinere Programme auf Anfrage von Mitarbeitern erstellt, jedoch würde SemantikUB ein größeres Softwareprojekt werden. Die Haupttätigkeit der IT-Abteilung liegt in der Webentwicklung mit standardisierten Frameworks sowie der Wartung und Pflege der Server und Mitarbeiter PCs. Aus diesem Grund ist es im Augenblick nicht möglich, das Projekt zum Abschluss zu bringen.

Eine Alternative bestünde darin, weitere Diplom- oder Masterarbeiten auszuschreiben, um SemantikUB voranzubringen.

Das Ziel einer echten semantischen Suche stellt einen großen Gewinn dar und sollte deshalb konsequent weiter verfolgt werden.

Eine andere weitaus effektivere Möglichkeit bestünde darin, vorhandene Software einzukaufen, welche die im Lastenheft geforderten Punkte umsetzt. Wie eine Präsentation in der Universitätsbibliothek Heidelberg im Frühjahr 2005 von einem der führenden deutschen Hersteller von virtuellen Agenten gezeigt hat, würde der Preis für eine solche Software sehr hoch ausfallen. Der hohe Preis entsteht dadurch, dass eine Ontologie für die Universitätsbibliothek Heidelberg entworfen werden muss.

Für eine Software, welche eine semantische Suche umsetzt, müsste ebenfalls eine solche Ontologie entwickelt werden. Aus diesem Grund kann angenommen werden, dass die Kosten ähnlich hoch sind, wie bei den virtuellen Agenten.

Eine effiziente Möglichkeit die Kosten zu senken, könnte darin liegen, dass die Ontologie von anderen Universitätsbibliotheken wiederverwendet wird. Immerhin wird bereits heute bibliotheksübergreifend mit einheitlichen Auszeichnungsschemas für die einzelnen Medien gearbeitet. Es wäre zu prüfen, ob es nicht möglich wäre, eine einheitliche Ontologie für mehrere Universitätsbibliotheken zu entwickeln? Im Augenblick scheint es auf dem Markt noch keine wirklichen Anbieter für semantisches Suchen zu geben. Daher bleibt abzuwarten, wie sich die Technologien und der damit verbundene Markt in den nächsten Jahren entwickeln wird.

Abstract

Die Masterthesis richtet sich an alle Interessierten des Semantischen Webs und des Wissensmanagements sowie an Webentwickler, die an neuen Technologien interessiert sind.

In die Datenspeicher von Firmen, Universitäten oder des privaten PC gelangen täglich neue Informationen, wodurch ein Informationsüberfluss entsteht. Des Weiteren sind diese Daten oftmals unstrukturiert abgelegt. Das Problem wurde von Forschern frühzeitig erkannt. Seit mehreren Jahren wird daran geforscht, Wissen besser zu verwalten. Ein Konzept, dessen Umsetzung genau dieses zum Ziel hat, wird als das Semantische Web bezeichnet. Mit dessen Hilfe könnte sich ein weltweites strukturiertes Netz von Informationen schaffen lassen. Dabei werden XML-verwandte Technologien verwendet, welche die einzelnen Informationen mit standardisierten Attributen beschreiben und in einem Netzwerk von Verbindungen abzulegen. Durch die Spezifikation der einzelnen Attribute kann von diversen Programmen auf semantischer Ebene auf das Wissen zugegriffen werden.

Die Masterthesis beschreibt das Konzept, welches hinter dem Semantischen Web steht und implementiert eine praktische Anwendung mit Semantischen Web Technologien. Zur Realisierung dieses Projekts werden die verschiedenen Sprachen RDF/S und OWL analysiert und eingesetzt. Des Weiteren erfolgt eine Analyse verschiedener Semantischer Web Werkzeuge.

Das Ergebnis der Masterthesis zeigt, dass es durch das Semantische Web möglich ist, bessere Suchresultate zu erzielen als durch gegenwärtige Datenbankrecherchen und dass sich der Mehraufwand zur Entwicklung eines semantischen Webs lohnt.

Abstract

This master thesis is addressed to all those, interested in Semantic Web and knowledge management as well as for web designers interested in new technologies.

Every day loads of new information get into databases of companies, universities or private pc's resulting in an information overload. In addition, these data often filed without any structure, a problem which has been early recognized by scientists. For many years, research has been done to find a better way to administrate knowledge. A concept with this objective is the Semantic Web. A structured net of information could be developed with the help of Semantic Webs.

