# Masterarbeit

Artem Khvat

Ontologiebasierte Integration der verschiedenartigen Services

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Artem Khvat

# Ontologiebasierte Integration der verschiedenartigen Services

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Angewandte Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachterin : Dipl. Inform. Birgit Wendholt

Abgegeben am 11. Dezember 2006

**Artem Khvat**

## Thema der Masterarbeit

Ontologiebasierte Integration der verschiedenartigen Services

## Stichworte

XML, RDF, RDF(S), Jena, Protege, Onotlogien, Merge, RDQL, OWL, DAIML, OIL, Semantik Web

## Kurzzusammenfassung

Der Autor stellt die Untersuchung des Einsatzes der Ontologien für die Beseitigung der Problemen der Heterogenität in einer durch Semantik annotierten Umwelt vor.

**Artem Khvat**

## Title of the paper

Ontology-Based integration of diversified Services

## Keywords

XML, RDF, RDF(S), Jena, Protege, Onotlogy, Merge, RDQL, OWL, DAIML, OIL, Semantic Web

## Abstract

The author presents the investigation of the ontology application for problem solving of the heterogeneity in environment with semantic annotation.

*Meinen Eltern*

# Inhaltsverzeichnis

# 1. Einführung

## 1.1. Motivation

In den letzten Jahren ist das World Wide Web zu einem gigantischen Datenspeicher angewachsen. Der Trend der letzten Jahre zeigt, dass es sich in Bezug auf die Datenmengen ständig vergrößern wird. Parallel dazu, führt die schnelle Weiterentwicklung von Netzwerktechnologien und Arbeitsplatzrechnern zu einer Vergrößßerung der Bandbreite. Diese Entwicklungsprozesse stellen neue Qualitätsanforderungen an die Interaktionen mit dem World Wide Web. Immer mehr Herausforderungen des täglichen Lebens finden jetzt ihre Lösung im Internet. Aber nicht immer wird die eigentliche Lösung für den Benutzer dadurch einfacher und übersichtlicher. Als Beispiel dafür, betrachten wird folgendes Szenario welches die Interaktion zwischen dem Menschen und dem Internet beschreibt. Es ist aus verschiedenen Sichten ein Klassiker geworden, da es sich jeden Tag Millionen Mal ereignet. Es geht um die Suche nach Informationen im World Wide Web. Es gibt heute eine Reihe von Suchmaschinen im Internet, die ihre Dienste dem Benutzer frei zur Verfügung stellen. Die am stärksten bekannten davon sind Google (`http://www.google.de`), Yahoo (`http://www.yahoo.com`) und Altavista (`http://www.altavista.com`). Für unser Beispiel wurde die Suchmaschine Google benutzt.

Nehmen wir an, dass wir auf der Suche nach Informationen über ein Auto sind. In unserem Falle handelt es sich um Fahrzeuge des Baujahres 1967. Nachdem der Suchauftrag erledigt wurde, besteht das Ergebnis aus mehreren hundert Seiten, die in vielen Fällen ein unbrauchbares Ergebnis darstellen oder nur sehr wage mit dem ursprünglichen Anliegen in Verbindung stehen. Das einzige gemeinsame Merkmal aller Seiten ist das Vorhandensein der Wörter ’Baujahr’,’Fahrzeug’ und ’1967’. Dabei spielt z.B. die letzte Zahl nicht immer die Rolle eines Baujahres. Am Ende ist es dem Benutzer überlassen aus den vielen Hundert Seiten die für ihn brauchbaren herauszufiltern.

Wie man sieht, beschränkt sich das Suchen der heutigen Suchmaschinen im Internet nur auf das Auffinden von Seiten durch den Vergleich von Zeichenketten. Die Bedeutung des eigentlichen Inhalts bleibt der Maschine verschlossen und kann nicht als Suchkriterium verwendet werden. Die gefundenen Links auszuwerten, ist die oft nicht immer kleine Aufgabe

des Menschen. Eine erhebliche Verbesserung der Suche kann also dadurch erreicht werden, wenn man den Suchmaschinen die semantische Bedeutung der maschinenlesbaren Informationen im World Wide Web verständlich machen könnte.

Um den Maschinen das selbständige Herausfiltern und Verwerten von Informationen zu ermöglichen, wurde eine hierarchische Struktur von Sematic Web Sprachen und verschiedenen Mechanismen entwickelt. Manche davon befinden sich noch in den Prototypstadien, andere dagegen haben schon heute eine weite Verbreitung und Nutzung. Das Semantic Web baut auf dem schon bestehenden Web von heute auf. Dadurch ist die Kompatibilität mit den bestehenden Technologien im Web eine Anforderung. Der geistige Vater und Erfinder des Sematic Webs - Berners-Lee - beschreibt dessen Architektur mit Hilfe des so genannten 'Layer-cake'. Abbildung 1.1 wurde aus (http://www.aifb.uni-karlsruhe.de/WBS/dob/pubs/www2003.pdf) genommen.

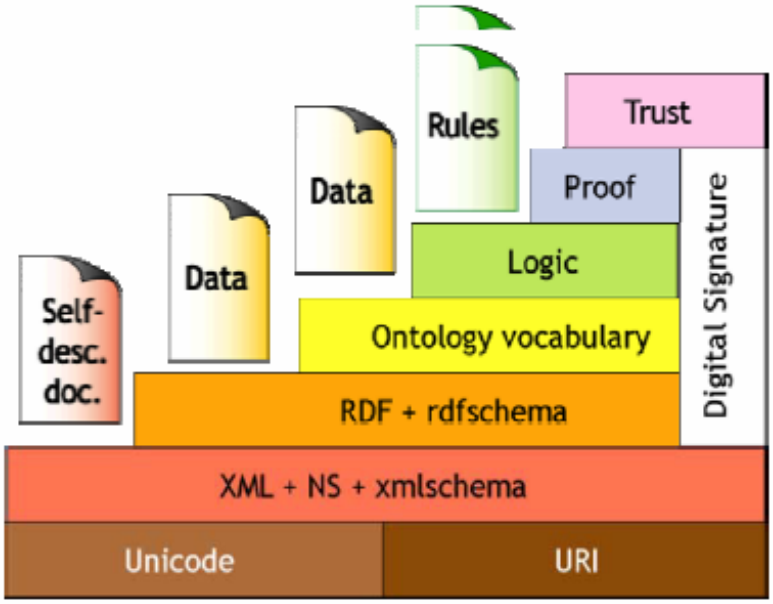


Abbildung 1.1.: Layer-cake des Semantic Web

Unicode ist ein weltweit universeller Zeichensatz. Die ISO Norm 10646 garantiert die Verwendung des international einheitlichen Standards für Bezeichnungen im World Wide Web.

Der 'Uniform Resource Identifier' (URI) [29] hat seine Einführung bereits bei der Entwicklung des World Wide Web gefunden. Er wird zur eindeutigen Bezeichnung von Ressourcen im Netz benutzt und wurde von der W3C eingeführt. Damit man über ein und dasselbe

Objekt in der Welt mit anderen Kommunikationspartnern sprechen kann, braucht man eine eindeutige Identifizierung dieses Objektes. Genau diese Aufgabe wird durch einen URI erledigt. Im heutigen Internet hat daher jede vorhandene Ressource einen nur für sie bestimmten URI. Dieser URI beinhaltet einen eindeutigen Namen und Pfad. Eine Ableitung des URI ist der so genannte URL ('Uniform Resource Locator'). Der Hauptunterschied zwischen den beiden besteht darin, dass der URL zusätzlich zu der eindeutigen Namenssicherung auch gleichzeitig die Lokalisierung des Objektes sichert, wie z.B. bei der URL der Webseite (http://www.google.de). Im Gegensatz dazu muss der URI keinen Pfad des Objektes enthalten.

Während die visuelle Wiedergabe der Daten auf einer Webseite mit Hilfe von HTML (Hyper Text Markup Language) ausgeführt wird, übernimmt XML (Extensible Markup Language) [14] den Austausch von Daten über das Netz. Hiermit werden dem Rechner viel mehr Angaben über die Strukturierung der Daten und teilweise Informationen über den Inhalt zur Verfügung gestellt. Z.B:

<sentence>

<fahrzeug href=http://www.seriouswheels.com/pics-1960-1969/1968-Pontiac-Firebird-Red-Blower-s-sy.jpg>'Firebird' Baujahr

<baujahr>1968</baujahr>

</fahrzeug>, ist das beste was dem Mann, passieren kann.

</sentence>

Der Informationsgewinn wäre in diesem Fall die Möglichkeit der Maschine folgende Dinge auch ohne das Eingreifen des Anwenders zu bestimmen.

- Es handelt sich um einen Satz.
- 'Firebird' ist eine Ressource vom Typ <fahrzeug>.
- Diese Resource wird eindeutig durch den URL (http://www.seriouswheels.com/pics-1960-1969/1968-Pontiac-Firebird-Red-Blower-s-sy.jpg) referenziert und lokalisiert.
- Die Zahl 1968 ist eine Instanz vom Typ <baujahr> und damit nicht nur einfach eine Zahlenfolge.
- Die Zahl 1968 vom Typ <baujahr> ist eine Teilstruktur der Instanz vom Typ <fahrzeug>.

Um die möglichen Probleme der Mehrdeutigkeit zu lösen, werden sog. Namespace-Präfixe (Namensräume) benutzt. Im Folgenden Beispiel kommt <auto> doppelt vor:

<bmw:auto>...</bmw:auto> und <mercedes:auto>...</mercedes:auto>

Die Namespaces mit URI sehen wie folgt aus:

<bmw:auto xmlns:bmw='http://www.bmw.com/xmlns/'>...</bmw:auto>

<mercedes:auto xmlns:mercedes='http://www.mercedes.de/'>...</mercedes:auto>

Sehr wichtig ist dabei, dass XML lizenzfrei als Technologie des W3C zur Verfügung steht und Unterstützung von Seiten der Industrie findet.

Ontologien beschreiben nun die Gegenstände formal. Der Schwerpunkt liegt dabei auf der Beschreibung ihrer Beziehungen zueinander. Die Basis von Ontologien bildet die RDF-Syntax und das RDF-Schema. Zusätzlich enthalten sie Konstrukte, die der Sprache Prolog äußerst ähnlich sind. Ein möglicher Einsatz von Ontologien wird durch folgendes Beispiel veranschaulicht. Nehmen wir an, der Benutzer will eine Suche über verschiedene Autoanbieter starten. Dabei hat dieser den Wunsch ein bestimmtes Auto zu finden. Für die Beschreibung der eigenen Bestände benutzen die Autoanbieter XML - Vorlagen. Des Weiteren nehmen wir an, dass unsere Vorstellung von dem gesuchten Auto wie folgt aussieht:

<?xml version='1.0' standalone='yes' encoding='UTF-8'?> <Auto>

<hersteller>Opel </hersteller>

<model>

<name> Vectra</name>

<farbe> Blau</farbe>

<jahr> 1998</jahr>

</model>

<motor>

<leistung> 75 KW</leistung>

<hub>1600</hub>

</motor>

</Auto>

Während der Suche stoßen wir auf zwei Anbieter, die das gesuchte Auto in ihren Beständen haben. Allerdings weicht deren Vorstellung von dem uns bekannten Auto ab.

Anbieter 1:

```
<?xml version='1.0' standalone='yes' encoding='UTF-8'?>
<Auto>
  <hersteller>Opel</hersteller>
  <name>Vectra </name>
  <farbe>blau</farbe>
  <jahr>1998</jahr>
  <leistung>75KW</leistung>
  <hub>1600</hub>
</Auto>
```

Wie leicht ersichtlich ist, handelt es sich hier um das gleiche Auto. Nur durch die andere Strukturierung des XML - Dokumentes schlägt die Suche fehl, da durch direkten Vergleich der Tags keine Erkennung der semantisch gleichen Autos möglich ist.

Anbieter 2:

```
<?xml version='1.0' standalone='yes' encoding='UTF-8'?>
<Auto>
  <manufacturer>Opel</manufacturer >
  <model>
    <name>Vectra </name>
    <color>Blau</color>
    <year>1998</year>
  </model>
  <motor>
    <power>75KW</power>
    <cylindercapacity>1600</cylindercapacity>
  </motor>
</Auto>
```

Auch bei diesem Anbieter, handelt es sich wieder um das von uns gesuchte Auto. Das Problem ist hier allerdings die unterschiedlichen Bezeichnungen der Tags. Auch dadurch ist ein Informationsgewinn durch einen direkten Vergleich der Tags nicht möglich und die Suche schlägt abermals fehl.

Wie das Beispiel zeigt, kann ohne das explizite Eingreifen des Anwenders die Suche in heterogenen Umgebungen fehlschlagen. Die Suchmaschine ist nicht in der Lage Zusammenhänge zu erkennen, die für uns Menschen offensichtlich erscheinen. Das Problem hierbei ist nicht der Mangel an Rechenkapazität von heutigen Rechnern, sondern vielmehr das Fehlen der Möglichkeit Schlussfolgerungen aus den gegebenen Repräsentationen zu ziehen. Dazu bedarf es parallel zur maschinenlesbaren Beschreibung der Daten einer Beschreibung der Semantik der Daten mittels XML. Und genau diese Lücke sollen die Ontologien schließen. Die Ontologien sollen den Maschinen ermöglichen genau diese Schlussfolgerungen aus Informationen zu ziehen, mit denen sie arbeiten. Die ersten brauchbaren Ontologien, die für diese Aufgabe erstellt wurden, sind schon im Einsatz.

Momentan kann man nicht von 'der einen' Ontologie sprechen. Es existieren verschiedene Stellvertreter davon. Zum Beispiel kam es durch die Initiative der Europäer zu einem Produkt namens OIL. Ein weiteres Beispiel ist die amerikanische Variante DAML. Später haben die beiden Forschungsgruppen ihre Ergebnisse zusammengefügt. Das Ergebnis dieser Vereinigung war die Ontologiesprache OWL (Web Ontologie Language). Allerdings existieren, wie oben schon erwähnt, neben OWL noch weitere Ontologiesprachen. Hier wurden nur die bedeutendsten genannt.

Die Logik, Proof -und Truststufen des 'Layer-cake' sind noch in einem Entwicklungsstadium. Das endgültige Ziel ist der Aufbau eines 'Web des Vertrauens'.

## 1.2. Ziele

Der Autor steht vor der Untersuchung des Einsatzes der Ontologien für die Beseitigung der Problemen der Heterogenität in einer durch Semantik annotierten Umwelt.

Die Untersuchung der Einsatzmöglichkeiten von Ontologien f̈ur die Beseitigung von Heterogenitätsproblemen in einer semantisch annotierten Welt ist das Ziel des Autors. Neben den Vor- und Nachteilen von Ontologien sollen auch Bedingungen bestimmt werden, die für den Einsatz von Ontologien in einer bestimmten Umgebung zu erf̈ullen sind. Dazu wird das im Masterprojekt „Ferienclub“, welches an der Hochschule f̈ur angewandte Wissenschaften Hamburg durchgeführt wurde, entwickelte Werkzeug MOnTo weiterentwickelt. Zur Zeit liegt dieses Werkzeug in der Version 0.3 vor. Die momentan prototypische Funktionalität von MOnTo umfasst neben den Grundfunktionen zur Arbeit mit Ontologien, welche mit den

Sprachen RDF, RDF(S) und OWL erstellt wurden, auch die Möglichkeit der manuellen Integration von mehreren primitiven Ontologien. Eine genaue Beschreibung dieser Software und des Projektes ist auf der Webseite der HAW-Hamburg [7] zu finden. Das Ziel des Autors ist es am Ende der Arbeit eine Software zur Verfügung zu haben, welche die Integration von neuen Informationsquellen in ein System bei gleichzeitiger Minimierung des menschlichen Einsatzes ermöglicht. Einer vollständig automatisierten Integration von Informationsquellen in ein System sind leider Grenzen gesetzt, die es bei der Erstellung der Software zu erkennen gilt. Diese Grenzen werden u.a. durch die Eigenschaften der Umgebung gesetzt. Da einer manuellen Integration nicht ausgewichen werden kann legt der Autor großen Wert auf die Gestaltung dieser, wobei hier auch Fragen der Ergonomie von Softwarewerkzeugen betrachtet werden. Zusammenfassend sollen mittels MOnTo Probleme und mögliche Lösungen, die sich in diesem Gebiet befinden aufgezeigt und verdeutlicht werden.

Folgende Aufzählung listet die in dieser Arbeit behandelten Gebiete der Informatik auf:

- Beschreibung von Inforamtionsartefakten mittels Ontologien
- Integration von Ontologien und die dabei entstehenden Probleme
- Automatisierung der Integration mehrerer Ontologien
- Ergonomiebetrachtungen der werkzeuggestützten manuellen Integration
- Untersuchung integrationsbeeinflussender Eigenschaften heterogener Umgebungen

# 2. Grundlagen

## 2.1. Architektur von Semantik Web

Die Publikation einer Roadmap von Tim Berners-Lee [4] hat die Geburt der neuen großen Vision in der Welt der Informatik gekennzeichnet. In dieser kurzen Zusammenfassung wurde die Zukunft des Internets dargestellt. Diese Zukunft führte den Begriff des „Semantik Web“ ein. Im Jahre 1998 wurden schon die Grundzüge dieses neuen Konzeptes festgelegt. Letztere bilden den Kern der Architektur des Semantik Webs und beinhalten folgende Themen:

- ***Explizite Metadaten***: Die genaue Beschreibung der zur Verfügung stehenden Daten wird durch den Einsatz von Metadaten [2] realisiert. Das führt zur Optimierung und Verbesserung der maschinellen Verarbeitung von Informationen und ermöglicht eine maschinenlesbare Darstellung der Bedeutung von Informationen.
- ***Ontologien*** [6]: Die Ontologien definieren die Zusammenhänge und Hierarchien innerhalb der Informationen und Klassen, die durch die Beschreibung mittels Metadaten entstanden sind.
- ***Logisches Schließen***: Auf Basis der beiden erstgenannten Konzepte können nun aus den bereitgestellten Informationen neue gewonnen werden.

In der Abbildung 1.1 wird der Gesamtaufbau des Semantik Web in sieben spezialisierte Schichten aufgeteilt. Diese werden in den folgenden Abschnitten genauer betrachtet.

## 2.2. XML

XML http://www.w3.org/XML/ ist eine Metasprache für die Erstellung und Festlegung von Dokumententypen, welche von der W3C 1998 standardisiert wurden. Mit XML erhält man Regeln, durch die spezielle Dokumententypen in Abhängigkeit von einem Anwendungsfall, erstellt werden können. Die Grundbausteine jedes XML-Dokumentes sind die sog. XML-Elementtypen, die auch als XML-tags bezeichnet werden. Ein Beispiel dafür ist das Element:

***<a>Das leben ist schön! </a>***

Das hier gezeigte Element „a“ besteht aus einem Start-Tag <a> und einem End-Tag </a>. Innerhalb dieser beiden Tags steht der Inhalt des Elements „Das Leben ist schö!“. Des weiteren bestimmen die beiden Tags auch den Typ des Elements. Z.B. ist das Element:

***<a> Alles ist ok! </a>***

vom selben Typ wie das vorherige Beispiel, nur mit einem anderen Inhalt. Werden in zwei verschiedenen XML-Dokumenten dieselben Elementtypen und die für diese Elemente geltenden Verschachtelungsregeln verwendet, so gehören beide dem selben Dokumententyp an. Ein Beispiel hierfür kann eine Menge von HTML-Dokumenten gesehen werden. Alle HTML-Dokumente haben die gleiche Menge von Elementen zur Verfügung und alle benutzen die selben Verschachtelungsregeln.

Die Grundidee von XML war es eine Möglichkeit zu schaffen, Dokumente, die bestimmte Gemeinsamkeiten in ihrem strukturellen Aufbau haben, auf gleiche Art und Weise verarbeiten zu können. Es sollte dabei keine Rolle spielen, ob diese von unterschiedlichen Erzeugern stammen. Es vereinfacht den Austausch solcher Dokumente und das Schreiben von Programmen zur Verarbeitung dieser. Durch solch einen gemeinsamen Standard sind in XML-konformen Dokumenten keine „Überraschungen“ in Bezug auf ihre Struktur verborgen, die verarbeitende Programme nicht handhaben könnten. Dies macht XML zu einer Richtlinie für die Schaffung von neuen Dokumenttypen.

Heute findet XML eine sehr breite Anwendung in unterschiedlichen Bereichen der Informationsverarbeitung. Es gab schon vor XML zahlreiche Versuche eine universelle Auszeichnungssprache zu definieren. Die meisten davon basieren auf SGML [15], und XML ist dabei keine Ausnahme aber der Trend der letzten Jahren zeigt, dass XML wohl die Führung unter diesen zahlreichen Standards übernehmen wird. Eine der wichtigsten Eigenschaften, die XML die Siegerposition sichert, ist ihre Einfachheit. Dies gilt besonders im Vergleich zu SGML. XML kann fast überall zum Einsatz kommen, wo der Datenaustausch eine zentrale Rolle spielt. Dies sind heutzutage vor allem Anwendungen aus dem Bereich der Web-Technologien, welche Web-Services als Kommunikationsschnittstelle nutzen. Aber auch in anderen Bereichen wie z.B. dem Bankwesen, der Textverarbeitung, der Künstlichen Intelligenz und der Erstellung von Entwicklungsumgebungen für Programmiersprachen ist XML ein unverzichtbares Hilfsmittel geworden.

Allerdings kommt es öfters zu Situationen, in denen die Möglichkeiten von XML überschätzt oder falsch verstanden werden. Z.B. wird manchmal behauptet, man könne mit XML seine eigenen Tags erstellen. Dies könnte dann so verstanden werden, dass durch die Einführung von XML keine eigenen Regeln mehr eingehalten werden müssen. Stattdessen müsse nur eine neue eigene Definition der Daten erstellt werden. Allerdings liefert XML nur die Techniken für die Verarbeitung von Unterschieden in verschiedenen Dokumenten auf der Basis

von Tags. Die Beschreibung der semantischen Bedeutung von Tags fehlt dagegen in XML. Dies impliziert die Voraussetzung das dem verarbeitenden Programm die Reaktionen der einzelnen Tags und ihrer Inhalte bekannt sein muss. Die Fähigkeit eines Programms XML zu verarbeiten ist nur die Möglichkeit die Struktur des aktuellen Dokuments zu erkennen und sie in der selben Ausgangsform wieder darzustellen.

Um die Bedeutung bestimmter Dokumente automatisch oder halbautomatisch zu erkennen sind weitere Techniken erforderlich. Diese werden in den folgenden Abschnitten präsentiert. Natürlich basieren alle diese Techniken auch auf XML, was die Bedeutung von XML für die Zukunft steigert. Eine ausführliche Beschreibung des XML-Standards findet man auf der http://www.w3.org/XML/ Web-Seite von W3C . Die für diese Arbeit verwendete Version von XML ist 1.1.

## 2.3. RDF

RDF [16] ist eine Abkürzung für „Resource Description Framework“. Dieses Rahmenwerk wird heutzutage sehr oft als grundlegende Methode für die Erzeugung, den Austausch und vor allem für die Beschreibung von Erstellung von Metadaten benutzt. Mittlerweile hat RDF seinen Weg in die Lösung unterschiedlicher Probleme in der Informatik gefunden. Ein großes Anwendungsgebiet ist die Suche nach Inhalten in diversen Quellen, zu denen das Internet und auch Bibliotheken gehören. Erstere benutzen RDF um die Qualität ihrer Suchmaschinen zu erhöhen und letztere nutzen das Framework um Beziehungen zwischen den Inhalten von Büchern zu beschreiben. Dies ermöglicht eine bessere Navigation in den elektronischen Katalogen. Die heutigen Blog-Communities nutzen RDF um ihre Benutzereinträge semantisch zu beschreiben. Der Benutzer muss dies von Hand vornehmen, ihm wird allerdings bei Bedarf durch vorgefertigte Schablonen geholfen. Dies optimiert eine spätere Suche nach bestimmten Einträgen. Der konkrete Nutzen der semantischen Beschreibung durch RDF besteht für Suchmaschinen in einem viel breiteren Spektrum an Suchkriterien. Konnten früher nur Einträge und Seiten auf Grund von eines positiven Vergleichs von Zeichenketten gefunden werden, so kann heute nach geografischen oder kulturellen Kriterien gesucht werden.

RDF hat die Hauptaufgabe Mechanismen zu definieren, die eine Beschreibung diverser Ressourcen ermöglichen. Ein Vorteil von RDF ist, dass keine Semantik im Voraus definiert werden muss. Stattdessen ist es erlaubt diese zu einem späteren Zeitpunkt festzulegen. Obwohl RDF nicht auf bestimmte Anwendungsfälle festgelegt ist, erlaubt es die trotzdem die Beschreibung beliebiger Informationen. Mit dem RDF-Datenmodell können sog. syntaxneutrale RDF-Ausdrücke erstellt werden. Man verwendet die Darstellung des Datenmodells, um die Treffer im Sinne der Aussagen rauszufiltern. Demnach sind zwei Aussagen erst gleich, wenn sie eine identische Darstellung im Datenmodell haben. Dadurch wird dem Benutzer

ein gewisser Spielraum, was die Definition von syntaktischen Eigenschaften anbelangt. Die gleiche Semantik kann also durch unterschiedliche Syntax beschrieben werden.

Ein RDF-Modell besteht aus Eigenschaften, die zu Ressourcen gehören und denen Werte zugewiesen werden. Damit entsteht ein sog. Ressource-Attribut-Werttripel mit denen Aussagen formuliert werden können. Im Folgenden werden die drei Grundbestandteile von RDF genauer beschrieben [16]:

- ***Ressourcen (Resources)***: Mit RDF werden Objekte beschrieben, die als Ressourcen bezeichnet werden. Um eine Ressource eindeutig zu identifizieren wird sie durch einen URI bestückt. Optional kann auch eine sog. Anchor-ID angegeben werden. Da es prinzipiell möglich ist für beliebige Objekte URIs zu definieren, beschränkt sich die Anwendbarkeit von RDF nicht nur auf Webdokumente und andere Ressourcen im Netz, sondern kann nach Belieben auf die reale Welt ausgedehnt werden.
- ***Eigenschaften (Properties)***: Unter einer Eigenschaft wird ein Aspekt, ein Charakteristikum oder eine Relation verstanden, welche zum Beschreiben einer Ressource verwendet werden kann. Jede Eigenschaft hat eine bestimmte Bedeutung, welche einerseits die verschiedenen Ressourcenarten festlegt, die mit ihr beschrieben werden können. Des weiteren werden durch sie auch die Beziehungen zu anderen Attributen festgelegt.
- ***Aussagen (Statements):*** Ein RDF-Statement wird durch ein Tripel aus einer Ressource, einer Eigenschaft und dem Wert dieser Eigenschaft für die jeweilige Ressource spezifiziert. Diese drei Teile einer Aussage bezeichnet man als:

  *Subjekt (die Ressource)*

  *Prädikat (die Eigenschaft)*

  *Objekt (der Wert der Eigenschaft)* Mögliche Objekte sind sowohl Zeichenketten oder andere einfache Datentypen, aber auch andere Ressourcen. Letztere werden über einen URI spezifiziert.