In this context XML-related technologies are used which describe single information with standardized attributes to file them in a network of connections. Because of this specification of the single attributes, other programmes can gain access to this knowledge on semantic level. This master thesis describes a concept behind the Semantic Web and implements a practical application with Semantic Web technologies. To realize this project the different languages RDF/S and OWL are analyzed and used, followed by an analysis of different Semantic Web tools.

The result of this master thesis shows that the Semantic Web makes it possible to get better search results as by regular databases research and that the effort to develop a Semantic Web is worth it.

Literaturverzeichnis

[Bec04] BECHHOFER, Sean: *OWL Web Ontology Language Reference*. http://www.w3.org/TR/owl-ref/, 2004

[BFS05] BEHME, Henning ; FLÜGGE, Matthias ; SCHMIDT, Kay-Uwe: Verständnisvolle Dienste. In: *IX* (2005), S. 136

[BL04a] BERNERS-LEE, Tim: *Semantic Web Stack*. http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html, 2004

[BL04b] BERNERS-LEE, Tim: Das Unvollendete. In: *Technology Review* 11 (2004)

[BL04c] BERNERS-LEE, Tim: *What the Semantic Web can represent*. http://www.w3.org/DesignIssues/RDFnot.html, 2004

[Boa99] BOARD, DCMI U.: *Dublin Core Metadata ElementSet*. http://dublincore.org/documents/dcmi-terms/, 1999

[Boa01] BOARD, DCMI: *Dublin Core in RDF*. http://web.resource.org/rss/1.0/modules/dc/, 2001

[Boa02a] BOARD, DCMI: *Dublin Core DTD*. http://dublincore.org/documents/2002/07/31/dcmes-xml/dcmes-xml-dtd.dtd, 2002

[Boa02b] BOARD, DCMI: *Dublin Core Qualifiers*. http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/, 2002

[Bri04] BRICKLEY, Dan: *RDF Vocabulary Description Language 1.0: RDF Schema*. http://www.w3.org/TR/rdf-schema/, 2004

[Chr05] CHRISTOF, Jürgen: *Staats- und Universitätsbibliothek Hamburg Carl von Ossietzky mit virtuellem Agent Stella*. http://www.sub.uni-hamburg.de/, 2005

[Cun04] CUNNINGHAM, Hamish: *Gate Homepage*. http://gate.ac.uk/, 2004

[DOS03] DACONTA, Michael C. ; OBRST, Leo J. ; SMITH, Kevin T.: *The Semantic Web*. Wiley, 2003

[EE04] ECKSTEIN, Rainer ; ECKSTEIN, Silke: *XML und Datenmodellierung*. dpunkt.verlag, 2004

[Fer03] FERBER, Reginald: *Information Retrieval*. dpunkt.verlag, 2003

[Gar04] GARCIA, Antonio: *The Abstract Factory Design Pattern*. http://exciton.cs.oberlin.edu/javaresources/DesignPatterns/FactoryPattern.htm, 2004

[Gea03] GEARY, David: *Simply Singleton*. http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-designpatterns.html, 2003

[GPFLC04] GOMEZ-PEREZ, Asuncion ; FERNANDEZ-LOPEZ, Mariano ; CORCHO, Oscar: *Ontological Engineering*. Springer Verlag Heidelberg, 2004

[Gro05] GROUP, ExoLab: *Castor Project Page*. http://www.castor.org/, 2005

[Gru93] GRUBER, T. R.: Toward principles for the design of ontologies used for knowledge sharing / Technical Report KSL-93-04. 1993. – Forschungsbericht

[Hes04] HESSE, Wolfgang: Ontologie(n). In: *Informatik Spektrum* 25 (2004), S. 477–480

[Hit05] HITZELBEGER, Florian: *Statistik – weltweit 60 Millionen Webseiten*. http://www.domain-recht.de/magazin/article.php?id=423, 2005

[Hje01] HJELM, Johan: *Creating the Semantic Web with RDF*. Wiley, 2001

[HKR+04] HORRIDGE, Matthew ; KNUBLAUCH, Holger ; RECTOR, Alan ; STEVENS, Robert ; CHRIS, Wroe: *A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0*. The University Of Manchester, 2004

[HMW04] HAARSLEV, Volker ; MÖLLER, Ralf ; WESSEL, Michael: *Racer Homepage*. http://www.sts.tu-harburg.de/~r.f.moeller/racer/, 2004