RDF verfügt über ein bei objektorientierten Programmiersprachen übliches Klassensystem. In diesem wird das Basisvokabular definiert. Die Menge der Klassen wird als Schema bezeichnet. Die Hierarchie der Klassen kann durch Unterklassenbildung erweitert werden. Diese Erweiterung führt dann zu einem weiteren Schema welches nur geringe Unterschiede zum Ausgangsschema aufweist. Über das bei der Beschreibung verwendete Schema ist es möglich Informationen über Ressourcen zu gewinnen.

Schemata können gemeinsam genutzt werden. Dadurch können Metadaten wiederverwendet werden. Durch die Erweiterbarkeit und die Möglichkeit der gemeinsamen Nutzung von Schemata, können sich verschiedene Arten der Vererbung zu Nutze gemacht werden. Dabei entstehen verschiedene Sichten auf die Informationsarchitektur und die Möglichkeit diverse

Definitionen miteinander zu mischen. Z.B. ist es möglich die Beschreibung einer Pizzabestellung im Internet zur Verfügung zu stellen, in der nicht nur die Zutaten, sondern auch der Preis und der Anbieter enthalten sind.

### 2.3.1. RDF Syntax

Die Syntax von RDF [17] ist in zwei Arten unterteilt. Bei der sog. Basic ***Serialization Syntax*** wird jede Eigenschaft in einem separaten Element abgelegt. Die zweite Art ist die Basic ***Abbreviated Syntax*** und bietet die Möglichkeit Aussagen in kompakter Form darzustellen. Im Folgenden wird genauer auf diese beiden Arten eingegangen.

#### Basic Serialization Syntax

In [17] vielen Fällen werden die Eigenschaften einer Ressource zusammengefasst, so dass man selten eine RDF-Aussage allein vorfindet. Diese Gruppierung von Aussagen wurde durch das Einführen eines speziellen Elements „description“ ermöglicht. Das Attribut „about“ des Elements „description“ enthält die Informationen, zu welchen Ressourcen die zutreffende Aussage zugeordnet ist. Es ist zudem möglich, dass die Ressource bisher noch nicht vorhanden ist. Es kann sich aber auch um eine physikalische Ressource ohne eine URI-Identifikation [29] handeln. In diesem Fall hat der Entwickler die Möglichkeit die Ressource durch die Erzeugung eines eindeutigen id-Attributs zu identifizieren. Man kann in einem description-Element mehrere Eigenschaften mit dem gleichen Namen unterbringen. Stellt man sich das ganze Dokument als einen gerichteten Graphen vor, so bedeutet jedes neue Einfügen eine weitere Kante in diesem Graphen.

#### Basic Abbreviated Syntax

Es [17] gibt Situationen, in denen es sehr nützlich sein kann eine kompaktere Schreibweise zu verwenden. Ein Beispiel hierfür ist die Implementierung von Programmen zur automatischen Erzeugung von RDF-Beschreibungen. Diese Art der Syntax nennt man RDF-Abbrevieted-Syntax. Diese unterteilt sich weiter in drei Arten. In der ersten ist es erlaubt Eigenschaften zu verwenden, die in einem description-Block nur einmal vorkommen, wobei die Werte der Eigenschaften Zeichenketten sein müssen. Diese Eigenschaften werden dann als Attribute des description-Elements eingetragen. Im Szenario des Internets wird es Browsern, die kein RDF unterstützen dadurch ermöglicht trotzdem die Eigenschaftswerte anzuzeigen.

Die zweite Art verwendet eine Verschachtelung der description Elemente. Sie wird verwendet, wenn eine Ressource als Wert einer Eigenschaft präsentiert werden muss, wobei diese Eigenschaft innerhalb der gleichen Instanz beschrieben wird.

Die dritte und letzte Art wird verwendet, wenn ein *type* als Wert einer Eigenschaft im description-Element steht.

#### Container

Es ist des Öfteren notwendig eine Menge von Ressourcen als Gruppe zur Verführung zu stellen. Dazu stellt RDF drei Arten von Containern zur Verfügung [18].

- ***Bag***- ist eine unsortierte Liste von Ressourcen oder Zeichenketten. Ein Bag wird verwendet, wenn eine Eigenschaft verschiedene Werte annehmen kann und keine spezialisierte Anordnung erforderlich ist.
- ***Sequence***- ist eine geordnete Liste von Ressourcen oder Zeichenketten. Man verwendet Sequenzen, wenn eine Eigenschaft verschiedene Werte annehmen kann und die Reihenfolge der Werte von Bedeutung ist.
- ***Alternative***- ist eine Liste von Ressourcen oder Zeichenketten, die eine Darstellung von Alternativen für den Wert einer Eigenschaft repräsentiert.

#### High-order statements

RDF bietet neben der Möglichkeit Aussagen über Ressourcen zu treffen, auch die Möglichkeit Aussagen über Aussagen machen zu können. Letztere werden mit Aussagen höherer Ordnung bezeichnet. Sie bestehen aus den folgenden Teilen [19].

- ***Subject*** - realisiert die Identifizierung der Ressource bei der Beschreibung, durch das modellierte Statement. Daraus folgt, dass der Wert des Subject die Ressource ist, über die die Ausgangsaussage getroffen wurde.
- ***Predicate*** - ist der Verweis auf die Ausgangseigenschaft der modellierten Aussage. Der Wert des Predicates ist eine Ressource, welche eine Spezifikation der Eigenschaft der ursprünglichen Aussage ist.
- ***Object*** - ist der Verweis auf den Eigenschaftswert in der modellierten Aussage. Die Object-Eigenschaft hat den Wert des Objektes der Ausgangsaussage.
- ***Type*** - identifiziert die Klasse einer Ressource.
- ***Value*** - ist die einfache Zuweisung einer Eigenschaft.

## 2.4. Ontologien

### 2.4.1. Definition

Ontologien [6] dienen als Grundlage für die Abbildung verschiedener Strukturbegriffe und des Wissens. Ursprünglich kommt das Wort aus der Philosophie, wo es als Begriff aus dem Bereich der Erkenntnistheorie bekannt ist. Mann kann die Definition der Ontologie auch im Duden finden, wo es als "die Wissenschaft des Seins"beschrieben wird. Im Bereich der künstlichen Intelligenz (KI) wird die Ontologie nicht mehr als Wissenschaft, sondern als ein Werkzeug mit unterschiedlichen Ausprägungen für bestimmte Anwendungsfälle betrachtet. In der KI versteht man eine Ontologie als eine Sammlung von Methoden für die Strukturierung von Wissen aus verschiedenen Anwendungsbereichen. Durch eine Ontologie wird Wissen in einer hierarchischen Struktur geordnet und durch Beziehungen miteinander verknüpft. Dies erlaubt eine weitere Verwendung dieses Wissens auf einer nächsten Abstraktionsebene. Im Prinzip stellt eine Ontologie eine Einigung und Festlegung auf Begriffe für die Beschreibung von Wissen innerhalb eines abgegrenzten Umfeldes dar. Letzteres wird als Domäne bezeichnet.

Durch den Einsatz solcher strukturierten Begriffe und durch die Festlegung ihrer Verwendung wird die Kommunikation innerhalb eines durch fachliche Aspekte spezialisierten Gebietes erleichtert. Das Hauptproblem dabei ist der Prozess der Einigung unter den einzelnen Kommunikationsmitgliedern. Das Ergebnis ist meistens eine neue Ontologie, die für alle Beteiligten verständliche Beschreibungsstrukturen definiert.

Ein weiteres Nutzungsfeld, welches der Einsatz von Ontologien eröffnet, ist die Möglichkeit aus denen in einer Ontologie enthaltenen Fakten neue Informationen zu deduzieren. Man kann dies mit einem PROLOG-System vergleichen in dem es durch die Abfrage des Systems zu einer Gewinnung von neuen Informationen, welche nicht explizit bei der Systemerstellung eingegeben wurden, kommt.

Ontologien bilden keine Abbildung der realen Welt. Stattdessen sind sie nur Modelle, welche für die optimale Lösung bestimmter Aufgaben konzipiert wurden. Man kann diese mit Programmlösungen aus der Software-Entwicklung vergleichen, bei der das Programm lediglich nur Mittel zur Problemlösung ist [6].

Bei der Erstellung der Ontologien versucht man oft das Problem zu vereinfachen. Es gibt auch Abstraktionsmöglichkeiten bei der Ontologiebildung. Das erlaubt die Verallgemeinerung einer Lösung für ein breiteres Spektrum von verwandten Aufgaben. Aus dem Betrachtungswinkel der KI wird eine Ontologie als Formalisierung und Festlegung der Konzepte und Modelle einer Domäne gesehen. Die bestimmten Ontologie-Modellierungssprachen ermöglichen einen automatisierten Umgang mit ihnen. Das Ontological Engineering ist relativ jung,

weshalb es noch viele ungelöste Probleme bei der Benutzung von Ontologien gibt. Im Folgenden werden einige davon kurz vorgestellt:

- Keine globalen verbindlichen Ontologien - das ***WWW*** hat heute viele Strukturen, die keine Hierarchie aufweisen und stark verteilt sind. Diese Tatsache kann dazu führen, dass es ähnliche Ontologien entstehen, welche die gleiche Domäne beschreiben. Der Autor definiert dieses Problem als Heterogenität in einer semantisch annotierten Umgebung. Es können unterschiedliche Arten der Heterogenität entstehen. Z.B. sind dies gleiche Konzepte mit unterschiedlichem Strukturaufbau oder unterschiedlicher Namensgebung. Eine genauere Beschreibung der möglichen Fälle wird in den späteren Kapiteln dieser Arbeit erläutert.

- Qualität von benutzerdefinierten Ontologien - hier kommt die Frage auf, nach welchen Kriterien die Qualität einer Ontologie beurteilt wird? Welche Möglichkeiten hat der Benutzer oder, was noch interessanter ist, ein Programm für die Bestimmung der Qualität von zwei Ontologien der gleichen Domäne? Sehr wichtig ist es, dem Endnutzer Schlussfolgerungen über die konzeptionelle Vollständigkeit zu ermöglichen. Dies umfasst die Fähigkeit Aussagen über die Vertrauenswürdigkeit, die Wiederverwendbarkeit und die Integrierbarkeit einer Ontologie machen zu können.

- Versionsmanagement - der fehlende hierarchische Aufbau großer Teile des WWW fordert eine ständige Weiterentwicklung der Domänen. Dabei werden ständig neue Versionen der zugehörigen Ontologien geschaffen. Es ist durchaus denkbar, dass ab einer bestimmten Version, die Ontologien nicht mehr kompatibel zueinander gemacht werden können. Man braucht einen Mechanismus, der es schafft zwischen den einzelnen Versionen einer Ontologie zu unterscheiden. In DAML+OIL wird diese Aufgabe durch das *<daml:version>*-Tag übernommen. Es gibt aber keine einheitliche Strategie für die Versionspflege von Ontologien. Dies führt zu Problemen, wenn diese z.B. verteilt entwickelt, bearbeitet oder ergänzt werden.

- Agenten: Ich weiß, dass ich nicht alles weiß- bei der Arbeit mit den Ontologien haben die Agenten und Schlussfolgerungssysteme keine selbständigen Annahmen zu machen, die ihr seitens der Ontologie vorgegebenes Wissen überschreiten. Sie dürfen auch nicht annehmen, dass sie das ganze Informationsspektrum der Domäne über die Ontologie zur Verfügung gestellt bekommen. Sie müssen immer davon ausgehen, dass auch andere Konzepte vorhanden sind, über die sie nicht informiert wurden.

- Entwicklung von neuen Ontologien - hier kommt die Frage, nach der Art und Weise, wie die neuen Ontologien entstehen sollen, auf. Wer soll der Autor sein, und welche Qualifikationen sollte er mitbringen? Soll am Ende die Erstellung einer Ontologie der Erstellung einer Web-Seite von heute ähneln, wo jeder der in der Lage ist HTML zu schreiben, automatisch zum Web-Designer wird? Oder soll diese Aufgabe den qualifizierten Fachkräften überlassen werden, um einem späteren Chaos von nutzlosen

vom Standart abweichenden Ontologien vorzubeugen. Es kann aber auch passieren, dass die Qualitätsanforderungen so hoch sein werden, dass es nur wenige Spezialisten gibt, die dazu in der Lage sein werden. Dies kann sogar zum Aussterben dieser Technologie auf Grund der hohen Komplexität führen. Als Beispiel hierfür kann die Metasprache SGML aus der Geschichte des Semantik Webs dienen.

- Wie geht man mit komplexen/unscharfen Begriffen um? Wie genau können die Ontologien wirklich die Definitionen der Natursprachen oder von mathematischen Formeln wiedergeben. Können Ontologien auch Begriffe aus nicht naturwissenschaftlichen Bereichen der Wissenschaft genauso exakt beschreiben, dass man bei jeder beliebigen Interpretationsstelle das gleiche Ergebnis bekommt?
- Wie genau werden die Begriffe im Markup eingesetzt? - Sogar bei klar definierten Begriffen und Beziehungen kann es passieren, dass Autoren seitens der Informatik es nicht an der richtigen Stelle oder im falschen Kontext einsetzen. Für die Lösung dieses Problems braucht man eine Anleitung für die Benutzung der einzelnen Elemente im Rahmen einer Ontologie der Domäne.

### 2.4.2. Typen von Ontologien

In der Literatur findet man heute generell zwei Kriterien nach dem Ontologien unterschieden werden können [6]. Zum Einen werden Ontologien nach Reichhaltigkeit der internen Struktur klassifiziert und zum Anderen geschieht die Unterscheidung auf Basis des Konzeptualisierungs-Subjektes. In diesem Kapitel wird die Klassifizierung der zweiten Art näher beschrieben und die Bedeutung der einzelnen Ontologietypen näher erläutert.

#### Allgemeine Ontologien

- repräsentieren allgemeine Wiederverwendbarkeit des Wissens innerhalb einer Domäne
- beinhalten ein Vokabular, welches in Verbindung mit Objekten wie Zeit, Ort, Funktion und Komponenten steht.

[htbp]

**Beschreibung der Zeiteinheit "Minute":**

GOMEZ-PEREZ, Asuncion: Ontological Engineering. Springer London, 2005. – ISBN 1-85233-551-3 Seite 32.

```
(defin -frame Minute)
    : own - slot
        (Documentation "Time Unit")
        (Instance-Of Unit-of-Measure))
    : axiom - def
        ((Quantity.Dimension Minute Time-Dimension)))
        (define - frame Second-of-Time
    : own - slots
        ((Documentation „ The SI standard unit of time")
        (Instance-Of Si-Unit Unit-of-Measure)
        (Quantity.Dimension Time-Dimension))
    : axiom - def
        ((=Minute (* 60 Second-Of-Time))))
```

Abbildung 2.1.: Allgemeine Ontologie

**Top-level Ontologien**

- beschreiben sehr allgemeine Konzepte
- liefern Regeln für das Verbinden der Wurzelterme einer Ontologie
- sind oft wiederverwendbar

**Domain Ontologien**

- sind wiederverwendbar innerhalb einer Domäne
- liefern ein Vokabular über Konzepte, Relationen und Aktivitäten innerhalb einer bestimmten Domäne
- basieren oft auf Top-level Ontologien

**Aufgabenbezogene Ontologien**

- Liefern ein Vokabular, welches für die Lösung des Problems benutzt wird, welches der Domäne gehören wird, oder auch nicht.

**Domänen-Aufgabenbezogene Ontologien**

- sind aufgabenbezogene Ontologien, die wiederverwendbar innerhalb einer Domäne und nicht domänenübergreifend sind
- sind applikationsunabhängig
- Beispiel Travel-Domain: next city, previous city

**Methodenontologien**

- Eine Ontologie über die Terminplanung, mittels Aufgabenaufspaltung würde dieser Kategorie angehören

**Applikationsontologien**

- applikationsabhängige Ontologien
- enthalten alle Definitionen, die notwendig für die Modellierung des Wissens bestimmter Applikationen sind
- Beispiel: Ontologie für Reisebüros mit Reisen nur nach Kanada

Wie man aus den oben vorgestellten Definitionen der einzelnen Typen von Ontologien entnehmen kann, entscheiden zwei wichtige Kriterien über die Brauchbarkeit einer Ontologie für die Lösung bestimmter Aufgaben:

- Nutzbarkeit
- Wiederverwendbarkeit

Diese beiden Eigenschaften stehen im direkten Konflikt zueinander. Je mehr eine Ontologie von einer Eigenschaft besitzt, desto weniger behält sie von der anderen.

### 2.4.3. Merge

Wie schon in ***Kapitel 1.2*** erwähnt wurde, ist eine der Hauptziele von Ontologien, eine strukturierte Darstellung von Wissen zur Verfügung zu stellen, die dann eventuell verteilt genutzt werden kann. Für diese Zwecke muss eine Domäne eine einheitliche Ontologie für alle Benutzer zugänglich machen. In der realen Welt kommt es oft vor, dass eine Domäne unterschiedliche Ontologien für ihre Beschreibung hat. Dies kann passieren wenn z.B. zwei Forschungsgruppen aus einem und demselben wissenschaftlichen Bereich, unabhängig voneinander am selben Problem arbeiten. Hier ist es durchaus möglich, dass jede Gruppe eine andere Vorstellung von der Wissensarchitektur der Domänen während der Arbeit entwickelt hat. Dies soll auch kein Problem darstellen, solange diese zwei Gruppen getrennt voneinander arbeiten oder zumindest kein Informationsaustausch bezogen auf die gemeinsam benutzte Domäne stattfindet. Soll aber eine Zusammenarbeit ermöglicht werden, müssen die erstellten Ontologien zusammengeführt und auf einander abgestimmt werden. Dieser Prozess der Vereinheitlichung von zwei oder mehreren Ontologien wird Merge (von lat) genannt. In der Literatur findet man unterschiedliche Einsätze für das Mergen von Ontologien. Die bekanntesten davon sind:

- ONIONS der Conceptual Modeling Group CNR Rome, Italian
- FCA-Merge der Universität Karlsruhe, Deutschland
- PROMPT von Stanford Medical Informatics Group Stanford University, USA

In diesem Kapitel will der Autor den PROMPT [9] Ansatz genauer beschreiben . Die PROMPT Methode des Ontologien-Merges wurde an der Stanford University von der Medical Informatics Group unter der Leitung von Natalie Noy entworfen und als plug-in für die Protege-2000 Plattform realisiert. Die nachfolgende Beschreibung basiert im Wesentlichen auf dem Beispiel des Tutorials des PROMPT plugins. Das Tutorial sowie das Plugin sind unter (`http://protege.stanford.edu/plugins/prompt/PROMPT_Tutorial.zip`) zu finden und können frei unter Berücksichtigung der Eigentümerlizenzen heruntergeladen und benutzt werden. Die Grundannahme der PROMPT Methode ist, dass die Formalisierung der verwendeten Ontologien durch allgemeine framebasierte Modelle realisiert werden. Der Merge der beiden Ontologien wird in einen Merge der Bestandteile der einzelnen Ontologien aufgeteilt (z.B. Merge von zwei einzelnen classes, merge von zwei einzelnen slots). Im PROMPT Plugin für Protege-2000 sind diese Grundoperationen implementiert, was eine gewisse Automatisierung des Merges unterstützt. Die Realisierung von diesen Grundoperationen erlaubt eine iterative Vorgehensweise auf der die gesamte Methode von PROMPT basiert. Jeder Wiederholungszyklus sieht die Benutzung dieser Operationen vor. Im Prinzip besteht der gesamte Mergevorgang mit der PROMPT Methode aus fünf aufeinander aufbauenden Aktivitäten.

- Aktivität 1. Erstellung von Listen mit Empfehlungen für die Benutzung der Grundoperationen. Die Grundoperationen sind hier Operationen, die für den einfachen Umgang von Ontologien notwendig sind [6] und Operationen, die speziell für die Aufgaben des Merges der beiden Ontologien zuständig sind. Es folgt eine Liste mit den meisten Operationen:

  1) Erstellung der neuen Klasse in der neuen Ontologie auf Basis der Ausgangsklasse der gemergten Ontologie

  2) Erstellung des neuen Slots in der neuen Ontologie auf Basis des Ausgangsslots der gemergten Ontologie

  3) Festlegung der neuen Beziehung zwischen Slot und der Klasse in neuer Ontologie auf Basis der Beziehung zwischen dem Slot und der Klasse der Originalontologie

  4) Erstellung der neuen Klasse in einer neuen Ontologie, durch die deep-copy dieser Klasse aus der Originalontologie

  5) Erstellung der neuen Klasse in einer neuen Ontologie, durch die shallow-copy dieser Klasse aus der Originalontologie

- Aktivität 2. Auswahl der nächsten Operation. Der Benutzer soll während dieser Aktivität sich für die Verwendung einer oder mehrerer Operationen aus der vorgeschlagenen Liste entscheiden

- Aktivität 3. Die Ausführung der ausgewählten Operationen. In dieser Aktivität werden die ausgewählten Operationen ausgeführt

- Aktivität 4. Aufdeckung der aufgetretenen Konflikte. Diese Aktivität ist dafür zuständig, dass nach der Ausführung der ausgewählten Operationen, die neu entstandenen und übriggebliebenen Konflikte aufgedeckt werden. PROMPT unterscheidet zwischen folgenden Arten von Konflikten:

  *Namenskonflikt* (mehr als ein Frame mit dem gleichen Namen)

  *Referenceskonflikt* (ein Frame verweist auf einen anderen Frame, der nicht mehr existiert)

  *Redundancekonflikt in Klassenhierarchie* (mehr als eine Vaterbeziehung zur Root-Klasse)

  *Slotwert Konflikt* (Inkompatibilität zwischen Typen)

- Aktivität 5. Aktualisierung der Operationslisten

Die aufgelisteten Aktivitäten werden in mehreren Iterationen stattfinden, bis alle Konflikte gelöst wurden oder der Abbruch des Vorgangs durch den Benutzer stattgefunden hat.

## 2.5. Ontologie Sprachen

### 2.5.1. RDFS

In RDF gibt es keine Möglichkeit zur Definition von Beziehungen zwischen Eigenschaften und Ressourcen. Für die Deklarationen dieser Beziehungen wurde RDF-Schema entwickelt. Um eine Ressource zu beschreiben, muss man in der Lage sein, bestimmte Aussagen über diese Ressource zu treffen. Eine Ressource aus dem Bereich Auto-Industrie kann z.B. solche Attribute wie "Hersteller", Ërstzulassung", "Baujahr", für ihre Beschreibung nutzen. Die Ressourcen aus dem Bereich der Lebensmittelproduktion werden die Attribute "Brennwert", "Haltbarkeitsdatum", SZutaten"haben können. Die Deklaration dieser Eigenschaften (Attribute) und ihrer Semantik wird durch RDF-Schemata auf Basis von RDF vorgenommen. Man legt nicht nur die Definition der Eigenschaften von Schemata fest, sondern hat auch die Möglichkeit die Art der Resource zu definieren, die zu dieser Eigenschaft gehören.

Die RDF(S)[20] Dokumente werden nicht nur für die Beschreibung von Vokabularen von deskriptive Elemente benutzt. Man kann mit RDF(S) auch Elemente und Klassen definieren und dann bei deren späterer Kombination bestimmte Einschränkung festlegen und deren Verletzung erkennen. Die Beschreibungssprache eines Schemas wird durch die Menge der Ressourcen und Eigenschaften definiert. Diese Ressourcen und Eigenschaften sind selbst Bestandteile eines jeden RDF-Modells und bauen auf RDF-Schemata auf.

Man kann das System von RDF-Schema [20] mit einem Klassensystem einer objektorientierten Programmiersprache vergleichen. Es gibt aber hier einen sehr wichtigen Unterschied. In einer objektorientierten Sprache wird die Definition von Objekten durch eine Eigenschaft, die ein Objekt hat realisiert. In den RDF-Schemata wird davon ausgegangen, dass die Identifikation der Klassen von Objekten möglich ist, ohne dass ihnen bestimmte Eigenschaften zugeordnet werden müssen. In den RDF-Schemata legt man für jede Eigenschaft fest, für welche Klassen von Objekten ihre Anwendung möglich ist.

***Kernklassen***

- *rdfs:Ressource* ist eine Klasse zu der alle Ressourcen gehören [20]. Alles was mit RDF beschrieben werden kann, gehört zu den Instanzen dieser Klasse.
- *rdfs:Property* ist die Klasse der Eigenschaften. Sie ist eine Unterklasse von Resource [20].
- *rdfs:Class* Das Klassenkonzept legt ein abstraktes Objekt fest und dient in Verbindung mit rdf:type zur Erzeugung von Instanzen [Wiki Literatur Hiweise]. Man kann es mit dem ähnlichen Konzept der Klasse aus C++ vergleichen. Es gibt aber einen sehr interessanten Aspekt. Wie man aus Abbildung 2.2 sehen kann, ist rdfs:Class eine Unterklasse von rdfs:Ressource, da die Definitionen der Klassen selbst Ressourcen sind

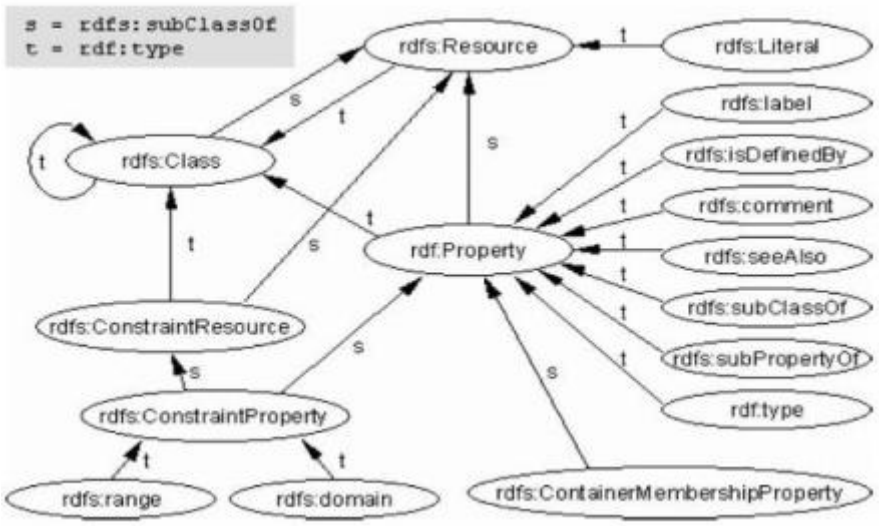


Abbildung 2.2.: RDFS (http://www.w3.org/TR/2000/CR-rdf-schema-20000327/hierarchy.gif)

und rdfsResource auch eine Instanz von rdfs:Class ist, da es wiederum selbst eine Klasse ist [20].

- *Rdfs:Literal* ist eine Klasse für Zeichenketten und etc [20].

***Core Properties*** Die folgenden Eigenschaften haben ihre Definition in den RDF-Schemata gefunden:

- *rdf:type* ist zuständig für die Möglichkeit der Zuordnung einer bestimmten Ressource zu einer gewünschten Klasse. Die Ressource wird zur Instanz dieser Klasse und hat allen typischen Charakteristiken dieser Klasse zu entsprechen.
- *Rdfs:subClassOf* definiert eine Untermengen/Obermengen-Beziehung zwischen Klassen. Sie hat die Eigenschaft der Transitivität und fordert von der Ressource eine Zugehörigkeit zu den Instanzen der Oberklasse, im Falle dass die Ressource eine Instanz einer Unterklasse dieser Klasse ist [30].
- *rdfs:subPropertyOf* Eine transitive Eigenschaft zur Festlegung von Vererbungshierarchien von Eigenschaften.
- *rdf:seeAlso* spezifiziert eine Ressource, die Informationen über die Ressource enthält, die mit dieser Eigenschaft ausgestattet ist.
- *rdfs:isDefinedBy* spezialisiert *rdfs:seeAlso* und verweist auf eine Ressource, die die Definition der vorliegenden Ressource ist.

- *rdfs:domain* Legt den Anwendungsbereich einer Eigenschaft in Bezug auf eine Klasse fest. Da in RDFS die Eigenschaften immer unabhängig von den Klassen definiert werden, muss mit diesen Eigenschaften festgelegt werden, für welche Klassen eine Eigenschaft sinnvoll ist. So könnte zum Beispiel als Domäne der Eigenschaft 'Farbe', sowohl die Klasse 'Auto', als auch die Klasse 'Ball' festgelegt werden [30].
- *rdfs:range* Legt den Wertebereich einer Eigenschaft fest. Damit kann zum Beispiel ausgesagt werden, dass die Eigenschaft - 'FanVonHelgeSchneider' als Wert nur ***TRUE*** oder ***FALSE*** annehmen kann [30].

### 2.5.2. OWL

OWL [21] ist eine weitere Sprache für die Beschreibung von Ontologien. OWL basiert genauso wie RDF(S) auf der RDF-Syntax und stellt eine weitere Stufe in der klassischen Semantik Web Architektur dar. Historisch gesehen ist OWL eine Weiterentwicklung der Ontologiebeschreibungssprache DAIML+OIL [22], die sich als Erfolg der W3C im Bereich des Ontological Engineerings bewiesen hat.

Die OWL Architektur besteht im Kern aus drei Ebenen, die aufeinander aufbauen und durch ihre semantische Ausdrucksmächtigkeit unterschieden werden können. Diese Ebenen sind - OWL Lite(), OWL DL (Description Logic) und OWL Full [21]. In letzterer steht dem Entwickler der gesamte Sprachumfang von OWL zur Verfügung. Die zwei anderen Ebenen definieren nur bestimmte Teile von diesem Umfang und sind in ihren Ausdrucksmöglichkeiten begrenzt. Dies hat einen bestimmten Grund. Die Komplexität der OWL-Sprache ist sehr hoch und nicht alle Aufgaben benötigen den kompletten Sprachumfang für ihre Lösung. Aus diesem Grund wollte man die Optimierung der Ressourcen Verteilung ermöglichen, indem man für Aufgaben mit unterschiedlicher Komplexität unterschiedliche Sprachvarianten anbietet. Das hat sich als sinnvoll erwiesen nachdem es Fällen kam, wo ein Logik-Reasoner nicht in der Lage war in endlicher Zeit zu einem Ergebnis zu kommen. Man kann schon aus dem Namen der Ebene Informationen über ihre Komplexität entnehmen. Die einfachste Form der OWL-Sprache bildet die OWL-Lite Variante. Als nächstes kommt OWL-DL und die Krönung dieser Hierarchie bildet OWL-Full, welche das komplette Spektrum der Möglichkeiten dieser Sprache dem Entwickler zur Verfügung stellt.

OWL Ebenen sind untereinander völlig abwärtskompatibel. Das heißt, alles was unter Verwendung einer niedrigeren Ebene beschrieben wurde, kann ohne weitere zusätzliche Änderungen in einer höheren Ebene verwendet werden. In den nächsten Abschnitten wird der Autor die Eigenschaften und Sprachkonzepte der einzelnen Ebenen nähere beschreiben.

### OWL Lite

Jede Klasse der Sprache OWL und jedes Attribut wird von der Klasse *owl:thing* abgeleitet. Man kann dies mit einer Klassenhierarchie in einer objektorientierten Programmiersprache wie z.B. Java vergleichen, wo alles eine Ableitung der Klasse java.lang.Object ist. In OWL Lite [23] werden folgende RDF(S) Konzepte verwendet: *Class, rdfs:subClassOf, rdf:Property, rdfs:subPropertyOf, rdfs:domain, rdfs:range.*

Mit OWL Lite ist der Entwickler in der Lage Äquivalenz- und Differenzeigenschaften zum Ausdruck zu bringen. Für solche Ausdrücke können Sprachkonzepte wie *owl:sameAs, owl:differentFrom, owl:equivalentClass* etc. verwendet werden. Man kann die Kardinalitäten zwischen Instanzen der einzelnen Klassen festlegen. Dafür hat man die Sprachelemente *owl:minCardinality, owl:maxCardinality, owl:cardinality* eingeführt. Es muss aber berücksichtigt werden, dass in OWL Lite nur die Werte 0 oder 1 für diese Zwecke erlaubt sind. In OWL DL wurde diese Beschränkung aufgehoben. Dort ist die Möglichkeit der Versionsverwaltung gegeben. Die Versionen von einzelnen Ontologien können angegeben und die veralteten Elemente als *deprecated* markiert werden.

### OWL DL und OWL Full

OWL DL [23] stellt fast den kompletten Sprachumfang von OWL zur Verfügung. Die Begrenzung liegt bei der Typentrennung. Eine Klasse kann z.B. nicht gleichzeitig eine Instanz und eine Eigenschaft sein. Das Hauptziel von OWL DL ist die Unterstützung von Beschreibungslogiken und ist auf die optimale Verarbeitung durch Inferenzmaschinen ausgerichtet.

OWL Full [23] enthält keine Beschränkungen mehr und stellt dem Entwickler den kompletten Sprachumfang zur Verfügung. Mit OWL Full hat man die Möglichkeit sehr komplexe Modellierungen durchzuführen. Ein Problem, dass bei der komplexen Modellierung mit OWL Full entstehen kann, ist die Schaffung von Ontologien die für Inferenzmaschinen unentscheidbar sind.

## 2.6. Datenabfrage

Die Abfrage von RDF Tripeln und Graphen ist sehr ähnlich den Abfragen, die an eine relationalen Datenbank gestellt werden. Im Gegensatz zu den Datenbanken, wo SQL [12] als ein Standard für Abfragesprachen feststeht, konkurrieren momentan die Abfragesprachen im Bereich der RDF noch miteinander. Die unterschiedlichen Kandidaten für Abfragesprachen unterscheiden sich jeweils in ihrer Syntax und Ausdrucksstärke. Einer der Favoriten dieses Rennens ist RDQL, welche allerdings auch gleichzeitig über die meisten Einschränkungen

verfügt. Eine Erweiterung dieser Sprache ist SeRQL, welche schon heute relativ breite Anwendung findet. Nach Meinung des Autors ist allerdings SPARQL die Zukunftssprache, da sie sich schon heute zu einem leistungsstarken Standard entwickelt hat. In den folgenden Abschnitten werden alle drei Sprachen kurz vorgestellt.

### 2.6.1. RDQL

RDQL [24] ist eine Anfragesprache die sich auf vorhandenen Daten des Modells orientiert. RDQL verwendet keine Inferenzmechanismen. In RDF werden diese Arten der Anfragesprachen als Tripel-Daten-Sprache genannt. Das Hauptmerkmal der Sprachen ist, dass sie keine Schemata oder Ontologieinformationen verwenden, Ausnahmen sind, wenn die RDF-Quellen explizit angegeben werden. RDF hat eine gerichteten Graphen, die Knoten dieses Graphs sind Ressourcen oder Literale. In RDQL ist man in der Lage einen Vergleichsgrapen zu definieren. Dieser Graph wird mit dem RDF-Graph verglichen, das Ergebnis dieses Vergleichs ist ein Teilgraph, der aus der Menge der Übereinstimmungen zwischen den beiden Graphen gebildet wird. Sprachbeschreibung RDQL ist sehr dem SQL-Sprache ähnlich. In SQL spezialisiert sich auf Datenbanken. In From-Klausel werden die Tabellen der Datenbank identifiziert. Die Bedienung wird in WHERE-Klausel untergebracht und kann durch AND erweitert werden. In RDQL ist Netwerk das was Datenbank in SQL ist und die FROM-Klausel defeniert die RDF-Modelle. Eine RDQL-Anfrage besteht aus folgenden Elementen:

**SELECT** dieses Element bestimmt Anfragevariablen, die in dem Anfrageergebnis zurückgegeben werden sollen.

**FROM** dieses Element realisiert die Spezifikation der zu untersuchenden Modelle mit Hilfe von URIs.

**WHERE** dieses Element beinhaltet die List von konjunktiv gebundenem Triple-Muster. Die Muster werden bei der Ausführung der Anfrage mit allem Tripel des ausgewählten Modells verglichen. Die übereinstimmenden Tripel werden in Ergebnis ausgegeben.

**AND** dieses Element stellt Bedienungen für die Gültigkeitsbereich der Variablen aus SELECT.

**USING** diese Element wird benutz für die Verbesserung der Übersichtlichkeit der Anfrage und ermöglicht die Nutzung der Platzhalter.

### 2.6.2. SeRQL

SeRQL [10] ist eine Abfragesprache des Sesame Framework [11] . Sie ist eine Erweiterung der Sprache RDQL und besitzt alle wesentlichen Eigenschaften der RDQL-Sytax. Man hat

in SeRQL zusätzlich die Möglichkeit der Einbindung von optionalen Pfaden hinzugefügt, indem sie in die eckigen Klammern innerhalb der FROM-Klausel eingegeben werden. Für die Differenzfunktion wird zusätzlich eine MINUS-Klausel hinzugefügt. Die Quantifizierung wird durch die Eingaben WHERE NOT EXIST realisiert. Es gibt aber keine Möglichkeit für die Aggregation von Daten.

Im Vergleich zu RDQL hat SeRQL von seitens des Sprachumfangs keine Nachteile, ist aber auf das Sesame Framework begrenzt und wird in der Zukunft von SPARQL abgelöst (Angekündigt für Version 2.0).

### 2.6.3. SPARQL

Im Moment wird die Weiterentwicklung von SPARQL [25] stark von der W3C angetrieben. Seit April 2006 ist SPARQL auf der Entwicklungsstufe Candidate Recommendation eingeordnet. SPARQL soll RDQL in naher Zukunft ablösen. Sie ist RDQL ähnlich mit dem Zusatz, dass sie über ein zusätzliches Protokoll verfügt. Die Syntax von SPARQL ist der von RDQL sehr ähnlich, bietet aber viel breitere Palette von Möglichkeiten an. Ein innovativer Schritt ist z.B. die Benutzung von Präfixen am Anfang einer Abfrage. Ein Präfix hat die Rolle einer Variablen für bestimmte URIs. Das ermöglicht die Verwendung dieser Variablen direkt in der Abfrage, was die Lesbarkeit erhöht. In SPARQL haben viele Eigenschaften aus SeRQL ihren Platz gefunden, wie z.B. optionale Pfade und Constraints.

Die Benutzung von SPARQL ist heutzutage schon mit dem Jena Framework [1] möglich.

## 2.7. Reasoning

Für die Generierung von neuem Wissen aus abgefragten RDF-Daten und für die Ableitung der Informationen, die in einer Ontologie implizit verborgen sind, werden sog. Reasoning Systeme für RDF benötigt [6] (Kap. 4.2.2). Solche Systeme bestehen aus zwei wesentlichen Komponenten: Die Regeln, die es ermöglichen zum ursprünglichen Wissen zu kommen und der Reasoner selbst, der diese Regeln verarbeiten kann. Die Regeln, die bei dem Reasoning benutzt werden, sind spezielle Aussagen die nur dann ihre Gültigkeit gewinnen, wenn die dazugehörigen Voraussetzungen erfüllt sind. Man kann sagen, dass jede Aussage aus Voraussetzungen und Folgerungen besteht. Im Semantik Web ist die ausdrückliche Formulierung dieser Regeln möglich aber man kann, wie bereits erwähnt wurde, die Formulierung der Regeln aufgrund der impliziten Informationen innerhalb einer Ontologie gewinnen. Zur Veranschaulichung wird folgendes Beispiel betrachtet: Man definiert in OWL eine Klasse 'Cabrio', die eine Unterklasse (subclassOf) der Klasse 'Auto' ist. Dann kann daraus folgendes

geschlossen werden: Hat man eine Instanz der Klasse 'Cabrio', dann handelt es sich um ein 'Auto'.

Es ist auch möglich Äquivalenzen unter einzelnen Konzepten festzulegen. Man nimmt an, dass in einer Domäne eine Klasse/Eigenschaft (hatFarbe) existiert und bei den anderen Domänen hat man die äquivalente Klasse/Eigenschaft (hasColor). Nun besteht die Möglichkeit die beiden Klassen für die Datenabfrage durch Reasoner äquivalent darzustellen. Es wird ein entsprechender Ausdruck (*owl:equivalentOf*) benutzt. Dadurch wird eine implizite Regel der Ontologie erstellt welche besagt, dass die Klasse/Eigenschaft (hatFarbe) genauso wie Klasse/Eigenschaft (hasColor) zu behandeln ist.

Die Formulierung der logischen Regeln ist auch unabhängig von der jeweiligen Ontologie möglich. Als Beispiel kann man die folgende Situation betrachten: Man definiert die Typen von Autos in der Abhängigkeit ihrer Leistungen. Die Leistung unter 100 PS wird als Familienauto bezeichnet und die Autos über 159 PS als Sportwagen.

Für die Anwendung solcher definierten logischen Regeln verwendet man Inferenzprogramme. Die Logikebene des Semantik Web wird gemeinsam durch solche Regeln und Inferenzprogramme gebildet. Die Programme RACER [6] (Seite 266) ,FaCT [6] (Seite 265) und Pellet [6] (Seite 265) sind die bekanntesten Vertreter davon.

Die Abfragesprachen RDQL [24] , SeRQL [10] und SPARQL [25] sind nur für die reine Datenabfrage gedacht und können nur innerhalb einer Ontologie agieren. Versucht man Daten zu arbeiten, die aus verschiedenen Ontologien stammen, ist man auf die Anwendung der Reasoner angewiesen. Diese leiten den Zwischenschritt, der erforderlich ist um von den Daten einer Ontologie zu den Daten der anderen zu kommen, ab.

Ein weiteres Anwendungsgebiet von Reasonern ist die Instanzprüfung. Mit dieser kann festgestellt werden, ob eine Klasse Instanz einer anderen ist oder nicht. Dabei können mehrere unterschiedliche Ontologien bearbeitet werden.

## 2.8. Websevices

Web Services finden in der letzten Zeit immer mehr Verwendung bei der Lösung unterschiedlicher Anwenderaufgaben aus dem Bereich der verteilten Systeme. Sie bieten dem Entwickler große Vorteile bei ihrer Verwendung in Bezug auf Plattformunabhängigkeit, Sprachneutralität und der automatisierten Zusammenarbeit verteilter Komponenten. Man versucht auf Basis des World Wide Web [26] eine Struktur aufzubauen die sowohl für die Integration von Anwendungen innerhalb von Unternehmen, als auch für die Realisierung von Aufgaben, die die Grenze des Unternehmens überschreiten (z.B. B2B [31] - und B2C [32]), geeignet ist.

Man findet heute diverse Web Services Lösungen, die vor allem durch ihre Komplexität unterschieden werden können. Es fängt bei den einfachen zustandslosen Applikationen, die in der Lage sind nur die Anfrage-Antwort Kommunikation zu behandeln an und reicht bis zu sehr aufwendigen, stark verteilten, heterogenen, prozessorientierten Szenarien (zum Beispiel Verwaltung der Besucher in einem Ferienclub [7]). Es kann dabei sehr oft passieren, dass die einzelnen Komponenten solcher Anwendungen selbst aus mehreren Web Services bestehen. Unter Berücksichtigung dieser Eigenschaften, kommt man auf folgende Definition. Web Services sind abgeschlossene modulare Dienste, die sich selbst mit Hilfe von XML [14] beschreiben. Somit können Anwendungen diese Dienste auf Grund ihrer Beschreibung finden und weitere für die Lösung ihrer eigenen Aufgaben verwenden. Die Endvision von diesem Ansatz ist es, Anwendungen selbständig auffinden, auswählen und einsetzen zu können. Web Services sollen die Integration von unterschiedlichen Modulen in einem schon bestehenden System erheblich vereinfachen. Man kann mit ihrer Hilfe schon vorhandene Teilfunktionen von Altanwendungen kapseln und sie dadurch für die restliche Welt verfügbar machen. Ein weiterer Schritt in diese Richtung ist die Anreicherung der Beschreibung von Web Services durch zusätzliche Semantik. Die Semantik soll Probleme bei der automatisierten Auswahl, dem Auffinden und dem Einsetzen von Prozessen lösen. Heute passiert dies immer noch sehr oft durch die vordefinierten Angaben des Programmierers. Außerdem soll die Maschine in der Lage sein aus der Semantik der Beschreibungen eigenständige Schlussfolgerungen extrahieren zu können. Dies kann zum Beispiel sehr hilfreich bei der Suche nach Alternativen für einen nicht vorhandenen Dienst sein. Viele der heutigen Technologien (zum Beispiel WSDL [27]) unterstützen solch eine semantische Beschreibung nicht, was unter anderem eine exakte syntaktische Festlegung der Schnittstellen voraussetzt. Die semantischen Web Services sind ein wichtiger Bestandteil der Architektur von intelligenten Web Services, welche in engem Zusammenhang mit der Vision des Web of Trust von Berners-Lee [4] steht. Im World Wide Web von morgen werden die meisten Aufgaben von Agenten gelöst ohne oder nur mit minimalem manuellen Einsatz. Die intelligenten Web Services sollen dabei die Rolle der Vermittler, Anbieter und Suchportale für die Agenten übernehmen. In diesem Zusammenhang ist die in Kapitel drei vorgestellte Ontologie Sprache OWL-S [28] nach Meinung des Autors von großer Bedeutung. Es spielen auch vier andere Technologien eine wichtige Rolle bei der semantischen Anreicherung von Diensten. In diesem Zusammenhang müssen die Ontologiesprachen RDF [16] und RDF(S) [20] erwähnt werden, welche in relativ einfacher Form eine semantische Beschreibung ermöglichen. Mit RDF und RDF(S) ist die Festlegung eines gemeinsamen Vokabulars möglich, welches dann von OWL-S [28] als Basis zur Beschreibung der Ontologie verwendet wird. Ein mögliches Beispiel, wo die semantische Anreicherung von Web Services durchaus Sinn macht, wäre ein Unternehmen welches die Aufgabe hat die Angebote verschiedener Dienstanbieter zu verwalten. Dies war auch die konkrete Aufgabe im Projekt Ferienclub [7]. Hierbei waren die verschiedenen Dienstanbieter Autovermietungen, welche ihre Angebote an den Ferienclub weiterleiten wollten. Das Problem bestand in der Heterogenität der unterschiedlichen Proto-

kolle, welche zum Datenaustausch verwendet wurden. Durch eine semantische Beschreibun der Dienste wird die automatische Abfrage der einzelnen Angebote möglich. Im besten Fall wird das manuelle Eingreifen von seitens der Benutzer für die mögliche Vermittlung zwischen den unterschiedlichen Anwendungen überhaupt nicht benötigt.

# 3. Analyse

## 3.1. Analyse der Modellierungsunterschieden

In semantisch annotierten Umgebungen kommt es zu verschiedenen Formen der Modellierungsunterschieden. Letztere werden in diesem Abschnitt untersucht. Zur Veranschaulichung werden Beispiele dienen, die allesamt aus dem Kontext des Projekts Ferienclub genommen wurden. Konkret handelt es sich um eine Autovermietung. In dem Beispielumfeld existieren mehrere dieser Unternehmen, wobei jedes eine eigene Vorstellung von der Welt entwickelt und diese entsprechend modelliert hat. Eine Voraussetzung dabei ist, dass alle Unternehmen die selbe Ontologiesprache [Kapitel 2.5]für die Modellierung verwendet haben. Die Aufgabe besteht nun darin, die möglichen Unterschiede der Modelle zu analysieren und zu klassifizieren. So kann später eine systematische Lösung für die auftretenden Fälle der Modellierungsunterschieden entwickelt werden. In folgenden Beispielen werden eigene Bezeichnungen für die einzelne Fälle eingeführt. Diese Namen sollen die Kerneigenschaften der jeweiligen Modellierungsart verdeutlichen. Insgesamt sind fünf Fälle zu unterscheiden. Hierbei wird nicht der Anspruch auf Vollständigkeit der vorgestellten Klassifizierung erhoben. Es ist durchaus möglich, dass bei speziellen Anwendungen und Problemlösungen andere Variationen semantischer Modellierungsunterschieden auftreten können oder eine feinere Unterscheidung erforderlich sein kann. Der Autor unterscheidet grundsätzlich zwischen zwei Arten der Modellierungsunterschieden.

#### Syntaktische Modellierungsunterschiede

Diese Form von Modellierungsunterschieden ist im Wesentlichen dadurch gekennzeichnet, dass die Unterschiede zwischen zwei Modellen im Bereich der Namensgebung liegen. Betrachten wir folgendes Beispiel: Autovermietung A die ihr Hauptgeschäft überwiegend im englischsprachigen Raum hat, besitzt folgende Vorstellung von einem Auto (s. Abbildung 3.1).

Jetzt ist die Autovermietung A, wegen der Erweiterung ihres Geschäftes in Europa, zu einer Kooperation mit dem Partner (Autovermietung B) aus dem deutschsprachigen Raum gezwungen. Aufgrund der Tatsache, dass die Autovermietung B ihr früheres Geschäft immer in

Deutschland betrieben hat, hat sie auch ihre eigene Vorstellung von einem Auto entwickelt. Diese kann wie in Abbildung 3.1 aussehen.

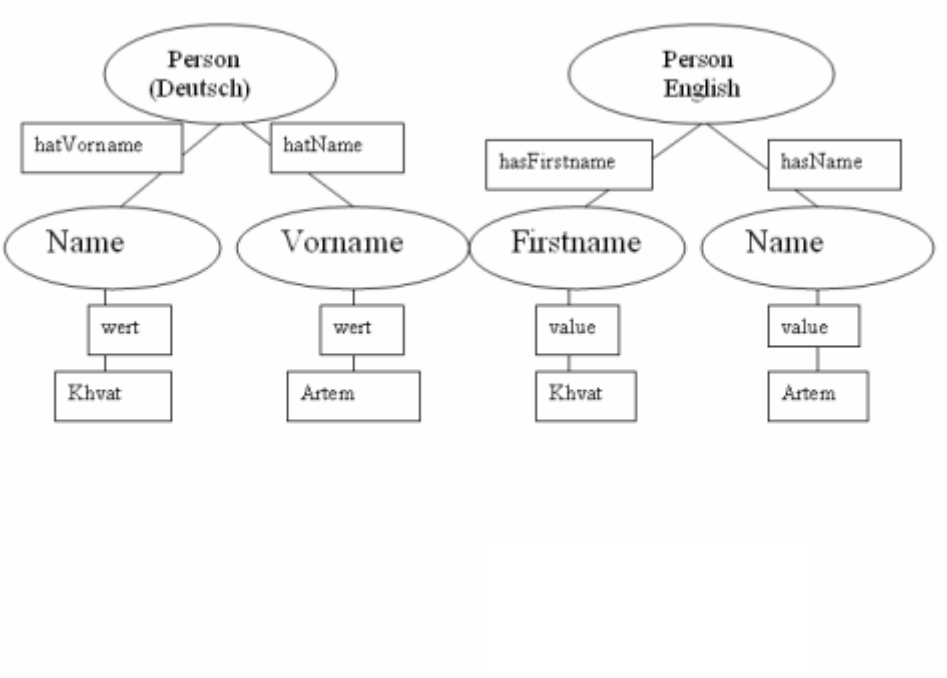


Abbildung 3.1.: Syntaktische Modellierungsunterschiede

Wie aus beiden Modellen zu entnehmen ist, liegen die Hauptunterschiede zwischen den beiden Autovermietungen generell im Bereich der Namensgebung (es wurden unterschiedliche natürliche Sprachen benutzt). Diese Art der Modellierungsunterschieden wird als syntaktische Modellierungsunterschiede bezeichnet.