[HS04] HEINER STUCKENSCHMIDT, Harmelen van F.: *Information Sharing on the Semantic Web*. Springer Verlag, 2004

[KK04] KAYA, Birgül ; KAYA, Nilgün: *XML Topic Maps und Resource Description Framework*, Diplomarbeit, 2004

[Kos] KOST, Martin: *Web Ontology Language (OWL) bzw. DAML+OIL*, Diplomarbeit

[KR96] KRISHNAMOORTHY, C. S. ; RAJEEV, S.: *Artificial Intelligence and Expert Systems for Engineers*. CRC Press, 1996

[Lei05] LEIWESMEYER, Barbara: *Regensburger Verbundklassifikation*. http://www.bibliothek.uni-regensburg.de/Systematik/systemat.html, 2005

[Man96] MANFRED, Spitzer: *Geist im Netz*. Spektrum Akademischer Verlag, 1996

[MBL05a] MAYLEIN, Leonard ; BRAUN, Martin ; LANGENSTEIN, Annette: *Heidelberger Virtuelle Bibliothek*. http://www.ub.uni-heidelberg.de/helios/, 2005

[MBL05b] MAYLEIN, Leonard ; BRAUN, Martin ; LANGENSTEIN, Annette: *Science Citation Index (SCI)*. http://www.ub.uni-heidelberg.de/helios/epubl/info/daba/sci.html, 2005

[Med05a] MEDICAL, Stanford: *Protege Homepage*. http://protege.stanford.edu/index.html, 2005

[Med05b] MEDICAL, Stanford: *Schemaweb Homepage*. http://www.schemaweb.info/, 2005

[MH04] MCGUINNESS, Deborah L. ; HARMELEN, Frank van: *W3C OWL Übersicht*. http://www.w3.org/TR/owl-features/, 2004

[Mil04] MILVICH, Michael: *Portierung der Java Anwendung Lendplan nach Cocoon*. 2004

[MM04] MANOLA, Frank ; MILLER, Eric: *RDF Primer*. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2004

[NM00] NOY, Natalya F. ; MCGUINNESS, Deborah L.: *Ontology Development 101: A Guide to Creating Your First Ontology*. http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html, 2000

[Pow03] POWERS, Shelley: *Practical RDF*. O'Reilly, 2003

[RC04] ROBINA CLAYPHAN, Rebecca G.: *Library Application Profile*. http://dublincore.org/documents/library-application-profile/, 2004

[Sav04] SAVARESE, Daniel: *Jakarta ORO*. http://jakarta.apache.org/oro/, 2004

[SH04] STUCKENSCHMIDT, Heiner ; HARMELEN, Frank van: *Information Sharing on the Semantic Web*. Springer Verlag, 2004

[Sow00] SOWA, John F.: *Knowledge Representation: logical, philosophical, and computational foundations*. Pacific Grove, CA : Brooks/Cole, 2000

[TE03] TOLKSDORF, Robert ; ECKSTEIN, Rainer: *Infotag Semantic Web*. www.inf.fu-berlin.de/inst/ag-nbi/lehre/04/P_SW/Einleitung.pdf, 2003

[Tea02a] TEAM, Seekport: *Seekport Haupteite*. http://www.seekport.de/, 2002

[Tea02b] TEAM grenzspiele.online: *Das semantische Web*. http://www.3sat.de/grenzspiele/sendung/39307/, 2002

[Tea04a] TEAM, DAML W.: *DAML Ontology Library*. http://www.daml.org/ontologies/, 2004

[Tea04b] TEAM, HP: *HP Labs Semantic Web Research*. http://www.hpl.hp.com/semweb/, 2004

[Tea04c] TEAM, Jena: *The Jena 2 Ontology API*. http://jena.sourceforge.net/ontology/index.html, 2004

[Tea05a] TEAM, EZB: *Elektronische Zeitschriftenbibliothek*. http://rzblx1.bibliothek.uni-regensburg.de/ezeit/, 2005

[Tea05b] TEAM, Jena: *An Introduction to RDF and the Jena RDF API*. http://jena.sourceforge.net/tutorial/RDF_API/index.html, 2005

[Tea05c] TEAM, Jena: *Jena Framework – Javadoc API*. http://jena.sourceforge.net/javadoc/index.html, 2005

[Tea05d] TEAM, Jena: *Jena Framework – Javadoc API*. http://jena.sourceforge.net/documentation.html, 2005