#### Topologische Modellierungsunterschiede

Diese Form der Modellierungsunterschieden ist im Wesentlichen dadurch gekennzeichnet, dass die Unterschiede zwischen den betrachteten Modellen im Bereich der Baumtopologie der Ontologien liegen. In diesem Fall unterscheidet der Autor zwischen topologischen Unterschieden auf einer abstrakten Klassenebene und topologischen Unterschieden auf der Attributebene der resultierenden Instanzen. Diese beiden Arten müssen dabei nicht getrennt voneinander auftreten. Im Folgenden werden alle möglichen Fälle klassifiziert und durch entsprechende Beispiele verdeutlicht.

- ***Attribut-topologische Modellierungsunterschiede :*** Hier handelt es sich um den Fall, bei dem das Attribut einer Klasse in zwei oder noch mehr Domänen dargestellt

ist. Die Attributdarstellung geht jedoch nicht über die Grenze einer Klasse in jeder beteiligten Ontologie. Ein Beispiel dafür kann folgendermaßen aussehen. Betrachtet werden drei Autovermietungen A, B und C. Alle drei haben eine unterschiedliche Vorstellung von einem Auto, welche in Abbildung 3.2 dargestellt werden. Man erkennt, dass sich diese drei Modelle durch die Darstellung des Attributs 'Person' unterscheiden. Die Verteilung der Attributdarstellung geht in jeder der drei Ontologien nicht über die Grenzen einer Klasse hinaus. In diesem Fall ist es überall die Klasse 'Person'.

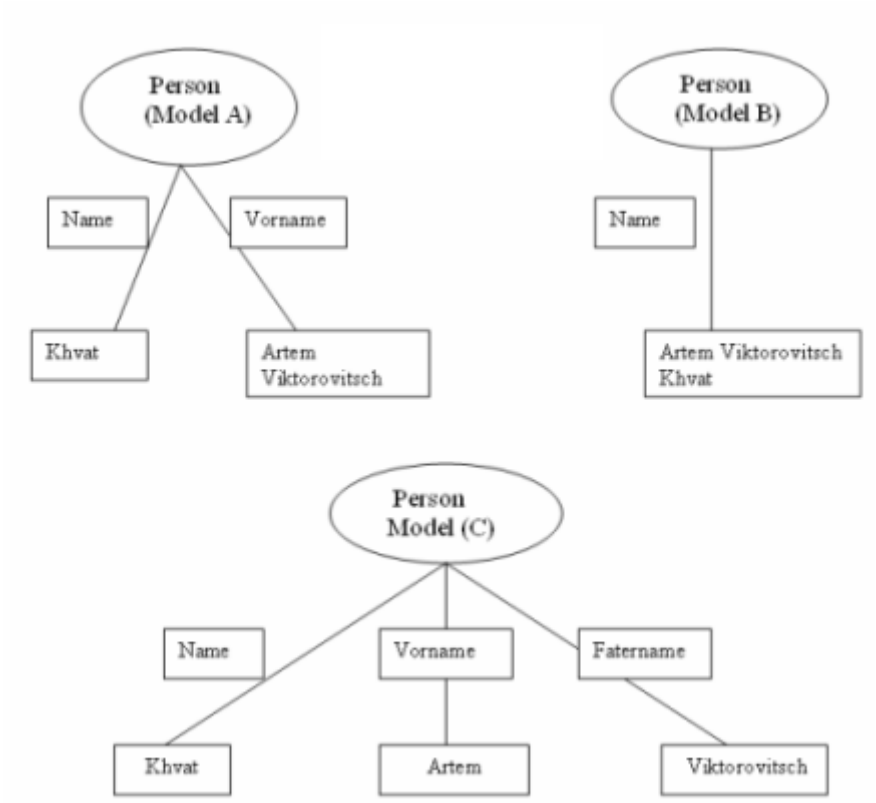


Abbildung 3.2.: Attribut-topologische Modellierungsunterschiede

- ***Attribut-abstrakte topologische Modellierungsunterschiede :*** Hier handelt es sich um den Fall, bei dem das Attribut in zwei oder noch mehr Domänen unterschiedlich dargestellt wird. Die Attributdarstellung in jeder einzelnen Ontologie kann auf mehrere Klassen verteilt werden. Ein Beispiel dafür kann folgendermaßen aussehen. Betrachtet werden zwei Autovermietungen A und B. Die unterschiedlichen Vorstellungen eines Autos, die die einzelnen Unternehmen haben, werden in Abbildung 3.3 dargestellt. Man erkennt, dass sich diese Modelle durch die Darstellung des Attributs 'Person' unterscheiden. Die Verteilung der Attributdarstellung ist bei der Autovermietung A auf die Klasse 'Person' beschränkt. Bei Autovermietung B sind es zwei Klassen 'Name' und 'Vorname'. In Abbildung 3.4 wird eine weitere Art der Verteilung, die zu dieser Art der Modellierungsunterschieden gehört, dargestellt.

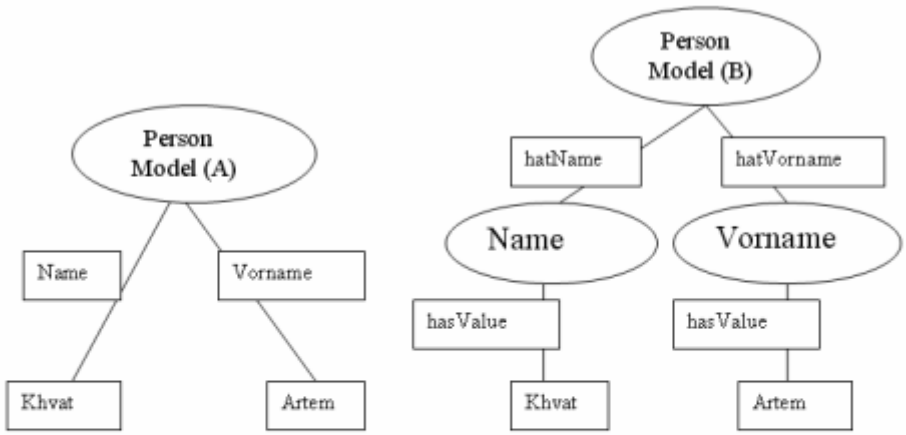


Abbildung 3.3.: Attribut-abstrakte topologische Modellierungsunterschiede

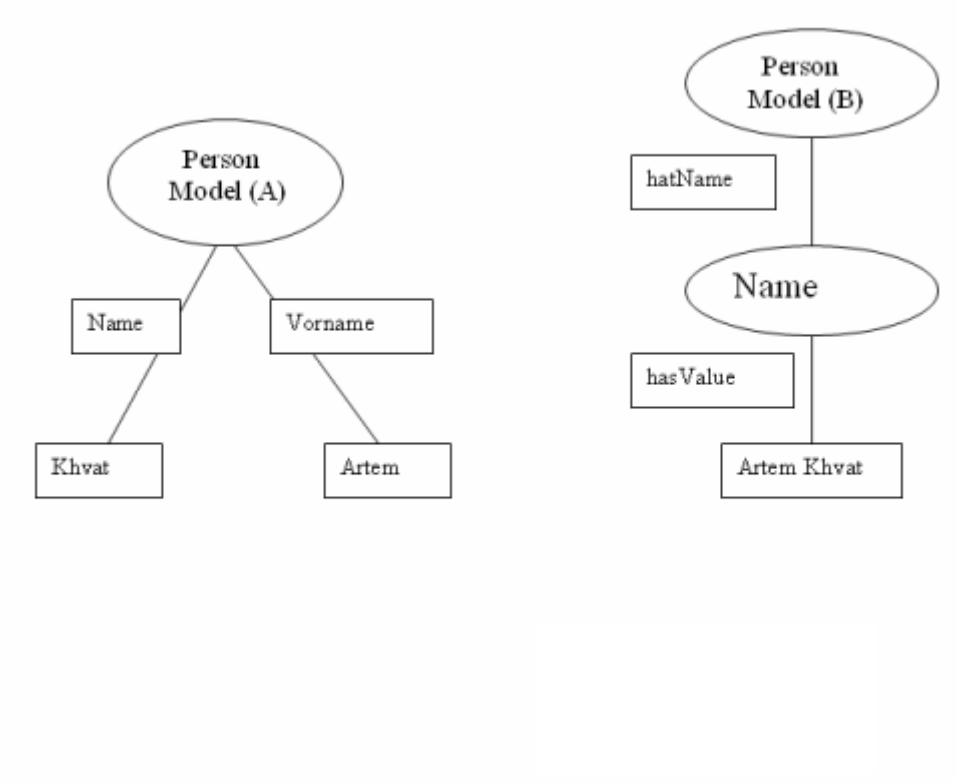


Abbildung 3.4.: Attribut-abstrakte topologische Modellierungsunterschiede

- ***Abstrakte Modellierungsunterschiede :*** Hier geht es um den Fall, bei dem das Konzept durch unterschiedliche Klassen und Klassen-Hierarchien bei zwei oder mehreren Domänen verschieden dargestellt wird. Die Klassenverteilung der Konzeptdarstellung darf nicht über die Grenzen einer Domäne hinausgehen. In Abbildung 3.5 wird ein Beispiel hierfür dargestellt, bei dem das Konzept Person durch unterschiedliche Klassen und deren Hierarchie dargestellt wird.

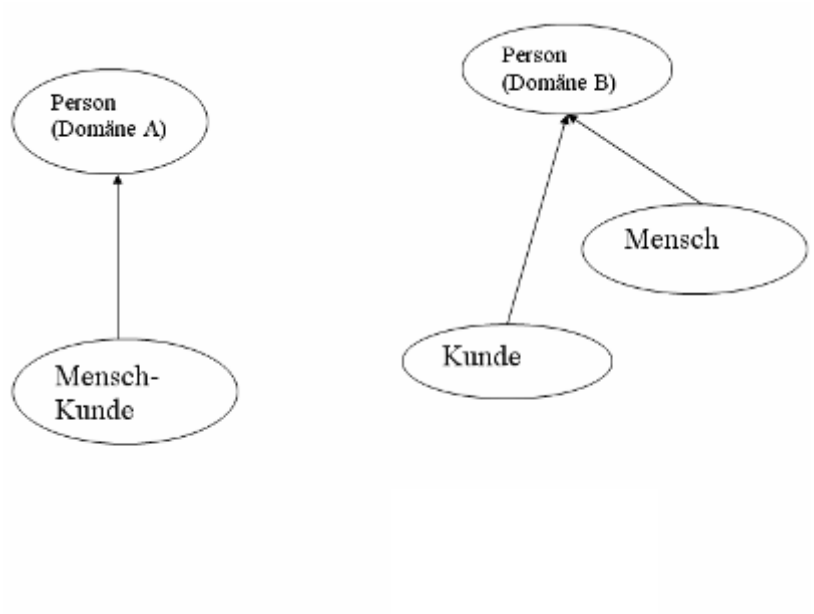


Abbildung 3.5.: Abstrakte Modellierungsunterschiede

- ***Konzept Heterogenität:*** Hier geht es um den Fall, bei dem das Konzept durch unterschiedliche Klassen und Klassen-Hierarchien in zwei oder mehreren Domänen verschieden dargestellt wird. Hierbei darf die Klassenverteilung der Konzeptdarstellung über die Grenzen einer Domäne gehen. In Abbildung 3.6 wird so ein Fall dargestellt.

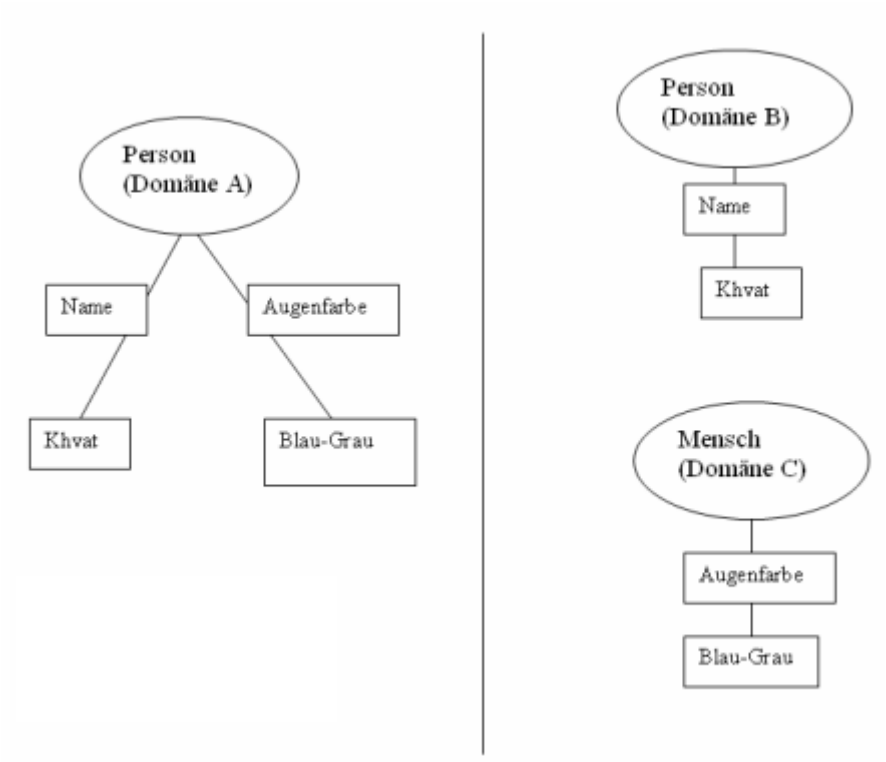


Abbildung 3.6.: Konzept Modellierungsunterschiede

## 3.2. Anforderungsanalyse

In diesem Abschnitt wird eine Analyse der Anforderungen für das zu implementierende System durchgeführt. Dabei ist die Analyse auf die wesentlichen Anforderungen, die vom System erfüllt werden müssen, begrenzt. Des weiteren werden zusätzlich mögliche Lösungsansätze zu auftretenden Problemen vorgestellt, wobei Implementierungsdetails allerdings ausgelassen werden.

### 3.2.1. Integration der neuen Dienste

Auf Grund der inhärenten Dynamik des Systems sind Mechanismen zur Verfügung zu stellen, die in der Lage sind neue Dienstanbieter in das System zu integrieren und vorhandene Dienstanbieter zu verwalten. Im folgenden werden die konkreten Aufgaben dieser Mechanismen beschrieben.

- ***Erkennung von neuen Mitgliedern:*** Soll ein neuer Dienst in das System integriert werden, stellen sich zwei grundsätzliche Fragen nach dem Auffinden dieses Dienstes im neuen System. Zum einen muss der Dienstanbieter die Adresse einer Vermittlungsstelle wissen um sein Angebot verfügbar zu machen. Zum anderen müssen Nutzer von Diensten das Erscheinen des neuen Dienstes mitbekommen und in der Lage sein ihn aufzufinden. Sind diese Dinge geklärt muss eine Beschreibung des Angebots hinterlegt werden, die von den Dienstnutzern verstanden werden kann. Eine mögliche Lösung für diese Situation stellen Web Services [Kapitel 2.8] dar. Sie ermöglichen einerseits das Beschreiben von Angeboten und andererseits das gegenseitige Auffinden von Dienstanbieter und Vermittlungsstelle. Für letztere Aufgabe könnte ein Verzeichnisdienst mit der UDDI-Technik [13] , welche ein Bestandteil der Web Services Technologie ist, aufgesetzt werden.
- *Dienste verlassen das System:* Verlässt ein Dienstanbieter das System muss dies von der Vermittlungsstelle erkannt werden. Dies kann z.B. durch eine erzwungene Abmeldung des verlassenden Dienstes vom System erreicht werden. Aufgrund dieser Benachrichtigung kann darauf die Vermittlungsstelle ihren Datenbestand aktualisieren um keine falschen Informationen über die vorhandenen Dienste weiterzugeben. Ein weiteres Problem ist allerdings das temporäre Verlassen eines Systems. Letzteres kann durch den temporären Ausfall eines Dienstes stattfinden. In diesem Fall wäre es von Vorteil wenn der Dienst nicht sofort automatisch aus dem System entfernt werden würde, da er ansonsten die vielleicht aufwändige Anmeldeprozedur wiederholen müsste. Eine Lösung würde hier ein zeitlicher Schwellwert sein, der nur bei Überschreitung zu einem Entfernen des Dienstes aus dem System führen würde. Die Vermittlungsstelle speichert Daten über die Historie des Dienstanbieters, solange der

Schwellwert nicht überschritten wird. Diese Daten können bei einem erneuten Anmelden den Dienstes zur Beschleunigung der Anmeldeprozedur genutzt werden. Dies Verhindert eine unnötige Ressourcenverschwendung und beugt gleichzeitig einer böswilligen Stilllegung des Systems vor.

- *Kommunikation zwischen einzelnen Diensten:* Die Kommunikation zwischen den einzelnen Systemmitgliedern sollte auf einer einheitlichen Basis stattfinden. Die Entwicklung eines eigenen Protokolls oder eines eigenen Kommunikationsbusses wäre für solche Zwecke nicht sinnvoll. Web Services [Kapitel 2.8] stellen hier eine geeignete Alternative dar, weil sie sich in den letzten Jahren gut bewährt haben und eine ständige Weiterentwicklung und Unterstützung von führenden Softwareherstellern und der W3C [26] erfahren.
- *Verwaltung von Diensten:* Die wichtigen Interaktionen zwischen einzelnen Diensten erfordern eine Speicherung der Interaktions- und Dienstanbieterdaten. Die rechtfertigt den Einsatz eines Datenbanksystems, welches vor allem zur Speicherung der semantisch annotierten Daten genutzt werden sollte.
- *Sicherheit:* Bei Fragen der Sicherheit sollte auf einen einheitlichen Satz von Sicherheitsrichtlinien gesetzt werden. Dies ist allerdings ein Problem, da jeder Dienstanbieter höchstwahrscheinlich seine eigenen Vorstellungen von Sicherheit hat. Von zentraler Bedeutung ist eine einheitliche Unterstützung der geforderten Sicherheitsrichtlinien der Vermittlungsstelle. Eine Möglichkeit um dies zu erreichen ist die Veröffentlichung einer Anforderungsliste durch den Vermittlungsdienst. Die Registrierung eines neuen Dienstes erfolgt dann nur noch bei Gewährleistung aller geforderten Sicherheitsrichtlinien.

Zusätzlich der obigen Anforderungen kommen neue Anforderungen in Bezug auf Integration und Verwaltung auf das System zu. Dies fordert eine offene und modulare Architektur des Systems. Um diese zu Verwirklichen wird eine spezielle Zwischenschnittstelle entwickelt, wodurch die Erweiterung der Funktionalität durch Modulersetzung ermöglicht werden soll. Für die Entwicklung der Schnittstelle ist es ausserdem möglich die bekannten Entwurfsmuster für Software einzusetzen.

### 3.2.2. Beseitigung der Heterogenität

In einem verteilten System ist es unvermeidlich, dass die unterschiedlichen Dienstanbieten und die Vermittlungsstelle alle einzigartige Vorstellungen von ihren Anwendungsbereichen haben. Dies rechtfertigt die Existenz eines semantischen Arbeiters der die Aufgabe hat, genau diese semantischen Unterschiede zu kapseln und sie jedem Teilnehmer verständlich zu machen. Im Wesentlichen handelt es sich um die Kapselung der in Kapitel 3.1 beschriebenen

Arten der semantischen und strukturellen Modellierungsunterschieden. Der semantische Arbeiter ist dazu in erster Linie auf die Fähigkeit des Merges von zwei oder mehr Ontologien angewiesen. Nach dem Merge muss die resultierende Ontologie vom System gespeichert und verwaltet werden können. Des Weiteren ist eine visuelle Unterstützung für den oben schon erwähnten manuellen Merge von Ontologien erforderlich. Letzterer wird mittels einer graphischen Benutzerschnittstelle realisiert.

### 3.2.3. Automatisierung des Merges

Der Grad der Automatisierung ist kritisch für den Anteil der manuellen Arbeit in Bezug auf den manuellen Merge von Ontologien. Um die manuelle Arbeit zu minimieren, ist es das Ziel Algorithmen für die Erkennung von topologischen, syntaktischen und semantischen Ähnlichkeiten zwischen den einzelnen Elementen der beteiligten Ontologien zu entwickeln.

### 3.2.4. Support/Advice System

Da der Anteil an manueller Arbeit höchstwahrscheinlich nicht restlos eliminiert werden kann, muss der manuelle Merge nach besten Kräften unterstützt werden. Hierzu soll ein Support/-Advice System entwickelt werden. Letzteres hat die Aufgabe den Benutzer mit hilfreichen Informationen über den Merge, wie z.B. verschiedene mögliche Alternativen, zu versorgen.

### 3.2.5. Lernsystem (Cyc)

Zum Start des Systems wird eine Person der Hauptentscheidungsträger bei der Integration von neuen Dienstanbietern in das System sein. Sie hat konkret die Aufgaben der Wissensakkumulierung und Transformation von Daten. Wächst das System, so ist irgendwann eine kritische Größe erreicht, bei der ein manuellen Eingreifen nicht mehr ausreicht. Ab hier soll das System in immer größeren Teilen diese Arbeit übernehmen. Dieser Ansatz wird gewählt, da die Größe des akkumulierten Wissens direkten Einfluß auf den Automatisierungsgrad der Integration von neuen Dienstanbietern hat. Der oben verwendete Ansatz wird als Cyc Methode [6] (Seite 111) bezeichnet, welche Mitte der 80er Jahre von der Computer Technology Corporation entwickelt wurde und in genauer beschrieben wird.

### 3.2.6. Ergonomie des Systems

Der Faktor der Ergonomie ist kritisch für die Produktivität des Systems im Falle des manuellen Merges. Unglücklicherweise müssen die Qualifikationen in dem Bereich des ontological engineering überdurchschnittlich hoch sein. Dies rechtfertigt den Versuch der Entwicklung einer Benutzerschnittstelle, welche die Komplexität geschickt vor dem Benutzer versteckt. Dadurch kann das Qualifikationsniveau des bedienenden Personals gesenkt werden.

## 3.3. Analyse der Entwicklungsumgebung

In der Vorbereitungsphase dieser Arbeit wurde eine Untersuchung der möglichen Werkzeuge durchgeführt. Eine sehr wichtige Anforderung war, dass das ausgewählte Werkzeug auf der Programmiersprache Java basieren sollte. Der Grund dafür war die große Erfahrung des Autors mit dieser Sprache. Als Ergebnis dieser systematischen Suche waren die beiden Frameworks Protégé 2000 von der Stanford University (USA) und Jena 2.0 vom HP Research Lab Bristol (UK). Die beiden Frameworks sind weit verbreitet und haben sich als mächtige Mittel für die Lösung der unterschiedlichen Aufgaben und Problemstellungen im Bereich des Semantic Webs und insbesondere des Ontologie Engineerings bewährt. Diese beiden Frameworks werden hier kurz vorgestellt. Dabei wird nicht auf die komplette API sondern nur auf die wesentlichen Merkmale eingegangen.

### 3.3.1. Protégé

Protégé ist:

- ein Werkzeug um Ontologien zu erstellen und Daten in diese einzutragen
- OpenSource
- eine java-basierte Plattform, die um Plug-Ins erweitert werden kann
- eine Bibliothek die von anderen Anwendungen verwendet werden kann um Daten einzusehen und diese zu manipulieren
- ursprünglich zur Wissensrepräsentation in der medizinischen Informatik von der University of Stanford entwickelt worden

Protégé bietet:

- Erstellung und Manipulation von Ontologien
- Erstellung von Informationen zur weiteren Verwertung

- Lesen, Schreiben und Interkonvertierung folgender Formate:

  Relationale Datenbanken (ODBC)

  UML / XMI [33]

  XML / XML Schema [14]

  RDF [16]

  Topic Maps [5]

  DAML+OIL [22]

  OWL [21]

- Eine benutzerfreundliche GUI (engl. Graphical User Interface, Abk. GUI)
- Mehrbenutzerfähigkeit
- Ein Protege-Server, mehrere Clients, Wissensdatenbank liegt auf Server, Abgleich mit Clients in Echtzeit
- Echtes verteiltes Arbeiten an einer Wissensdatenbank

#### Pro und Contra

Ein wesentlicher Vorteil von Protégé ist offensichtlich die Vielfalt der vorhandenen Lösungen, die mit Hilfe von Protégé erstellt werden können. Protégé bietet ein mächtiges API welches für die Lösung von sehr anspruchsvollen Aufgaben konzipiert wurde. Wie oben schon erwähnt, bietet Protégé eine Plattform, die Eclipse sehr ähnlich ist. Diese kann durch Plug-Ins beliebig erweitert werden. Ein Vorteil dabei ist, dass man auf sehr einfache Weise durch geschickte Kombination der Plug-Ins eine sehr effiziente Lösung erschaffen kann, ohne dabei großen Implementierungs-Aufwand betreiben zu müssen. Es gibt aber ein Problem, welches die Arbeit mit Protege, auf Grund des sehr großen Lösungspotenzials, sehr erschwert. Man braucht eine sehr gute Dokumentation um die Einarbeitung möglichst schnell zu ermöglichen. Diese ist bei Protégé nicht gegeben. Sorgfältige Untersuchungen des Autors ergaben, dass fast keine Dokumentation für Protégé 2000 vorhanden ist. Dadurch wird ein schneller Einstieg verhindert und die Arbeit mit Protégé folglich erschwert. Diese Restriktion war für den Autor aufgrund des knappen Zeitrahmens für das Projekt nicht akzeptabel.

### 3.3.2. Jena

Jena ist ein Java Framework für die Erstellung von Semantic Web Applikationen. Es hat folgende Funktionen:

- RDF API für die Generierung und Verarbeitung von RDF-Dokumenten
- Unterstützung von diversen DB-Systemen z.B. Oracle
- Reasoning Subsystem für RDFS[20], OWL/Lite [21] und Teile von OWL/Full [21]
- Ontology Subsystem für die Unterstützung bei der Arbeit mit RDFS [20], OWL [21], DAML+OIL [22]
- Unterstützung der Sprache RDQL, für die Erstellung von Anfragen an Ontologien

#### Pro und Contra

Bei der späteren Arbeit mit Jena 2.0 hat sich herausgestellt, das Jena für manche Arten von Aufgaben nur Grundfunktionalitäten bietet. Dies führt wegen eigener Implementierungsarbeit zu einem Zusatzaufwand bei anspruchvolleren Aufgaben. Ein sehr wichtiger Vorteil bei Jena 2.0 ist die ausführliche Dokumentation mit vielen nützlichen Beispielen und Tutorien. Ein weiterer sehr wichtiger Vorteil von Jena ist die ständige Weiterentwicklung und Verbesserung des Produkts durch das HP Research Lab Bristol (UK). Die Entwickler haben eine große Bereitschaft zur Zusammenarbeit gezeigt, und waren sehr sachlich und hilfsbereit bei Supportanfragen des Autors.

Aus diesen Gründen ( insbesondere wegen der Dokumentation ), hat sich der Autor für den Einsatz von Jena 2.0 als Entwicklungswerkzeug für das Projekt entschieden.

## 3.4. Analyse existierender Softwarelösungen

In diesem Abschnitt werden vorhandene Softwareprodukte, die sich auf dem Mergen von Ontologien widmen analysiert. Die Analyse beschränkt sich auf die Vorstellung der wesentlichen Eigenschaften dieser Lösungen. Es wurden insgesamt drei produkte untersucht, welche sich in den letzten Jahren etabliert haben.

### 3.4.1. PROMPT (Protege 2000)

PROMPT wurde von der Stanford University als ein Plug-In für die Protege 2000 Plattform entwickelt. Es folgt eine Aufzählung der mit PROMPT möglichen Operationen auf Ontologien mit Erläuterungen.

- *Versionsvergleich von Ontologien und Erstellung einer neuen Version:* In diesem von PROMPT bereitgestellten Modus wird der Benutzer in die Lage versetzt zwei Versionen der gleichen Ontologie miteinander zu vergleichen. Dazu benutzt PROMPT einen Satz von Heuristiken zum Auffinden der miteinander korrespondierenden Frames beider Versionen. Dabei werden auch Strukturvergleiche zwischen den beiden Versionen durchgeführt. Z.B. ist ein ungemapptes Blatt derselben Eltern in beiden Versionen sehr wahrscheinlich dasselbe Frame nur mit unterschiedlichen Namen. Ein Vergleich kann ein gutes Entscheidungsmerkmal bei der Versionskontrolle in Zusammenhang mit der Suche nach Differenzen zwischen zwei Projekten sein.
- *Framemigration zwischen Projekten:* Protege erlaubt eine modulare Wiederverwendbarkeit von Teilen eines Projektes. Hierbei dient PROMPT als Mechanismus für die Migration und Integration von einzelnen Frames. Der Vorgang kann mittels PROMPT [9] im Hinblick auf die Art der Migration und die Tiefe des Kopiervorgangs gesteuert werden. Bei letzterem ist z.B. die shallow copy und die deep copy möglich. Zusätzlich werden von PROMPT auch die jeweils bei der Migration auftretenden Konfliktsituationen überwacht.
- *Das Mergen zweier Ontologien in eine neue Ontologie:* Der sog. Merge ist die Hauptaufgabe von PROMPT.

  Hierzu definiert und unterstützt es eine vollständige Mergemethode. Letztere wurde bereits im Abschnitt Merge vorgestellt.
- *Extraktion von Teilen einer Ontologie in eine andere:* Oft sollen nur bestimmte Teile von Ontologien für bestimmte Arbeiten verwendet werden. Für diese Zwecke eröglicht PROMPT [9] das Extrahieren von Teilen einer Ontologie, welche dann als separate Ontologie gespeichert werden können. Ein möglicher Anwendungsfall für diese Fähigkeit wäre das Debuggen von großen Ontologien, indem sie in kleinere domänenspezifische Ontologien aufgeteilt werden.

### 3.4.2. Chimera

Chimera ist ein Mergewerkzeug und Teil des Ontologieservers [6] ( Seite 296) . Es erlaubt das mergen von zwei oder mehreren Ontologien, welche der beliebten OKBC-Form [6] (Seite 201) genügen . Des weiteren werden verschiedene Formate wie z.B. Ontoliguan [6] (Seite

48), KIF [6] (Seite 48), Protege-2000 [3], DAML+OIL [22]. Der Merge selber ist semiautomatisch wobei folgende Automatismen unterstützt werden.

- Erstellung von Listen mit Äquivalenzen zwischen Termen der beteiligten Ontologien
- Generierung von Empfehlungen für Mergekandidaten
- Unterstützung von Basisoperationen an Ontologien wie z.B. die Erstellung einer Subklasse, das Kopieren der einzelnen Klassen mit unterschiedlicher Tiefe oder der Merge von zwei Klassen.

### 3.4.3. Observer

Observer ist ein Mergewerkzeug für das Mergen von heterogenen Beschreibungsontologien der gleichen Domäne. Die Mergekriterien sind dabei die Synonym-, Hyponym- und Hypernymbeziehungen innerhalb von Ontologien. Der Merge passiert vollautomatisch und wird in folgenden Schritten abgewickelt.

- Der Name der Ontologie wird zu den Namen der einzelnen Terme innerhalb der Ontologie hinzugefügt um eine Unterscheidung von gleichnamigen Termen zu ermöglichen.
- Bearbeitung von Synonymbeziehungen: Sind zwei Terms durch eine Synonym verbunden, wird nur einer der beiden in die neue Ontologie übernommen und alle anderen werden ersetzt.
- Bearbeitung von Hyponymbeziehungen: Wird eine Hyponymbeziehung zwischen zwei Termen nachgewiesen (*T1 ist Hyponym zu T2*), so ist dies ein Indiz für eine *subclass-Of* Beziehung (*T1 ist subclass-Of T2*).
- Bearbeitung der Hypernym-Beziehungen: Wird eine Hypernymbeziehung zwischen zwei Termen nachgewiesen (*T1 ist Hypernym zu T2*), ist dies ein Indiz für eine *subclass-Of* Beziehung (*T1 ist subclass-Of T2*).

### 3.4.4. Fazit

Keine der vorgestellten Lösungen bittet vollautomatische Verarbeitung der Ontologien. Es wird immer noch für die meisten Aufgaben einen manuellen Angreifen notwendig. Der Grad der Automatisierung ist abhängig von dem, wie gut die Ontologien vorher modelliert und beschrieben wurden.

# 4. Design und Realisierung

## 4.1. Gesamtarchitektur

Für den Entwurf des Systems wurde eine klassische drei Schichten Architektur [8] gewählt, welche in Abbildung 4.1 dargestellt ist.

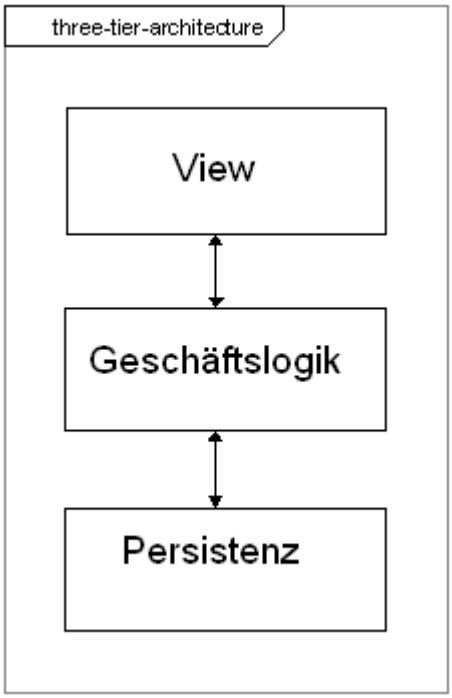


Abbildung 4.1.: three-tier-architecture

Der View ist für die Darstellung aller Daten und für die Dialogführung zuständig.

In der ***Geschäftslogik*** wird die Trennung zwischen dem Kern der Anwendung, Speicherung und der Präsentation realisiert. Der Informationsaustausch mit der Benutzungsoberfläche wird durch Standardmethoden (setAttribut und getAttribut) übernommen. Der Controller bekommt kein Wissen über die Benutzerschnittstelle zur Verfügung gestellt. Das bedeutet, dass Änderungen letzterer keinen Einfluss auf den Controller haben.

Die ***Persistenzschicht*** ist für die Datenspeicherung, z.B. in einer Datenbank zuständig. Die Benutzerschnittstelle weiß nichts über den Datenzugriff. Diese Architektur hat ihre Qualität in der Praxis bereits unter Beweis gestellt . Ihr Einsatz bietet folgende Vorteile.

- Schärfere Trennung von Datenhaltung, Geschäftslogik und Bedienungslogik und damit eine bessere Trennung von Fachlichkeit und Technik.
- Serverseitige Fachlogik erleichtert die Interaktion mit bestehenden Systemlandschaften und somit die Migrations- und Interoperabilitätsmöglichkeiten.
- Serverseitige Fachlogik verbessert die Skalierbarkeit und die Verfügbarkeit von Anwendungslogik bzw. Anwendungsmodulen.
- Durch serverseitige Fachlogik wird der Client unabhängiger. Damit werden die Stabilität und die Einsatzmöglichkeiten vergrößert.
- Drei Schichten Architekturen verbinden die Vorteile von zentralisierten und dezentralisierten Systemen.

In Abbildung 4.1 spiegeln sich nicht alle Anforderungen wieder, die im Rahmen dieser Arbeit an das System gestellt werden. Z.B. fehlen die Sicherheits- und Kommunikationsanforderungen. Aus diesem Grund zeigt Abbildung 4.2. die Anpassungen der Architektur, die auf Grund der zusätzlichen Anforderungen nötig waren.

Abbildung 4.2 stellt die Gesamtarchitektur des Systems unter Berücksichtigung der Kernideen der klassischen Drei Schichten Architektur dar. Ersichtlich ist, dass das System aus folgenden Modulen besteht:

**View** *Admin-GUI:* Dieses Modul übernimmt Aufgaben der Interaktion zwischen dem Administrator und dem System. Das kann in Form GUI, Report Erstellung, diversen Benachrichtigungssystemen realisierte werden.

**Geschäftslogik** *Kommunikationsmodul:* Dieses Modul übernimmt Aufgaben der Kommunikation mit neuen und bestehenden Mitgliedern des Systems. Die Aufgaben der Anmeldung von neuen Mitgliedern sind auch im Kompetenzbereich dieses Moduls anzusiedeln. Das Modul soll die unterschiedliche Kommunikationsmechanismen unterstützen, wie z.B. Internet via Web- Services.

*Controllermodul:* Dieses Modul übernimmt Aufgaben der Transaktionsverwaltung und steuert den Ablauf der internen Protokolle. *Logik Modul:* Das Modul übernimmt die Aufgaben der Realisierung der speziellen Algorithmen und Abläufe z.B. wie dem Merge zweier Ontologien.

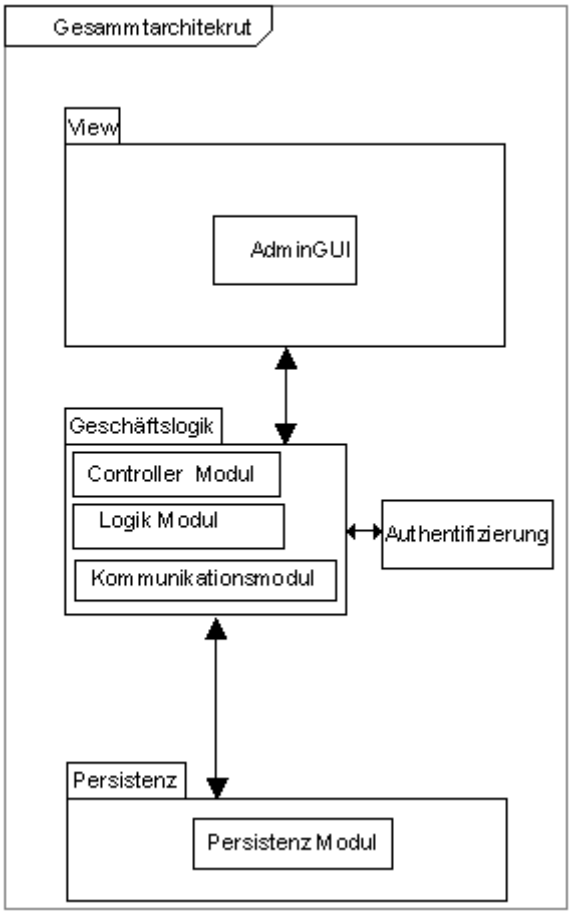


Abbildung 4.2.: Gesamtarchitektur

**Persistenz** *Authentifizierung:*Dieses Modul übernimmt die Aufgaben der Verwaltung von Zugriffsrechten von außen und innen. Zusätzlich wird auch hier die Festlegung der Sicherheitskriterien implementiert.

**Persistenz** *Persistenz Modul:* Dieses Modul übernimmt die Aufgaben der Aufbewahrung und Verwaltung diverser Daten die für das System von Bedeutung sind.

Jede neue Aufgabe die von dem System unterstütz wird, soll auf diese Module verteilt werden. Dies unterstützt die Modularität und Wiederverwendbarkeit innerhalb des Systems. Es erleichtert und spezialisiert die Kommunikation zwischen den einzelnen Systemaufgaben. Letztere braucht nicht auf allen Ebenen implementiert zu werden. Stattdessen ist nur eine Schnittstelle zwischen den einzelnen Aufgaben in relevanten Modulen zu implementieren. Der Rest der Architektur bleibt unverändert.

Zur Veranschaulichung des obigen Konzepts, betrachten wir folgendes Beispiel. Es wird eine Lösung für die An- und Abmeldung der Mitglieder des Systems benötigt. Diese soll nun in das bestehende System integriert werden. Dazu muss diese Aufgabe in kleinere Stücke unterteilt werden. Die Aufteilung muss so erfolgen, dass die einzelnen Teilaufgaben in die Kompetenz eines der Systemmodule fallen.

In der *Admin-GUI* werden die Funktionalitäten und Workflows untergebracht, die für die Visualisierung und Benachrichtigung des System-Administrators zuständig sind. Dies kann in diesem Fall die Benutzerschnittstelle sein. Im Kommunikationsmodul werden die Abläufe der Kommunikation mit der Außenwelt implementiert wie z.B. Web Services oder RPC Aufrufe von Corba.

Das *Authentifizierungsmodul* wird die gesamten Regeln für das Verhalten der Mitglieder beim Neueintreten verwalten, reglementieren und überprüfen. Es kann sein, dass die neuen Mitglieder Restriktionen auf Datenbankzugriffe erhalten und Premium Mitglieder bei ihrer Anmeldung bevorzugt werden.

Das *Controllermodul* überwacht die Abläufe der einzelnen Operationen und ihrer Reihenfolge. Hier wird u.a. auch die Vollständigkeit und Atomarität der Operationen kontrolliert und Verhalten für ein mögliches Rollback der Operationen definiert.

Das *Logikmodul* kann in diesem Beispiel die Einhaltung der Datenkonsistenz und die Überprüfung der semantischen Beschreibung der übermittelten Anmeldedaten der Mitglieder übernehmen. Hier werden auch die Reasoningalgorithmen für Ontologien untergebracht.

Das *Persistenzmodul* kann die Speicherung der Anmeldedaten übernehmen. Es kann je nach Komplexität und Größe der Aufgabe zwischen Datenbanken verschiedener Qualität und Mächtigkeit gewählt werden.

In folgendem Sequenzdiagramm wird ein mögliches Ablaufszenario der oben beschriebenen Aufgaben dargestellt Abbildung 4.3.

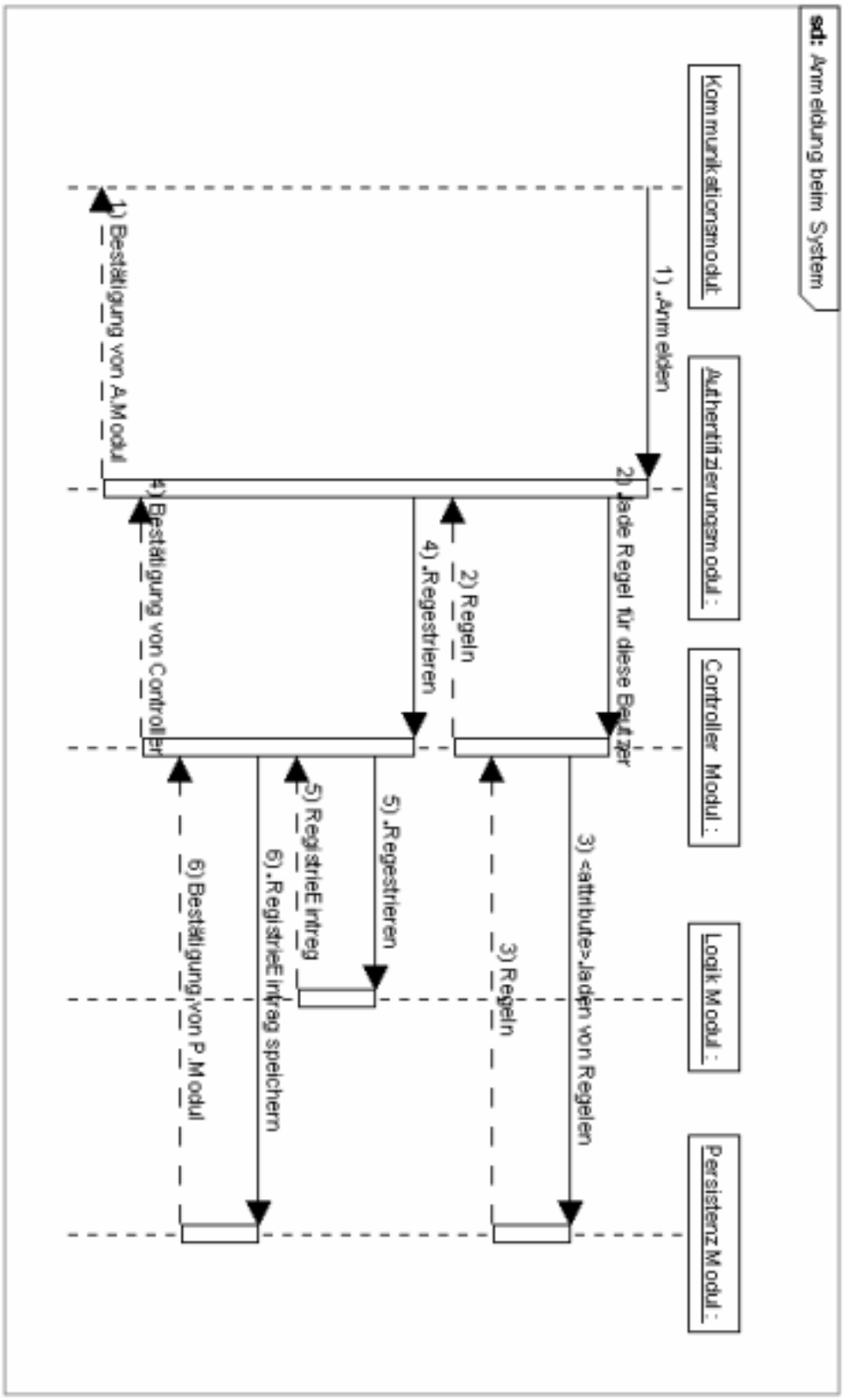


Abbildung 4.3.: Anmeldung beim System

**Anmeldung bei System** Der neuen Mitglieder will in System beitreten und Kommuniziert mit Kommunikations- Modul.

1. Das Kommunikationsmodul schickt die Registrierungsanforderung an das Authentifizierungsmodul.
2. Das Authentifizierungsmodul fordert den Regelsatz für diese Art von Benutzer vom Controller an.
3. Der Controller leitet die Anforderung an das Persistenz Modul weiter.
4. Das Persistenzmodul liefert die Antwort über den Controller an das Authentifizierungsmodul zurück.
5. Das Authentifizierungsmodul überprüft den Benutzer und fordert die Registrierung des Benutzers von Controller.
6. Der Controller fordert das Logikmodul auf einen Registrierungseintrag zu erzeugen.
7. Der Controller fordert das Persistenzmodul auf den Registrierungseintrag vom Benutzer zu speichern.
8. Der Controller schickt eine Bestätigung für die Registrierung an das Authentifizierungsmodul.
9. Das Authentifizierungsmodul leitet die Bestätigung der erfolgreichen Registrierung über das Kommunikationsmodul an den Benutzer weiter.

Wie man aus dieser Beschreibung entnehmen kann, ist die Architektur des Systems sehr modular gehalten. Das entspricht dem Anforderungen, die in Kapitel 3.2.1 an das System gestellt wurden. In den weiteren Kapiteln wird der Autor die einzelnen Systemmodule näher vorstellen.

## 4.2. Admin-GUI

Das Modul soll die Ergonomieanforderungen aus Kapitel 3.2 erfüllen. Kernbestandteil dieser war es die komplexen Systemaufgaben durch eine übersichtliche und verständliche Benutzerschnittstelle oder durch die Definition eines vereinfachten Workflows zu gestalten. Es ist möglich, dass bestimmte Aufgaben manuell noch besser gelöst werden können. Es ist der Verantwortung des Entwicklers überlassen, welche Kombination der vorhandenen Techniken

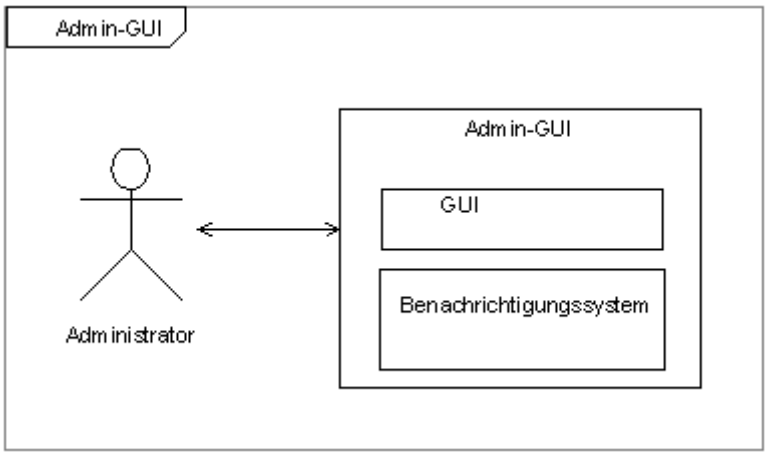


Abbildung 4.4.: Admin-GUI

hier sinnvoll erscheinen mag. Es muss nur berücksichtigt werden, dass auf Grund der Komplexität der Aufgaben eine sehr hohe Qualifikation der Mitarbeiter von Nöten ist. Dies kann zu hohen Kosten und am Ende dem Verzicht auf die jeweilige Lösung führen.

## 4.3. Kommunikationsmodul

Dieses Modul soll die Anforderungen aus Abschnitt 3.2.1 erfüllen. Es soll das Ansprechen des Systems ermöglichen, die Mechanismen der Kommunikation von außen und von innen realisieren, die Verwaltung der Kommunikationsprotokolle und die Transparenz zwischen den verschiedenen Kommunikationsschienen gewährleisten. Für die Kommunikation des Systems mit der Außenwelt schlägt der Autor den Einsatz von Web Services vor. Das wird dem Plattformunabhängigkeit verleihen. Der Einsatz von Web Services (Kapitel 2.8) wird zudem auch das Auffinden von Systemangeboten im Internet für Agenten erleichtern und die Verteilung der einzelnen Komponenten unterstützen. Es ist aber durchaus möglich, dass wegen der Komplexität der Realisierung eine andere Form der Kommunikation benötigt wird. Als mögliche Alternative bieten sich hier Lösungen wie Corba www.corba.org/ , ein HTTP-Server und ein einfaches Reporting per E-Mail oder ein Benachrichtigungssstem an. Es muss allerdings beachtet werden, dass diese alternativen Lösungen bestimmte technische Restriktionen auf die Vollständigkeit der Realisierung der angestrebten Anforderungen (Abschnitt 3.2.1) setzen. Eine dritte Alternative Lösung wäre nach Meinung der Autors die eigene Implementierung des gesamten Kommunikationsmoduls. Das erlaubt die Feinabstimmung der Lösung in Bezug auf die Bedürfnisse des Systems, fordert allerdings von allen Mitgliedern die Unterstützung des neuen Standards, was eventuell zu einer erschwerten Verbreitung des Systems führen wird.

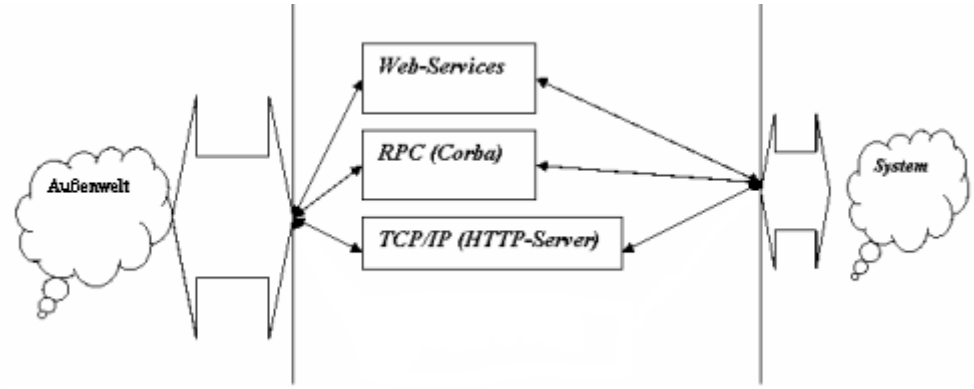


Abbildung 4.5.: Kommunikationsmodul

## 4.4. Authentifizierungsmodul

Das Authentifizierungsmodul soll die Anforderungen aus dem Kapitel 3.2.1 (Sicherheit) erfüllen. Dies schließt Fragen der Sicherheit, die Regeln für die An- und Abmeldung, die Überprüfung der Unterstützung der Sicherheitsanforderungen einzelner Mitglieder des Systems und die Verwaltung der Zugriffsrechte mit ein. Auf Grund dieser Besonderheiten und der vielfältigen Anforderungen ist die Konstruktion einer allgemeinen Lösung ausgeschlossen. Folglich ist der Entwickler gezwungen, seine eigene spezielle Lösung zu implementieren. Es ist aber durch aus möglich, einige konzeptionelle Lösungen aus Systemen mit ähnlichen Sicherheitsanforderungen zu benutzen. Dazu gehören die Verwendung von Verschlüsselungstechniken, Richtlinien für die Benutzerverwaltung in großen Domänen und die Anwendung von Architekturen wie Kerberos http://de.wikipedia.org/wiki/Kerberos_(Informatik).