[Tea05e] TEAM, Schema W.: *Schema Web - RDF Schemas Directory*. http://www.schemaweb.info/, 2005

[Tea05f] TEAM, W3C: *Representing Classes As Property Values on the Semantic Web*. http://www.w3.org/TR/swbp-classes-as-values/#acknowledgements, 2005

[Tea05g] TEAM, Yahoo: *Yahoo Verzeichnis*. http://dir.yahoo.com/, 2005

[Wah01] WAHLSTER, Wolfgang: *Semantisches Web und Wissensmanagement*. 2001

[Wik05a] WIKIPEDIA: *Hauptkategorien der deutschen Wikipedia*. http://de.wikipedia.org/wiki/Kategorie:%21Hauptkategorie, April 2005

[Wik05b] WIKIPEDIA: *OWL Beitrag in Wikipdia*. http://de.wikipedia.org/wiki/Web_Ontology_Language#Weblinks, 2005

[Wle03] WLEKLINSKI, Fabian: *Suche im Semantic Web*, Universität Frankfurt am Main, Diplomarbeit, 2003

Abbildungsverzeichnis

Abbildungsverzeichnis

Beispieleverzeichnis

Glossar

API ..... *Application Programming Interface*, Schnittstelle zu einer Programmbibliothek, die von einem Anwendungsprogrammierer verwendet wird. Zum Beispiel bietet Jena eine komfortable API zur Manipulation von Ontologien.

Cocoon ..... XML-orientiertes Web-Publishing-Framework.

DAML + OIL ..... Ontologiesprachen, aus welchen OWL hervorging.

FOL ..... *First Order Logic*,Prädikatenlogik erster Stufe.

Graph ..... Aus der Mathematik entliehener Begriff, welcher aus Knoten und Kanten besteht.

Java ..... Objektorientierte Programmiersprache. Wird von der amerikanischen Firma Sun entwickelt und hat sowohl im akademischen als auch im industriellen Umfeld eine weite Verbreitung erreicht.

JENA ..... Java-Framework zum Erstellen von Anwendungen für das Semantische Web. Das Jena-Framework wurde von der *Hewlett Packard Labs Semantic Web Research* Gruppe entwickelt. Die aktuelle Version ist Jena 2.2 und steht kostenlos zur Verfügung.

Literal ..... Zeichenkette, welche im Zusammenhang mit RDF genutzt als Objekt in einem Statement eingesetzt wird

Namensraum ..... Mechanismus, mit dem man erreichen kann, dass alle Elemente beispielsweise einer Ontologie eine global eindeutige URI bekommen. So werden Doppeldeutigkeiten verhindert. Der Namensraum wird dazu im Dokument definiert und auf alle Einträge angewendet.

Ontologie ..... Explizite begriffliche Konzeptualisierung eines Anwendungsbereiches mit Hilfe eines definierten Vokabulars.

OWL ..... *Web Ontologie Language*, benannt nach der Eule (engl. owl) aus 'Winnie the Pooh'. Standardsprache des W3C zur Definition von Ontologien.

Protege ..... Werkzeug zum Erstellen und Pflegen von Ontologien

RDF ..... *Resource Description Framework Schema*, Sprache zur Spezifikation von Ontologien.

RDF ..... *Resource Description Framework*, Sprache zur semantischen Beschreibung von Dingen, wobei Tripel vom Typ Subjekt, Prädikat und Objekt eingesetzt werden.

RDQL ..... *Resource Description Query Language*, SQL ähnliche Abfragesprache für RDF-Daten.

Statement ..... Genau ein RDF-Tripel.

Abbildungsverzeichnis

UML ..... *Unifed Modelling Language*, allgemeine graphische Beschreibungstechnik, welche im Bereich der Softwareentwicklung eingesetzt wird.

URI ..... *Unifed Resource Identifier*, Standardsyntax des W3C zur Identifikation einer Ressource.

URL ..... *Unifed Resource Locator*, ein Typ von URI, der eine über das Internet mit einem definierten Protokoll, den Ort und die Ressource selbst identifziert.

W3C ..... *World Wide Web Consortium*, hat die Aufgabe Spezifikationen und Empfehlungen für die Weiterentwicklung des Internets zu entwerfen.

XML ..... *Extensible Markup Language*, ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur.

XML – Schema ... Sprache zur Beschreibung der Struktur von XML-Dokumenten