## 4.5. Controllermodul

Das Controllermodul ist zuständig für die Erfüllung der Anforderungen aus Kapitel 3.2.1. Hier werden die Algorithmen für die Steuerung und die Transaktionsverwaltung der internen Protokolle für systemmodulübergreifende Aufgabe implementiert. Wie man in Abbildung 4.1 sehen kann, ist das Controllermodul eine Art Systemkoordinator. Benötigt eine Systemaufgabe (z.B. der Merge zweier Ontologien) für ihre Lösung die Ressourcen von mehreren Systemmodulen (in diesem Fall sind es die Admin-GUI , das Logikmodul und das Persistenzmodul ), so übernimmt das Controllermodul die Ablaufüberwachung und die Koordination des Zusammenspiels zwischen den einzelnen Systemkomponenten. Die Einführung des zentralen Überwachers ermöglicht es die Atomarität und die Vollständigkeit von Systemtransaktionen zu garantieren. Der Entwickler wird an dieser Stelle herausgefordert, eine eigene Lösung zu implementieren, die einzigartige Merkmale des Systems berücksichtigen muss.

## 4.6. Logik Modul

Das Logikmodul soll die Anforderungen aus dem Kapitel 3.2.2 erfüllen. Es soll die Automatisierung des Merges, die Beseitigung der Heterogenität, das Support/Advice System und die Algorithmen des Lernsystems implementieren. Als Entwicklungsplattformen bieten sich für die Lösung dieser Aufgaben die in Kapitel 3.3 vorgestellten Frameworks und die in Kapitel 2.6 erwähnte Abfragesprache an. Im Rahmen dieser Arbeit werden einige der im Kapitel 3.2 angesprochenen Anforderungen vom Autor realisiert. Die genauere Beschreibung der Realisierung wird in den weiteren Kapiteln stattfinden.

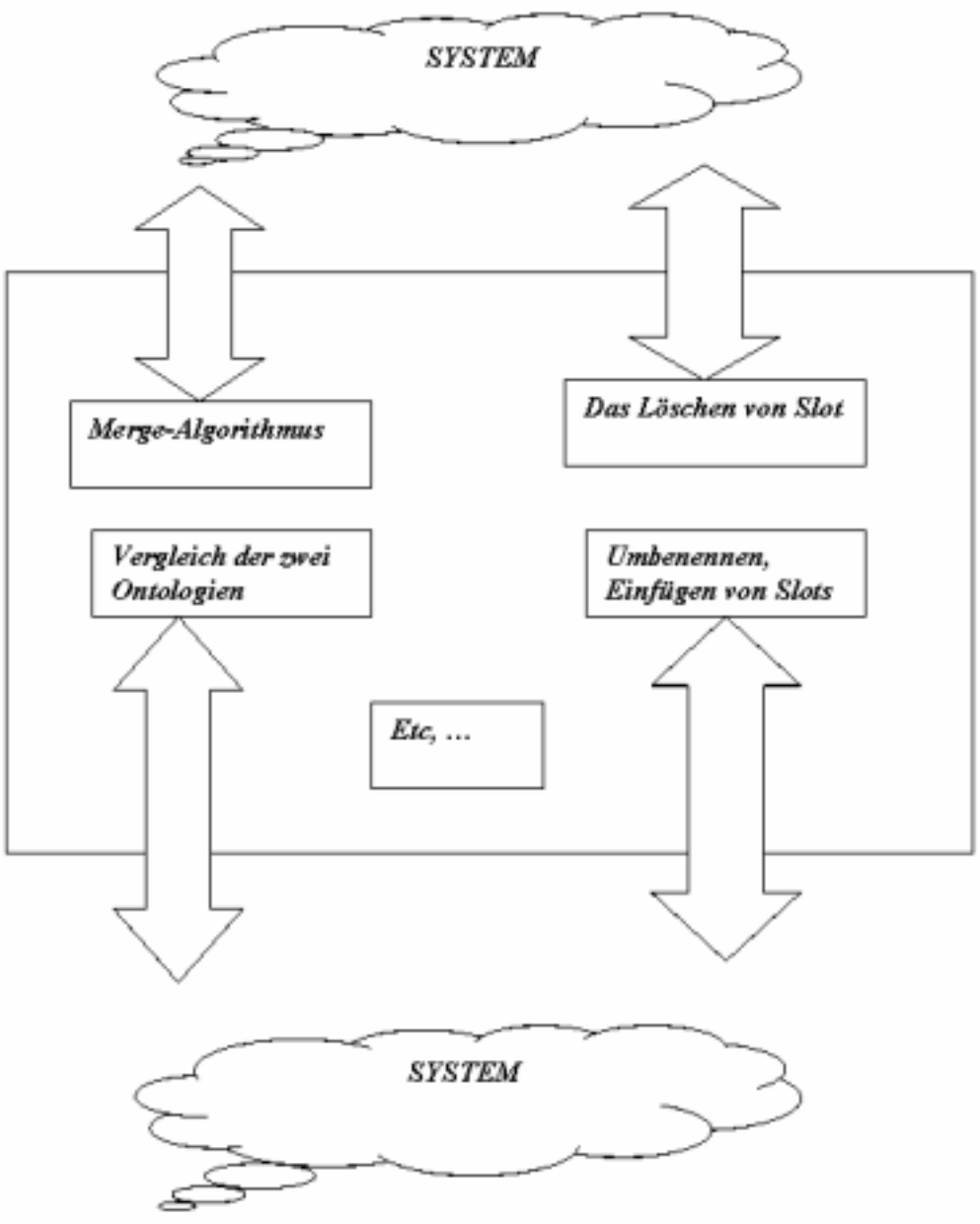


Abbildung 4.6.: Logik Modul

## 4.7. Persistenzmodul

Das Persistenzmodul soll die Anforderungen aus Kapitel 3.2 erfüllen. Es soll die Aufgaben der speziellen Datenaufbewahrung und Datenverwaltung erfüllen. Es wird notwendig parallel zu den gewöhnlichen Datensätzen auch Daten mit semantischen Beschreibungen in Form von RDF, RDF(S) oder OWL abzuspeichern. Für diesen Zweck eignet sich der Einsatz von speziellen DB-Systemen, die mit solchen Formaten umgehen können. Der Vorteil ist hier, dass kein Zusatzaufwand für ihre Verwaltung zu betreiben ist. Als mögliche Lösung bietet sich an dieser Stelle der Einsatz des 3Stone Servers an. 3Stone ist unter SourceForge frei verfügbar und kann etwa 25 Millionen RDF - Tripel speichern. Das Anzeigen der abgespeicherten Inhalte der Datenbank wird durch den 3Stone Browser ermöglicht. Diese Lösung hat sich bereits bei dem Projekt CS AKTive Space http://cs.aktivespace.org/ bewährt, welches als großer Erfolg im Jahre 2003 gefeiert wurde (Der Server hat die Web Challenge 2003 gewonnen).

## 4.8. Realisierung der wichtigsten Abläufe

### 4.8.1. Vergleich

In dieser Arbeit wurden drei Arten des Vergleichs von Ontologien realisiert. Alle diese Arten haben als Ziel die Erkennung von Ähnlichkeiten oder Unterschieden zwischen zwei Ontologien.

***Difference:*** Es werden zwei Ontologien verglichen, als Ergebnis bekommt man alle Elemente die in beiden Ontologien unterschiedlich sind. Das Verfahren sieht wie folgt aus: Man travestiert beide Bäume der Ontologien und erstellt eine Liste mit allen Elementen wobei zwischen Prädikaten und Ressourcen unterschieden wird. Nachdem die Liste vollständig überprüft wurde, werden die gleichen Elemente (Elemente welche den gleichen URI haben) aus der Liste gelöscht. Das Ergebnis wird dann in Form von der übrig gebliebenen Elemente dargestellt. Elemente, welche ihr Wurzelelement während der Operation verloren haben, werden an eine gemeinsame neu erzeugte Wurzel gehängt.

***Intersection:*** Es werden zwei Ontologien verglichen und als Ergebnis kriegt alle gemeinsamen Elemente der beiden Ontologien ausgegeben. Das Verfahren geht wie folgt vor: Man travestiert die beiden Bäume der Ontologien und erstellt eine Liste mit allen Elementen, wobei zwischen Prädikaten und Ressourcen unterschieden wird. Nachdem die Liste vollständig überprüft wurde, werden die unterschiedlichen Elemente (Elemente deren URI nicht in beiden Ontologien vorhanden ist) aus der Liste gelöscht. Das Ergebnis stellen dann die übrig

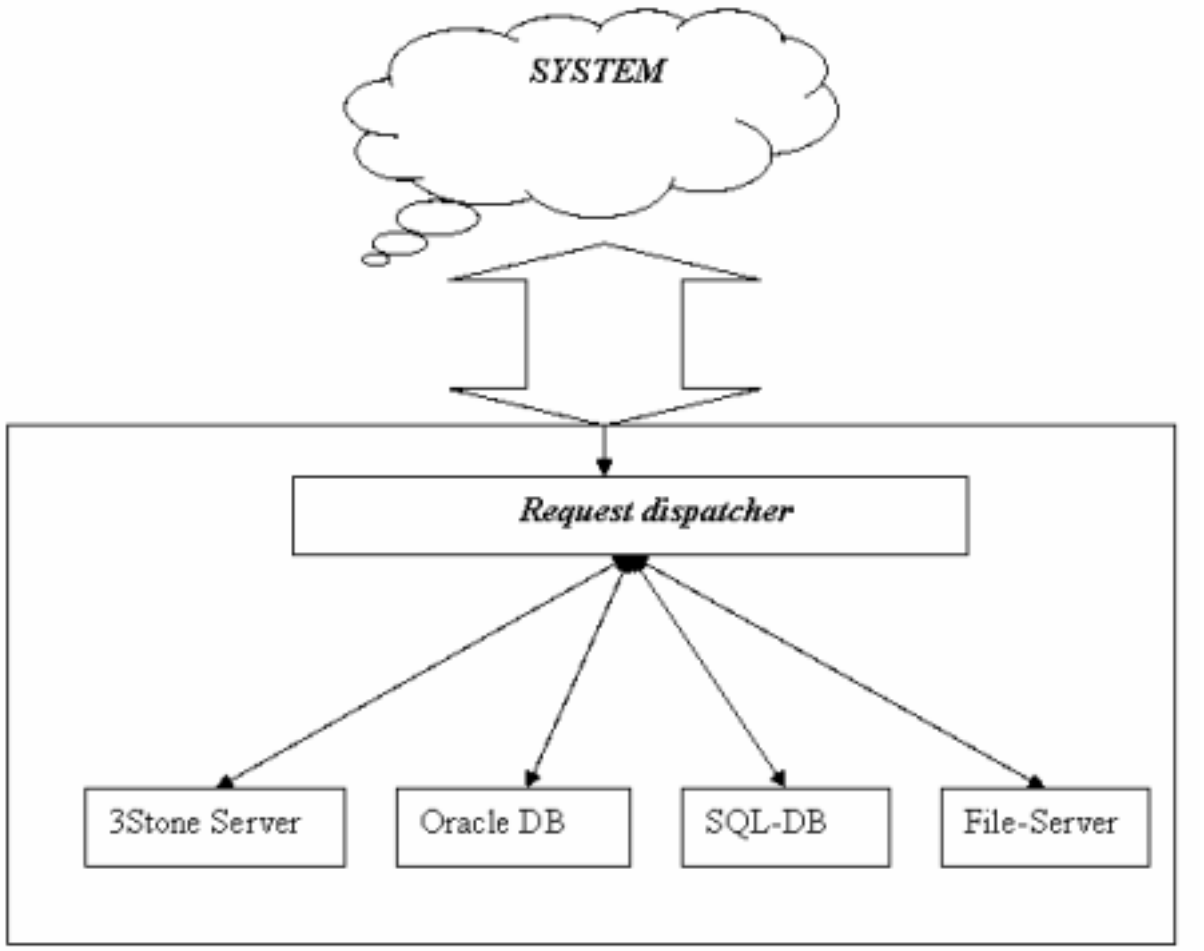


Abbildung 4.7.: Datenmanager Modul

gebliebenen Elemente dar. Mit Elementen, die ihre Wurzel während des Verfahrens verloren haben, wird wie oben bei der Differenzbildung verfahren.

***Topologischer Vergleich (topoChecker):*** Diese Arte des Vergleichs liefert Informationen über die Ähnlichkeiten der Strukturen beider Ontologien. Es werden nicht wie oben die syntaktischen Unterschiede im Bereich der Namensgebung, sondern stattdessen die Baumstrukturen beachtet. Das Ergebniss ist ein boolescher Wert, welcher die Gleichheit der beiden Baumstrukturen angibt.

### 4.8.2. Merge

Es werden zwei Ontologien verglichen wobei das Ergebnis die Zusammenführung beider Ontologien ist. Das Verfahren geht wie folgt vor: Man travestiert die beide Bäume der Ontologien und erstellt eine Liste mit allen Elementen wobei zwischen Prädikaten und Ressourcen unterschieden wird. Nachdem die Liste vollständig überprüft wurde, werden gleiche Elemente (Elemente deren URI vorhanden ist) zusammengeführt. Unterschiedliche Elemente bleiben dabei unverändert. Das Ergebnis ist die zusammengeführte Ontologie.

### 4.8.3. Zwischenmodulare Übersetzung von RDQL-Abfragen

In diesem Teil der Arbeit wird die Realisierung der wesentlichen Abläufe, die für die Übersetzung der zwischenmodularen RDQL-Abfragen notwendig sind, vorgestellt. Die wesentliche Aufgabe dieser Übersetzung ist die Abdeckung der in Kapitel 3.1 vorgestellten Arten der Modellheterogenität zwischen Ontologien. Die Hauptidee des Autors war eine Transparenzplattform für den jeweiligen Partner mit unterschiedlichen Weltmodellen zu schaffen. Der einfache Merge sieht das Erzeugen einer gemeinsamen Ontologie vor, welche die wesentlichen Merkmale der beteiligten Ontologien übernimmt. Das bietet aber nicht die Möglichkeit bestehende Informationen der beteiligten Systeme zu nutzen. Der Merge erlaubt die Speicherung von neuen Informationen in einem gemeinsamen Modell, welches für beide Partner verständlich ist. Ein transparenter Zugriff auf die schon bestehenden Daten wird allerdings nicht realisiert. Stattdessen ist man gezwungen, alle alten Datenbestände in das neue Merge-Modell umzuformen, was in erster Linie einen Zeitaufwand bedeutet. Es ist auch möglich, dass das neue Modell nur teilweise das bestehende Ausgangsmodell abdeckt. In diesem Fall kann es zu Informationsverlusten kommen. Die in Kapitel 3.4 vorgestellten Merge-Tools bieten nur die Möglichkeit für einen einfachen Merge. In dieser Arbeit soll allerdings ein Schritt weiter gegangen und ein Prototyp entwickelt werden, welcher eine Art semantische Transparenzplattforz für Ontologien mit unterschiedlichen Modellen darstellt. Für diesen Zweck, will der Autor einen zwischenmodularen Übersetzer implementieren. Dadurch soll keine neue Ontologie im Sinne des Merges erzeugt werden, um Daten zwischen den Partnern gemeinsam

zu nutzen. Den Partnern soll es ermöglicht werden, ihr jeweiliges Modell der Welt beizubehalten. Dabei ist es für die Partner nicht notwendig alle Modelle der Welt zu verstehen, außerdem werden die Anfragen von der Außenwelt auf Basis des Wissens über die eigene Welt beantwortet. Zusammengefasst bedeutet dies, dass die Partner bei der Erstellung der Abfragen nach Außen davon ausgehen können, dass alle Partner die gleichen Modelle der Welt haben. Dies zu erreichen, erfordert den Einsatz eines Vermittlers, dessen Aufgabe die Übersetzung zwischen diversen Weltvorstellungen ist. Es wird vom Partner verlangt, sich einmalig bei dem Vermittler anzumelden und die Übersetzer für ihre Modelle zu initialisieren. Alle Partner müssen nur die Initialisierung für die Modelle der Welt, die der Vermittler beherbert durchführen. Die Initialisierung wird teilweise manuell vorgenommen. Die Initialisierung garantiert die Übersetzung der Abfragen von Partnern. In Abbildung 4.8 wird ein Beispiel des Systemaufbaus dargestellt.

Um die Funktionsweise zu verdeutlichen, wird folgendes Beispiel betrachtet. Der neue Partner 1 will sich am System anmelden und anschließend eine Abfrage der Datenbestände der Partner 2 und 3 durchführen. Der Ablauf wird dann in folgenden Schritten passieren Abbildung 4.9.

1. Partner 1 führt die Initialisierung des Übersetzers durch.
2. Partner 1 erstellt eine Abfrage (auf Basis der eigenen Modelle).
3. Partner 1 schickt die Abfrage an den Übersetzer des Vermittlers.
4. Der Vermittler übersetzt die Anfrage in sein eigenes Modell.
5. Der Vermittler schickt die übersetzte Anfrage an den Übersetzer der Partner 2 und 3.
6. Partner 2 und 3 verarbeiten die Anfrage und liefern die Daten an den Vermittler.
7. Der Vermittler führt die Umformung der Daten für den Partner 1 durch und schickt ihm die Antwort.

In weiterem Verlauf des Kapitels werden die wichtigsten Aspekte der Übersetzungsabläufe dargestellt:

### Die Initialisierung des Übersetzers

Die Initialisierung des Übersetzers besteht im Wesentlichen aus einer Erstellung der Maps für die einzelnen Attribute und Klassen der beiden Ontologien. Diese werden dann in die Datenabfragesprache RDQL (Kapitel 2.6) umgesetzt. Für die Modellierung der einzelnen Ontologien wird in der Arbeit die Sprache RDF(S) aus dem Kapitel 2.5.1 verwendet. Es wird zwischen zwei Arten von Einträgen unterschieden:

**Entry:** Ist ein Attributeintrag für die Fälle, in denen das Abfragekonzept in der Ausgangsdomäne auf eine Klasse verteilt ist.

**ComposeEntry:** Ist ein Attributeintrag für die Fälle, in denen ein Konzept in der Ausgangsdomäne auf mehreren Klassen verteilt ist.

Die Einträge werden in einem RDF(S)-Schema [20] gespeichert. In jedem Eintrag wird der Weg von der Wurzel eines Konzeptes zum gewünschten Attribut in einem eigenem Modell und der Weg von der Wurzel eines Konzeptes bis zum zugehörigen Attribut in einem fremden Modell gespeichert. Es werden auch die zughörigen Verteilungsregeln für Attribute gespeichert. Letztere werden für die Erstellung der Übersetzungen von SELECT- und WHERE-Klauseln der Sprache RDQL [24] verwendet. Man kann die Initialisierung des Übersetzers in folgende Schritte aufteilen:

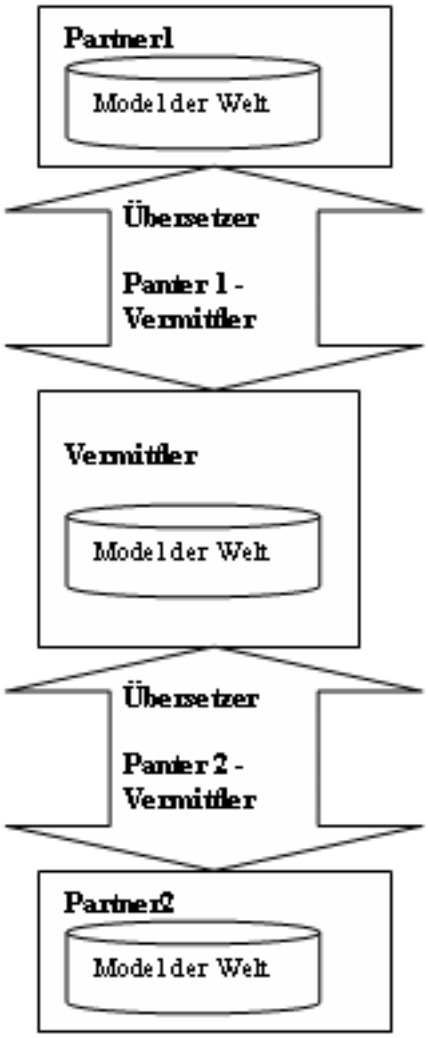


Abbildung 4.8.: Beispiel des Systemaufbaus

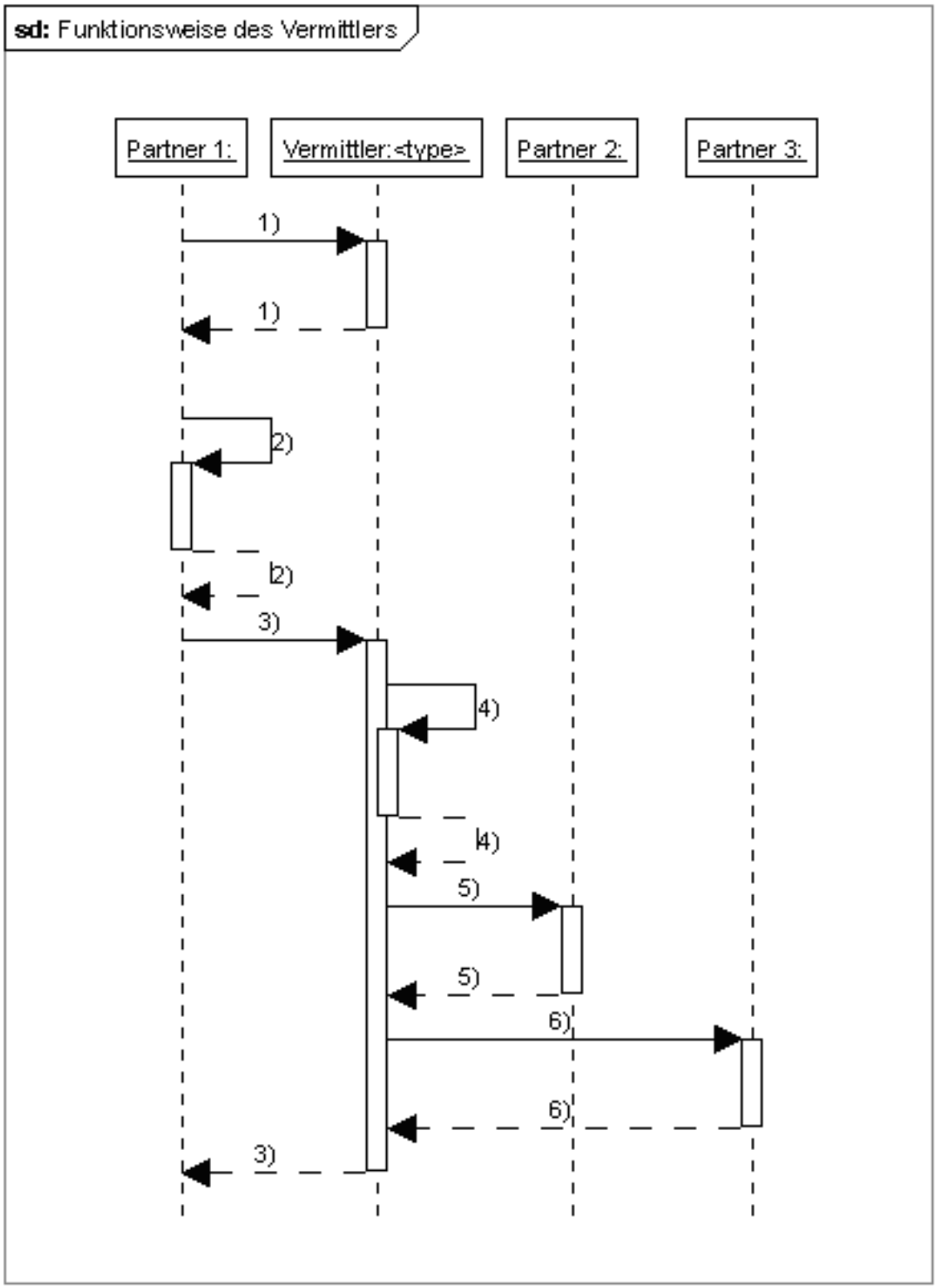


Abbildung 4.9.: Ablauf der Übersetzung zwischen 3 Partnern

1. *Die Definition der Austauschkonzepte:* In dieser Phase werden zwischen dem Vermittler und dem Partner die Konzepte bestimmt, welche dann später zum Austausch benutzt werden können. Dies ermöglicht die Kommunikation auf relativ einfache Weise zu spezifizieren. Hat man z.B. eine große Domäne, die sich mit vielen Themen beschäftigt, möchte man vielleicht jetzt nur die Konzepte der Autovermietung nach außen zur Verfügung stellen. Dazu müssen dann bei dem Übersetzer nur die Konzepte der Autovermietung angemeldet werden und die Restriktion wird dann automatisch so erzeugt, dass die anderen Konzepte vor der Anfrage von außen geschützt bleiben.
2. *Die Definition der Konzeptwurzel.* Die Konzeptwurzel ist ein Knoten in einem Ontologiebaum. Er stellt für alle Klassen, Prädikate und Attribute den Wurzelknoten in der Baumstruktur dar. Die Konzeptwurzeln werden für jedes Exportkonzept definiert und zu den Konzeptwurzeln des Vermittlers geordnet. Die Konzeptwurzel wird bei der Übersetzung in die SELECT- Klausel einer RDQL-Anfrage eingetragen.
3. *Die Bestimmung der Abfrageattribute von Konzepten.* Hier werden die Attribute bestimmt, die bei dem jeweiligen Konzept abgefragt werden können. Des Weiteren werden sie zueinander zugeordnet.
4. *Die Definitionen der Pfade von der Konzeptwurzel zu den Abfrageattributen.* Diese Pfade kommen in die WHERE-Klausel einer RDQL-Anfrage.
5. *Die Zuordnung der Pfade der Abfrageattribute zu den Pfaden der Abfrageattribute des Partners.* Parallel dazu wird die Verteilungsregel für die Attributwerte der Partnerontologie definiert und gespeichert.
6. Am Ende wird jeder Abfrage ein ID zugewiesen und in einem RDF-Schema, welches jeweils für beide Partner gedacht ist, gespeichert.

In den folgenden Beispielen werden für die unterschiedlichen Arten der Modell-Heterogenität aus dem Kapitel 3.1 die Initialisierungen des Übersetzers präsentiert.

***Syntaktischer Modellunterschied:***

*Vorbedingung:* Die Baumstruktur von beiden Ontologien ist gleich. Der Unterschied liegt in der Namensgebung von Klassen und Prädikaten. Wir gehen davon aus, dass beide Konzepte die gleiche semantische Bedeutung haben.

1. *Die Definition der Konzeptwurzel der beiden Konzepte.* Dies sind jeweils die Klassen A1 und die Klasse B1.
2. *Die Bestimmung der Abfrageattribute beider Konzepte und ihre Zuordnung.*

| | |
|---|---|
| Att1 | Att3 |
| Att2 | Att4 |

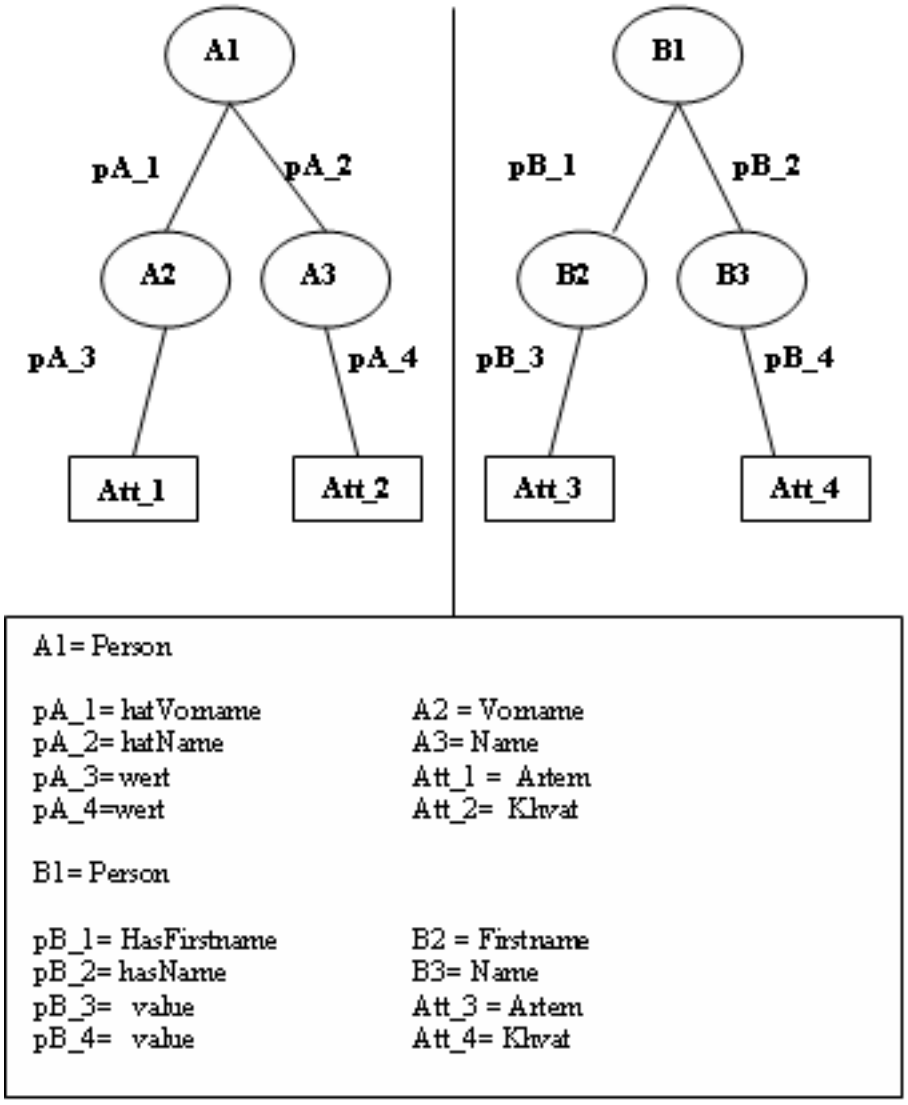


Abbildung 4.10.: Syntaktischer Modellunterschied

3. *Die Definitionen der Pfade von der Konzeptwurzel zu den Abfrageattributen.*

| | |
|---|---|
| Att1 | A1->pA1->A2->pA3 |
| Att2 | A1->pA3->A3->pA4 |
| Att3 | B1->pB1->B2->pB3 |
| Att4 | B1->pB3->B3->pB4 |

4. *Die Zuordnung der Pfade der Abfrageattribute zu den Pfaden der Abfrageattribute des Partners mit zugehörigen Verteilungsregeln.*

| | |
|---|---|
| A1->pA1->A2->pA3 | B1->pB1->B2->pB3 |
| A1->pA3->A3->pA4 | B1->pB3->B3->pB4 |

5. *Die Übersetzung der Abfrage in RDQL wird dann folgendenmaßen aussehen. (Pseudo- Code keine Berücksichtigung der RDQL-Syntax):*

| Domäne A: | Domäne B: |
|---|---|
| **SELECT** | **SELECT** |
| A1 | B1 |
| **WHERE** | **WHERE** |
| A1,pA1,A2,pA3,Att1 | B1,pB1,B2,pB3,Att3 |
| A1,pA3,A3,pA4,Att2 | B1,pB3,B3,pB4,Att4 |

***Attributtopologischer Modellunterschied:***

*Vorbedingung:*

- Die Baumstruktur beider Ontologien ist unterschiedlich.
- Die Namensgebung von Klassen und Prädikaten ist unterschiedlich.
- Der Strukturaufbau von Attributen ist unterschiedlich.
- Es wird davon ausgegangen, dass beide Konzepte die gleiche semantische Bedeutung haben.

1. *Definition der Konzeptwurzel beider Konzepte.* Dies sind jeweils die Klassen A1 und B1.
2. *Die Bestimmung der Abfrageattribute beider Konzepte und ihrer Zuordnung.*

| | |
|---|---|
| Att1 | Att3 |
| Att2 | Att3 |

3. *Die Definitionen der Pfade von der Konzeptwurzel zu den Abfrageattributen.*

| | |
|---|---|
| Att1 | A1->pA1->A2->pA3 |
| Att2 | A1->pA3->A3->pA4 |
| Att2 | B1->pB1->B2->pB2 |

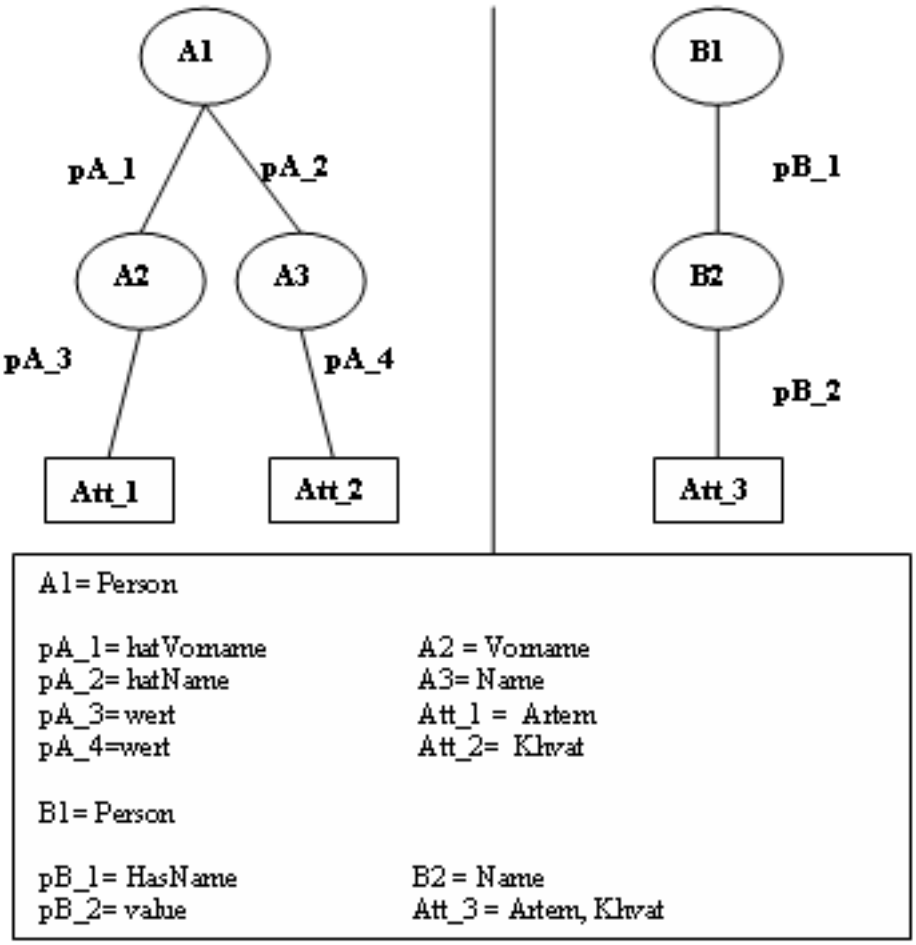


Abbildung 4.11.: Attributtopologischer Modellunterschied

4. *Die Zuordnung der Pfade der Abfrageattribute zu den Pfaden der Abfrageattribute des Partners mit zugehörigen Verteilungsregeln.* Wenn z.B. das Attribut 1 den Namen, das Attribut 2 den Vornamen und das Attribut 3 den Namen und Vornamen zusammen durch Leerzeichen getrennt speichert, entstehen:

| | |
|---|---|
| A1->pA1->A2->pA3<br>(Regel 1 zu 1, Trennzeichen ' ') | B1->pB1->B2->pB2<br>(Regel 1 zu 1, Trennzeichen ' ') |
| A1->pA3->A3->pA4<br>(Regel 1 zu 2, Trennzeichen ' ') | B1->pB1->B2->pB2<br>(Regel 2 zu 1, Trennzeichen ' ' ) |

5. *Die Übersetzung der Abfrage in RDQL wird dann folgendermaßen aussehen (Pseudo-Code keine Berücksichtigung der RDQL-Syntax):*

| Domäne A: | Domäne B: |
|---|---|
| **SELECT**<br>A1 | **SELECT**<br>B1 |
| **WHERE**<br>A1,pA1,A2,pA3,Att1<br>A1,pA3,A3,pA4,Att2 | **WHERE**<br>B1,pB1,B2,pB2,Att3 |

***Attribut-abstrakter topologischer Modelunterschied:***

*Vorbedingung:*

- Die Baumstruktur beider Ontologien ist unterschiedlich.
- Die Namensgebung von Klassen und Prädikaten ist unterschiedlich.
- Der Strukturaufbau von Attributen ist unterschiedlich.
- Es wird davon ausgegangen, dass beide Konzepte die gleiche semantische Bedeutung haben.

1. *Die Definition der Konzeptwurzel beider Konzepte.* Dies sind jeweils die Klassen A1 und B1.
2. *Die Bestimmung der Abfrageattribute beider Konzepte und ihrer Zuordnung.*

| | |
|---|---|
| Att1 | Att3 |
| Att2 | Att3, Att4 |

3. *Die Definitionen der Pfade von der Konzeptwurzel zu den Abfrageattributen.*

| | |
|---|---|
| Att1 | A1->pA1 |
| Att2 | A1->pA2 |
| Att3 | B1->pB1->B2->pB3 |
| Att4 | B1->pB1->B2->pB3 |

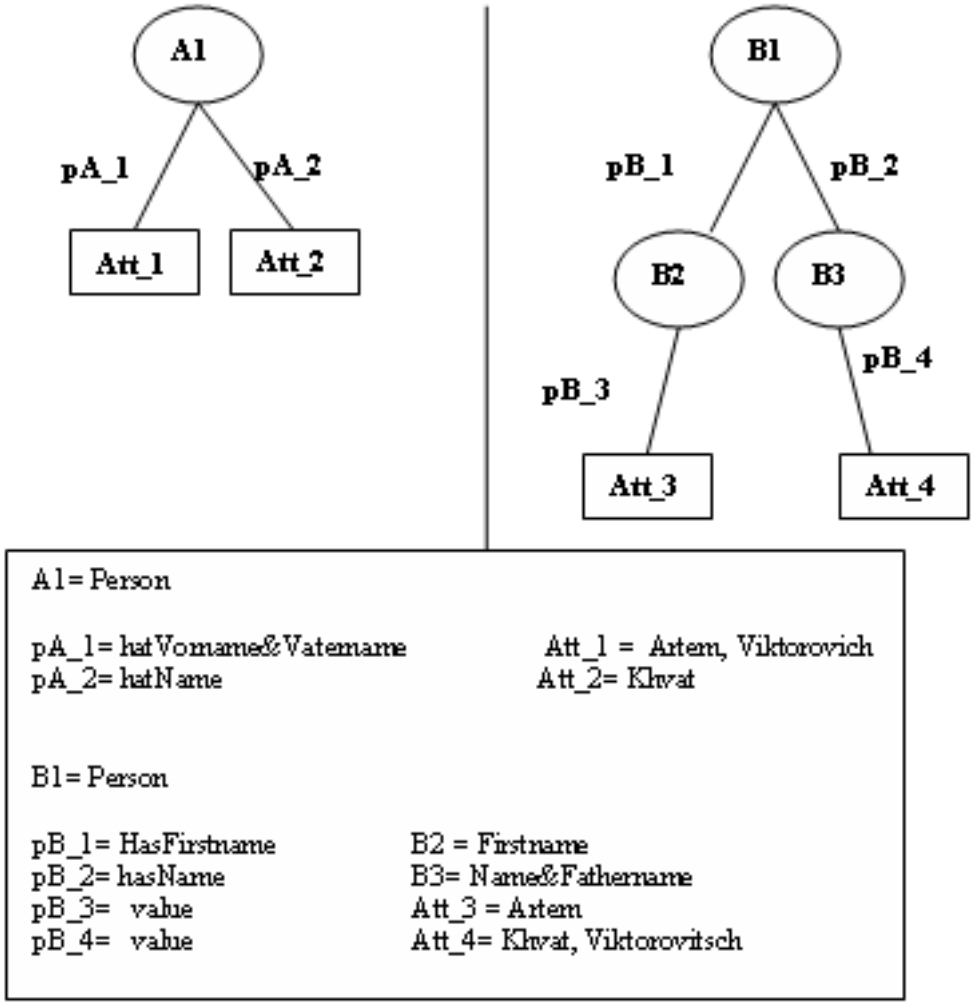


Abbildung 4.12.: Attributabstrakter topologischer Modellunterschied

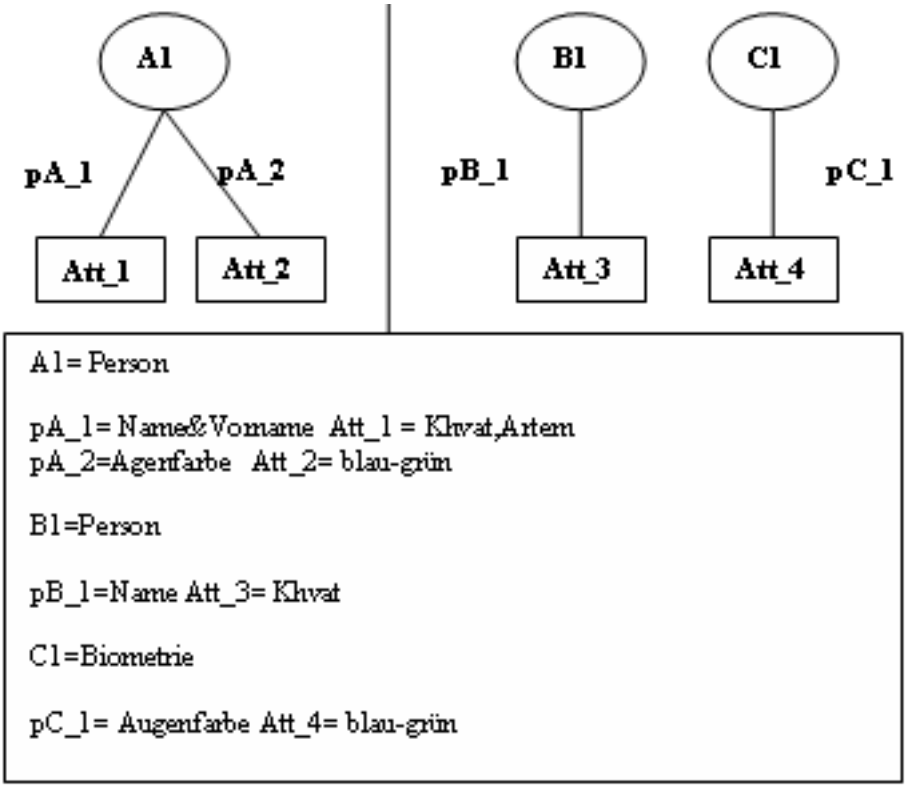


Abbildung 4.13.: Konzeptionelle Modellunterschiede

4. *Die Zuordnung der Pfade der Abfrageattribute zu den Pfaden der Abfrageattribute des Partners mit zugehörigen Regeln Verteilungsregeln.*

| | |
|---|---|
| A1->pA1 Regel (1 zu 1) Trennzeichen ' ' | B1->pB1->B2->pB3 Regel (1 zu 1) Trennzeichen ' |
| A1->pA2 Regel (1 zu 1) Trennzeichen ' ' | B1->pB3->B3->pB4 Regel (1 zu 1) Trennzeichen ' |
| A1->pA1 Regel (2 zu 2) Trennzeichen ' ' | B1->pB3->B3->pB4 Regel (2 zu 2) Trennzeichen ' |

5. *Die Übersetzung der Abfrage in RDQL wird dann folgendenmaßen aussehen (Pseudo-Code keine Berücksichtigung der RDQL-Syntax):*

| Domäne A: | Domäne B: |
|---|---|
| **SELECT** | **SELECT** |
| A1 | B1 |
| **WHERE** | **WHERE** |
| A1,pA1,Att1 | B1,pB1,B2,pB3,Att3 |
| A1,pA2,Att2 | B1,pB1,B2,pB3,Att4 |

***Konzeptionelle Modellunterschiede:***

*Vorbedingung:*

- Die Baumstruktur von beiden Ontologien ist unterschiedlich.
- Die Namensgebung von Klassen und Prädikaten ist unterschiedlich.
- Der Strukturaufbau von Attributen ist unterschiedlich.
- Es wird davon ausgegangen, dass ein Konzept in Domäne A auf zwei Konzepte der Domänen B und C verteilbar ist.

1. *Die Definition der Konzeptwurzel beider Konzepte.* Dies sind jeweils die Klassen A1,B1 und C1, welche als Kandidaten für eine Konzeptwurzel in Frage kommen. Angenommen wird dass die Domänen B und C sich an der Seite von einem Partner befinden und die Domäne A auf der Seite eines anderen Partners liegt. Die Zuordnung der Konzeptwurzel sieht in diesem Fall wie folgt aus:

| | |
|---|---|
| Att1 | B1,C1 |

2. *Die Bestimmung der Abfrageattribute beider Konzepte und ihrer Zuordnung.*

| | |
|---|---|
| Att1 | Att3 |
| Att2 | Att4 |

3. Die Definitionen der Pfade von der Konzeptwurzel zu den Abfrageattributen.

| | |
|---|---|
| Att1 | A1->pA1 |
| Att2 | A1->pA2 |
| Att3 | B1->pB1 |
| Att4 | C1->pC2 |

4. *Die Zuordnung der Pfade der Abfrageattribute zu den Pfaden der Abfrageattribute des Partners mit zugehörigen Verteilungsregeln.*

| | |
|---|---|
| A1->pA1 (Regel 1 zu 1) Trennzeichen ',' | B1->pB1 (Regel 1 zu 1) Trennzeichen ',' |
| A1->pA2 (Regel 1 zu 1) Trennzeichen | C1->pC2 (Regel 1 zu 1) Trennzeichen ' ' |

5. *Die Übersetzung der Abfrage in RDQL wird dann folgendermaßen aussehen (Pseudo-Code, keine Berücksichtigung der RDQL-Syntax):*

| Domäne A: | Domäne B,C: |
|---|---|
| **SELECT** | **SELECT** |
| A1 | B1,C1 |
| **WHERE** | **WHERE** |
| A1,pA1,Att1 | B1,pB1,Att3 |
| A1,pA2,Att2 | C1,pC2,Att4 |

#### Funktionsweise des Übersetzers

Die eigentliche Realisierung des Übersetzers findet im Logikmodul statt. Das Grundprinzip ist sehr einfach gehalten. Es wird die RDQL-Abfrage nach Attributpfaden, welche in der Map eingetragen sind untersucht. Sobald ein Pfad erkannt wurde, wird dieser durch den entsprechenden Pfad aus der Map ersetzt. Dieser Vorgang wird solange wiederholt, bis alle Pfade erkannt und ersetzt wurden.

## 4.9. MOnTo 0.7

MOnTo ist die Abkürzung für Merge Ontology Tool. Zur Zeit liegt es in der Version 0.7 vor und befindet sich im Prototypstatus. Es ist ein Werkzeug, dass für die Arbeit mit Ontologien geeignet ist. Es bietet dem Benutzer folgende Möglichkeiten durch die Interaktion mit der grafischen Benutzerschnittstelle:

- Die Visualisierung der Darstellung von Ontologien
- Die Erstellung und Modifikation von Ontologien
- Das Einfügen, Löschen und Umbenennen der einzelnen Elemente in Ontologien
- Die Unterstützung des manuellen und semi-automatischen Merges zweier Ontologien
- Die Aufdeckung von einfachen topologisch-syntaktischen Ähnlichkeiten
- Das Abfragen der Ontologien durch die Sprache RDQL
- Die Einbettung von Attributmaps, welche für eine automatische Übersetzung zwischen zwei Ontologien erforderlich sind (Sprache RDQL)
- Unterstützung der Ontologiesprachen RDF,RDF(S) und OWL

Die Software wurde, mit Hilfe des Jena 2.0 API [1] für die Programmiersprache Java erstellt, die schon in Kapitel 2.6 präsentiert wurde. Da es sich lediglich um einen Prototypen handelt, sind manche Operationen nicht sehr ergonomisch gestaltet. Dieses Defizit wird in folgenden Versionen von MOnTo beseitigt werden. Im folgenden Abschnitt werden die Gesamtarchitektur und die Implementierung der wichtigsten Funktionalitäten genauer erläutert. In MOnTo Version 0.7 findet die Implementierung der folgenden Systemmodule statt:

| | |
|---|---|
| **View** | *Admin-GUI* |
| **Geschäftslogik** | *Controllermodul*, *Logikmodul* |
| **Persistenz** | *Persistenzmodul* |

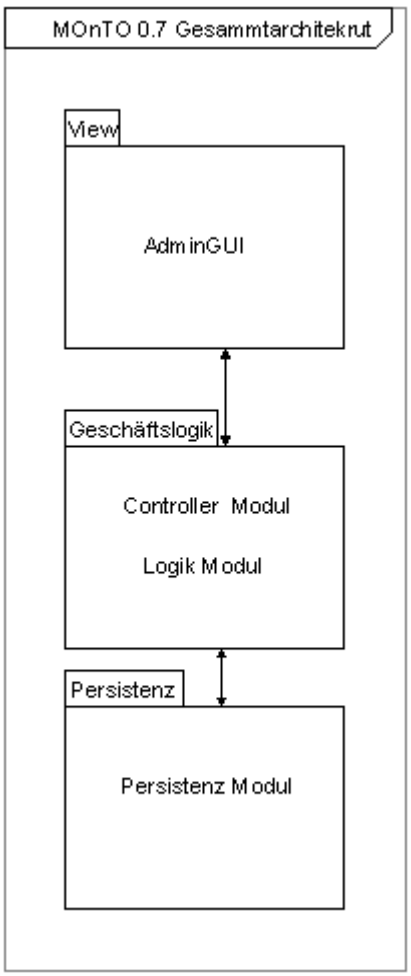


Abbildung 4.14.: MOnTo 0.7 Gesamtarchitektur

### 4.9.1. MOnTo Admin-GUI

Bei der Erstellung der grafischen Benutzerschnittstelle wurde der Standart GUI - Composer der Entwicklungsumgebung JBuilder 2005 www.borland.com/de/products/jbuilder/ verwendet. Letzterer ermöglicht die einfache und schnelle Erstellung der notwendigen Benutzeroberflächen. Die gesamte JBuilder 2005 Entwicklungsumgebung ist auf der Webseite von Borland für private und studentische Nutzung frei verfügbar. Es wird lediglich eine einmalige Registrierung mit einer gültigen E - Mail Adresse verlangt. Die meisten Komponenten der grafischen Benutzerschnittstelle kommen aus dem Java Swing Container und werden vom JDK 1.5 unterstützt. Die GUI ermöglicht die Interaktion mit dem Benutzer. Hier werden nur die Möglichkeiten der Ontologiebearbeitung beschrieben, die dem Benutzer während der Arbeit mit dieser grafischen Schnittstelle zur Verfügung stehen. Die Benutzerschnittstelle wird im Wesentlichen durch folgende vier Java Klasse realisiert, welche in dem Klassendiagramm in Abbildung 4.9 dargestellt sind.

***MOnToTabGUI*** - ist ein Container in welchem verschiedene Sichten untergebracht werden. Des weiteren ist sie eine Unterklasse der Swing Java Klasse JTabbedPane java.sun.com/j2se/1.4.2/docs/api/javax/swing/JTabbedPane.html.

***MOnToGUI*** - ist eine SchemaView-Sicht (Abbildung 4.10), mit der das Laden, das Anzeigen, die Modifikation, der Merge und der Vergleich zweier Ontologien möglich wird. An dieser Stelle wird nun auf die Operationen, die diese Sicht zur Verfügung stellt, eingegangen. Zunächst werden allerdings die einzelnen Hauptelemente der Benutzeroberfläche beschrieben. Man hat auf der linken und rechten Seite zwei Gruppen von Anzeigefenstern, welche jeweils aus drei Teilen in jeder Gruppe bestehen. Diese Anzeigefenster erlauben die Darstellung der geladenen Ontologien. Das obere Fenster zeigt den gesamten Baum der geladenen Ontologie. Das Fenster in der Mitte jeder Gruppe ist für die Darstellung der Ressourcen der Ontologie. Das Fenster darunter zeigt eine Liste von Prädikaten der geladenen Ontologie. Für die Darstellung der Baumstruktur der Ontologien wurde die Swing Java Klasse JTree java.sun.com/j2se/1.4.2/docs/api/javax/swing/JTree.html benutzt. Dies ermöglicht eine relativ übersichtliche Ansicht der geladenen Ontologie. Der Autor will besonders das Problem der Darstellung von großen Ontologien, was bei der Arbeit an dieser Version der Software zwar erkannt aber nicht gelöst wurde, betonen.

Beschreibung der einzelnen Operationen:

*Laden der Ontologien:*

1. Den Pfad zur gewünschten Ontologie im Eingabefeld [6] oder [21] eingeben
2. Das Wurzelelement der Ontologie in Eingabefeld [7] oder [23] eingeben
3. Die Taste [4] für die Home-Ontologie oder die Taste [22] für die Export-Ontologie

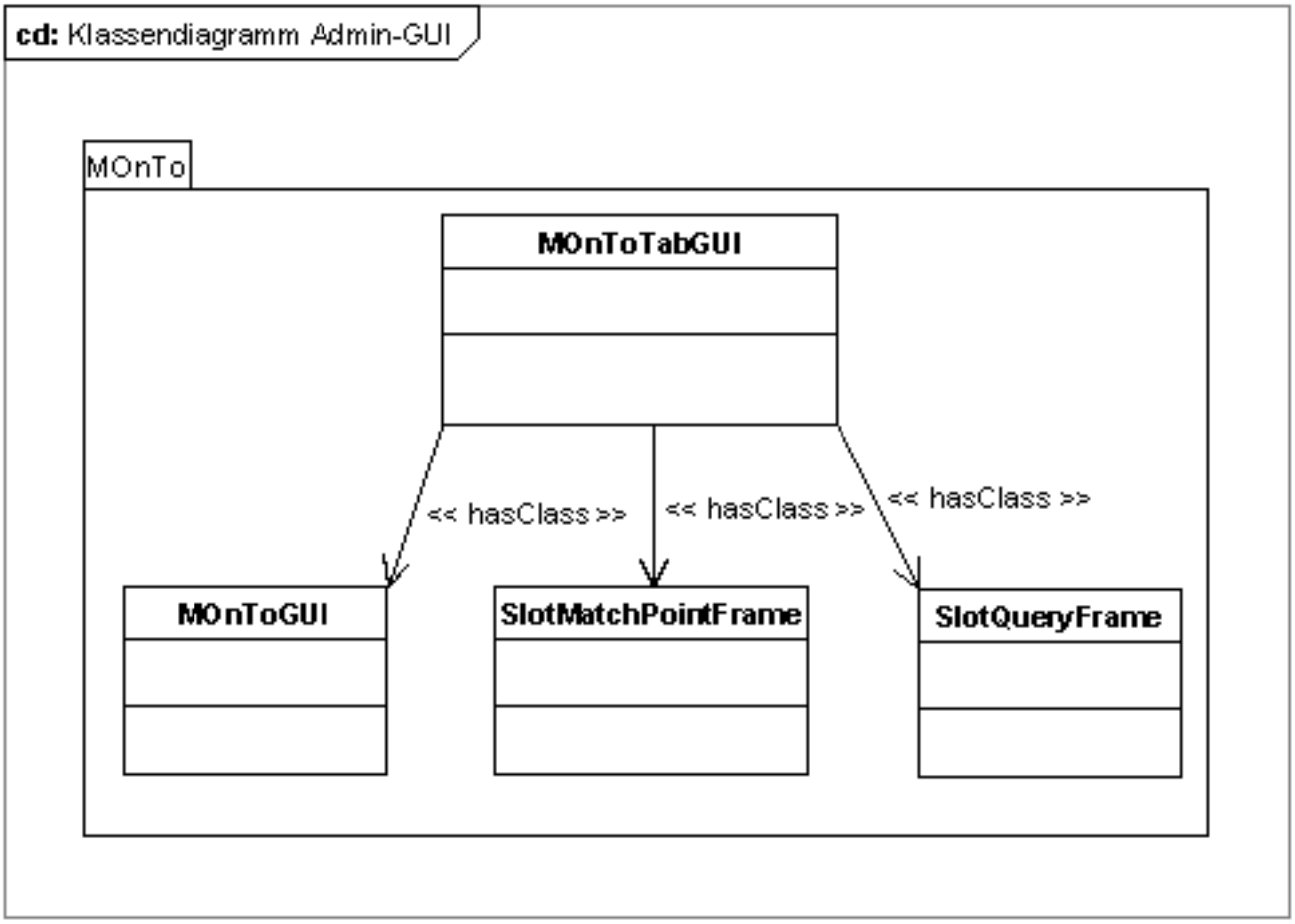


Abbildung 4.15.: MOnTo 0.7 Klassen Diagramm Admin-GUI

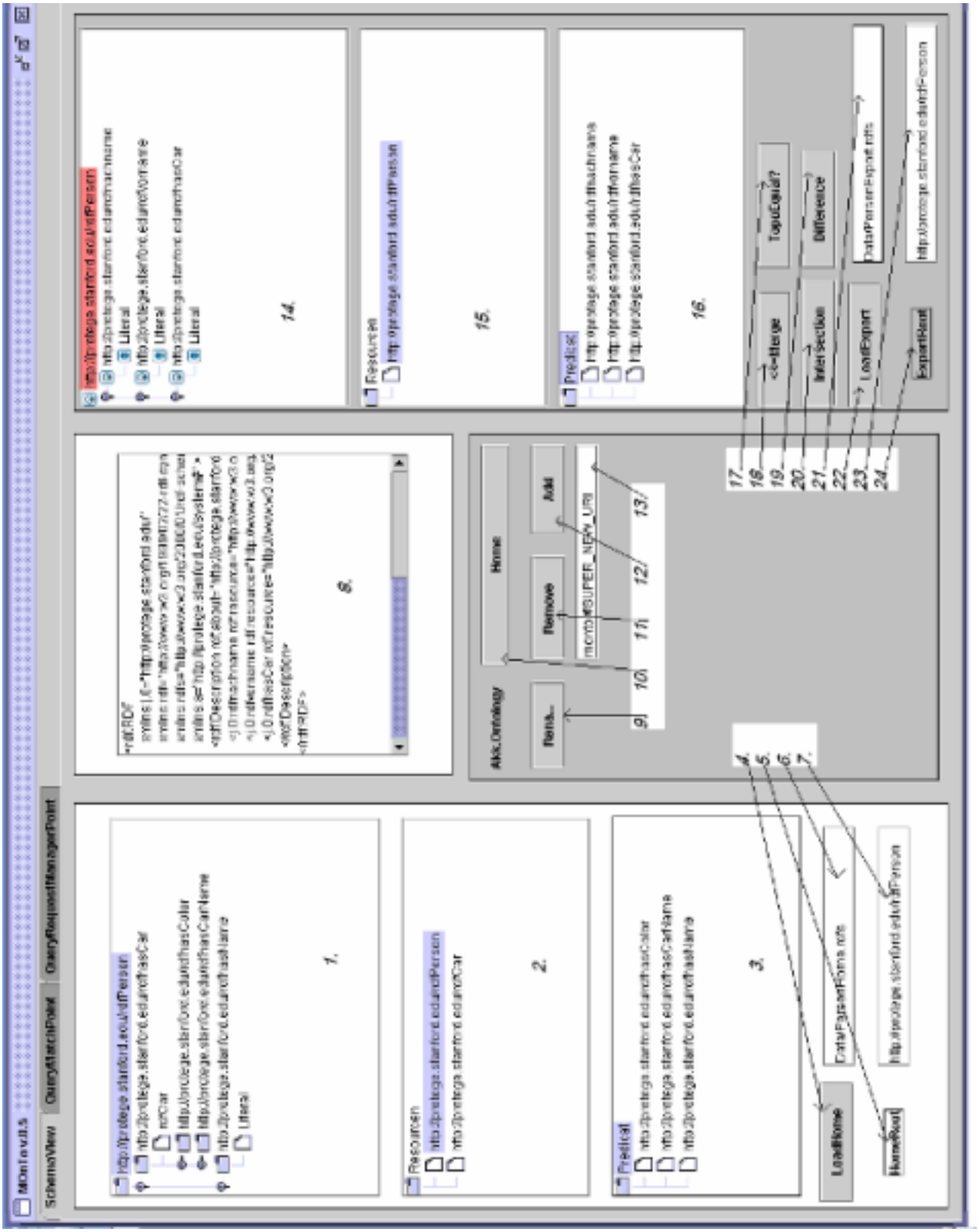

Abbildung 4.16.: MOnTo 0.7 SchemaView-Sicht

*Löschen:*

1. Die gewünschte Ressource oder das Rootelement des Teilbaums in Anzeigefenster [1] oder [14] mit der Maus markieren
2. Mit der Taste [10] die gewünschte Ontologie auswählen
3. Die Taste [11] betätigen

*Einfügen:*

1. Mit der Taste [10] die gewünschte Ontologie auswählen
2. Die gewünschte Ressource aus dem Ressourcen-Anzeigefenster [2] oder [15] auswählen
3. Das gewünschte Prädikat aus dem Prädikat-Anzeigefenster [3] oder [16] auswählen
4. Den gewünschten Teilbaum für das Einfügen in dem Ontologie-Anzeigefenster [1] oder [14] auswählen
5. Die Taste [12] betätigen

*Umbenennen:*

1. Mit der Taste [10] die gewünschte Ontologie auswählen
2. Die gewünschte Ressource aus dem Ressourcen-Anzeigefenster [2] oder [15] auswählen
3. Den neuen Namen in das Eingabefeld [13] eingeben
4. Die Taste [9] betätigen

*Merge:*

1. Mit der Taste [10] die gewünschte Ontologie auswählen
2. Das Wuzelelement der gewünschten Teilbäume im Ontologie-Anzeigefenster [1] und [14] auswählen
3. Die Taste [18] betätigen
4. Das Ergebnis des Merges wird im Ontologie-Anzeigefenster [8] ausgegeben

*Überprüfung auf Topologieübereinstimmung:* Bei diesem Prozess werden die beiden Ontologien auf ihre Topologie hin überprüft, wobei die Ressourcen vernachlässigt werden. Es werden nur Ontologiebäume und die Prädikate der jeweiligen Ontologiesprache verglichen. Dazu wird eine einfache Form der Graphenunifikation durchgeführt, die es erlaubt bei den gleichen Ausgangstopologien die Disjunktheit der Ontologien aufzudecken.

1. Das Wurzelelement der gewünschten Teilbäume im Ontologie-Anzeigefenster [1] und [14] auswählen
2. Die Taste [17] betätigen
3. Das Ergebnis wird in Anzeigefenster [8] ausgegeben

Um diese Operation zu verdeutlichen wird folgendes Beispiel gegeben. Man nimmt die Elemente Vierrad und Zweirad aus einer Ontologie für Automobile (Anhang A) und vergleicht diese. Wenn man die Topologien der Bäume dieser Elemente vergleicht, sind sie absolut identisch. Durch den Einsatz der Graphenunifikation kann allerdings die Disjunktheit beider Graphen festgestellt werden, ohne überhaupt etwas über Zweirad oder Vierrad wissen zu müssen. Dies ist möglich, weil die Disjunktheit eine bestimmte topologische Eigenschaft ist, welche durch Graphenunifikation feststellbar ist. Der Autor vermutet, dass die Weiterentwicklung dieses Ansatzes eine sehr große Rolle bei der automatischen Integration von neuen Informationsartefakten in bestehende Informationssysteme spielen kann.

*Difference:* Es werden zwei Ontologien verglichen. Als Ergebnis erhält man alle Elemente die in beiden Ontologien unterschiedlich sind.

1. Das Wurzelelement der gewünschten Teilbäume im Ontologie-Anzeigefenster [1] und [14] auswählen
2. Die Taste [19] betätigen
3. Das Ergebnis wird in Anzeigefenster [8] ausgegeben

*Intersection:* Es werden zwei Ontologien verglichen. Als Ergebnis erhält man alle Elemente die in beiden Ontologien gleich sind.

1. Das Wurzelelement der gewünschten Teilbäume im Ontologie-Anzeigefenster [1] und [14] auswählen
2. Die Taste [19] betätigen
3. Das Ergebnis wird im Anzeigefenster [8] ausgegeben

***SlotMatchPointFrame*** - ist eine QueryMatchPoint-Sicht, welche die Einstellung der Attributmap ermöglicht. Eine Attributmap wird angelegt, um die Übersetzung zwischen zwei Ontologien zu ermöglichen. In ihr werden die Zuordnungen zwischen den Elementen der beiden Ontologien gespeichert. Dafür wird ein RDF(S)-Baum aufgebaut und im Hauptverzeichnis als Datei mit dem Namen map.rdf abgespeichert. In der Abbildung 4.11 wird die QueryMatchPoint-Sicht der Benutzerschnittstelle dargestellt. Man kann zwei unterschiedliche Arten von Attributeinträgen, nämlich Entry und ComposeEntry, einlegen. Dadurch werden die topologischen Arten der semantischen Heterogenität aus Kapitell 3.1 abgedeckt.

**Entry** - sind Attributeinträge für den Fall, wenn das Abfrage-Konzept in der Ausgangsdomäne auf eine Klasse verteilt ist.

**ComposeEntry** - sind Attributeinträge für den Fall, wenn ein Konzept in der Ausgangsdomäne auf mehreren Klassen verteilt ist.

An dieser Stelle werden die einzelnen Hauptelemente der Benutzeroberfläche beschrieben. Man hat auf der linken und rechten Seite zwei Gruppen von Anzeigefenstern, die jeweils aus drei Teilen in jeder Gruppe bestehen. Diese Anzeigefenster [1] und [8] erlauben die Darstellung der geladenen Ontologien. Das obere Fenster zeigt den gesamten Baum der geladenen Ontologie. Die Fenster in der Mitte jeder Gruppe [2] und [9] sind für die Darstellung der Klassen der Ontologie. Die Fenster darunter [3] und [10] zeigen eine Liste von Attributen der in Fenster [2] und [9] ausgewählten Klassen. Für die Darstellung der Baumstruktur der Ontologien wurde die Swing Java Klasse JTree verwendet. Das Fenster [4] zeigt die aktuelle Liste schon erstellter Entrys und das Fenster [5] zeigt die jeweilige Verteilung dieser Entrys auf die Zielontologie. Das Fenster [6] zeigt die aktuelle Liste schon erstellter ComposeEntries und das Fenster [7] zeigt die jeweilige Verteilung dieser ComposeEntries in der Ausgangsontologie. In den Eingebefeldern [11] und [15] werden die Regeln für jedes Attribut eingegeben. Es ist möglich mehrere Regel für ein Attribut einzugeben. Die einzelnen Regeln werden durch Kommata getrennt. Die Regeln bestimmen die Zuordnung der Attributtokens. Links von den Trennzeichen steht der Index des Tokens in Ausgangsattribut. Rechts der Trennzeichen steht der Index des Tokens im Zielattribut. Für das bessere Verständnis kann an dieser Stelle folgendes Beispiel dienen. Betrachtet werden zwei Ontologien wobei sich in einer Ontologie die Klasse Name mit den Attributen Familienname und Vorname befindet. In der anderen Ontologie befindet sich die Klasse Name mit den Attributen VollständigerName, bei der Vorname und Name zusammengeschrieben werden. Ein Regel für das Attribut Familienname, bei der Erstellung von einem Attributmapeintrag, ist dann 0-0 bzw. 0-1 für das Attribut Vorname. Umgekehrt ist es für die Regel des Attributmapeintrag von VollständigeName. Diese sieht dann wie folgt aus: 0-0, 1-0. Man muss beachten, dass die Indizierung von Attributtokens bei 0 beginnt. Die Tasten [18] und [19] ermöglichen das Speichern und Laden der Attributmap.

Beschreibung der einzelnen Operationen:

*Erstellung von Attributeinträgen:*

1. Die gewünschten Klassen aus Anzeigefenster [2] und [9] auswählen
2. Die Regel in Eingabefeld [11] eintragen
3. Das Ausgangsattribut im Anzeigefenster [3] und das Zielattribut im Anzeigefenster [10] auswählen
4. Die Taste [12] drücken

*Erstellung von Attributeinträgen:*

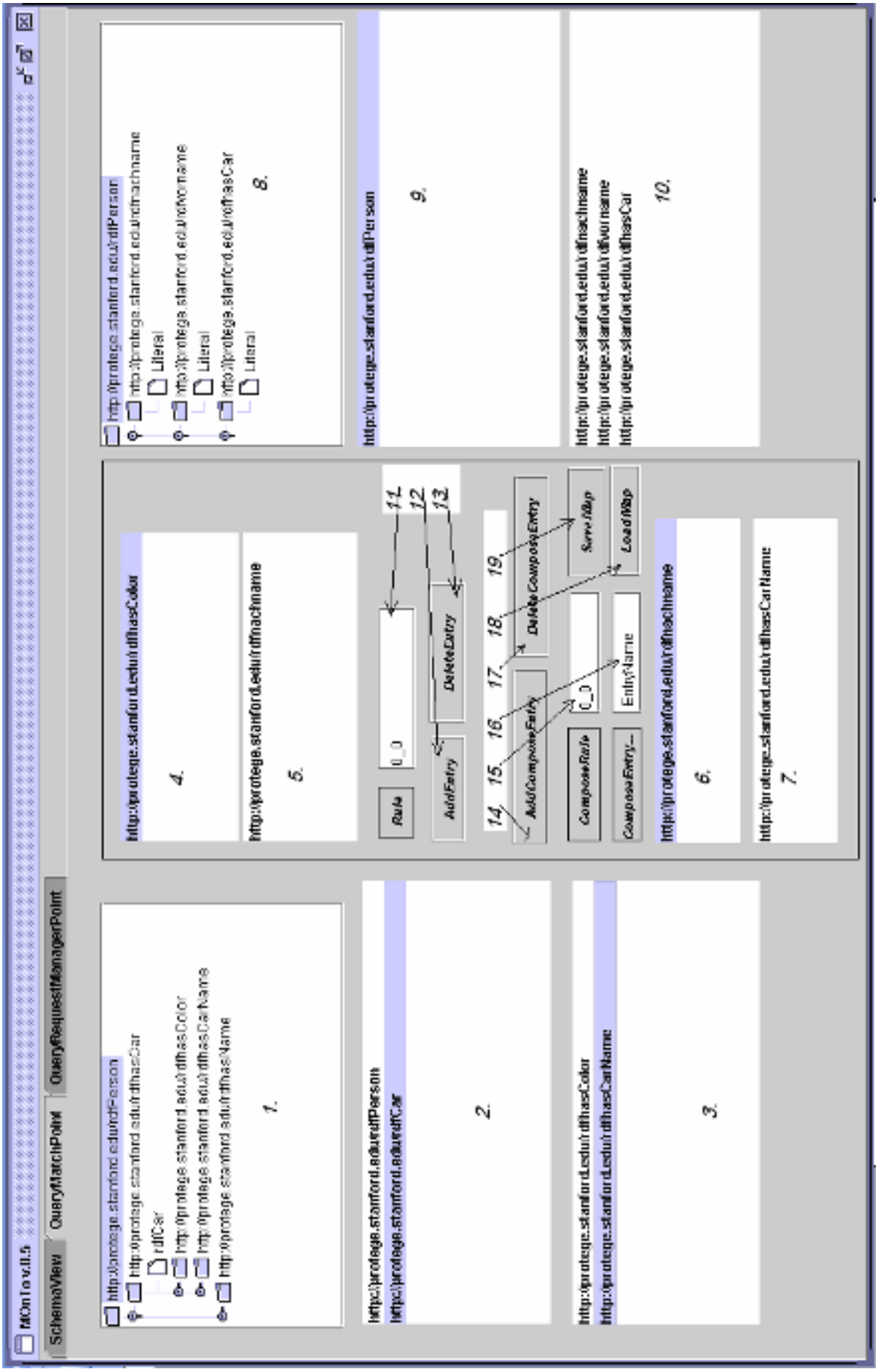


Abbildung 4.17.: MOnTo 0.7 QueryMatchPoint-Sicht

1. Die gewünschte Klassen aus Anzeigefenster [2] und [9] auswählen
2. Die Regel in Eingabefeld [15] eintragen
3. Name des Eintrags CompositEntry in Eingabefeld [16] eintragen
4. Taste [14] drücken

*Löschen von Einträgen:*

1. Im Anzeigefenster [4] den gewünschten Eintrag finden
2. Taste [13] drücken

*Löschen von ComposeEntry:*

1. Im Anzeigefenster [6] den gewünschten Eintrag finden
2. Taste [17] drücken

***SlotQueryFrame***- ist eine QueryRequestManagerPoint-Sicht, welche das Übersetzten von Abfragen zwischen zwei Ontologien ermöglicht. Man kann aus bestehenden Einträgen der Attributmap komplexe Abfragen erzeugen, welche dann automatisch in eine RDQL-Abfrage der Ziel-Ontologie übersetzt und abgefragt wird. Das Ergebnis besteht aus einem Container, welcher die zutreffenden Klassen enthält. Durch diese Eigenschaft wird die Fähigkeit der RDQL-Sprache zur reinen Datenabfrage erweitert. Man bekommt als Ergebnis kein einfache Zeichenkette mehr. Stattdessen werden RDF(S)-Sätze in einem Bag-Container zurückgeliefert (vgl. Kapitel 2.3.1). Im folgenden werden die einzelnen Elemente dieser Sicht der grafischen Benutzerschnittstelle vorgestellt. Die Beschreibungen beziehen sich dabei auf die Abbildung 4.12. In dem Anzeigefenster [1] wird die aktuelle Liste der erstellten Attributmapeinträge dargestellt. Es ist möglich einzelne Elemente auszuwählen. Dabei wird einerseits automatisch die dazugehörige Regelliste im Anzeigefenster [7] und andererseits die Auswahl der möglichen Operationen im Anzeigefenster [8] dargestellt. Im Eingabefeld [6] kann man die gesuchten Werte der Attribute und Klassen eingeben. Durch das Betätigen der Taste [9] kann der neue Abfragezweig in die aktuelle RDQL-Query [24] eingefügt werden. Im Anzeigefenster [2] wird die jeweils aktuelle RDQL-Query [24] dargestellt. Im Anzeigefenster [3] werden die URI der gefundenen Klassen zusammen mit den Namen der Abfragevariablen aufgelistet. Das Anzeigefenster [4] ermöglicht eine Einsicht in den Strukturbaum der gefundenen Klassen. Im Anzeigefenster [5] werden die Abfragedaten dargestellt. So sichergestellt werden, dass alle zutreffenden Datensätze richtig gefunden wurden.

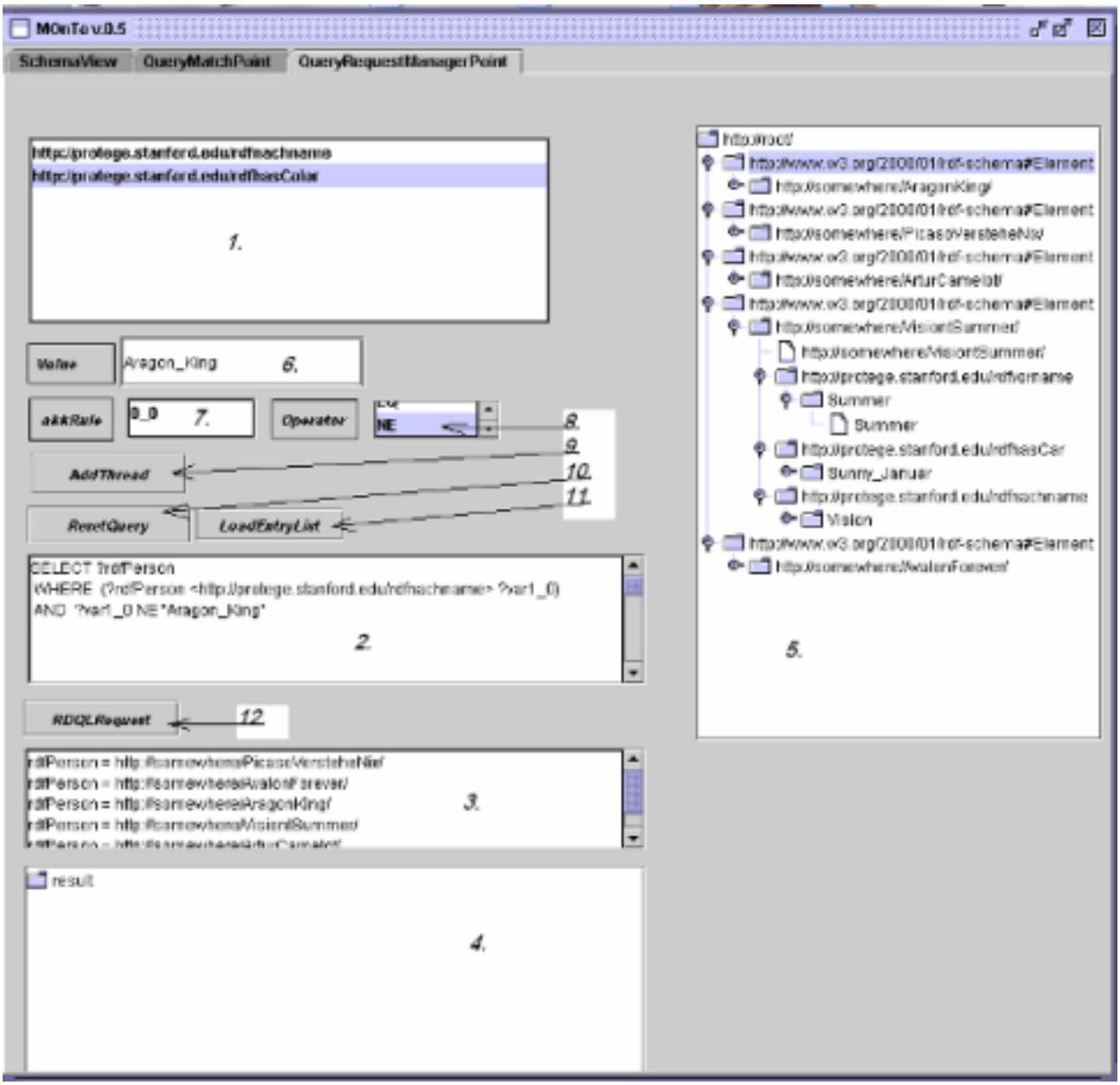


Abbildung 4.18.: MOnTo 0.7 QueryRequestManagerPoint-Sicht

### 4.9.2. MOnTo Controllermodul

Dieses Modul implementiert die Ablaufsteuerung der einzelnen Operationen sowie die Zusammenarbeit zwischen den GUI-, Logik- und Persistenzmodulen. Es ist in der Klasse MontoControl realisiert. Alle modulübergreifenden Operationen werden durch diese Klasse verwaltet.

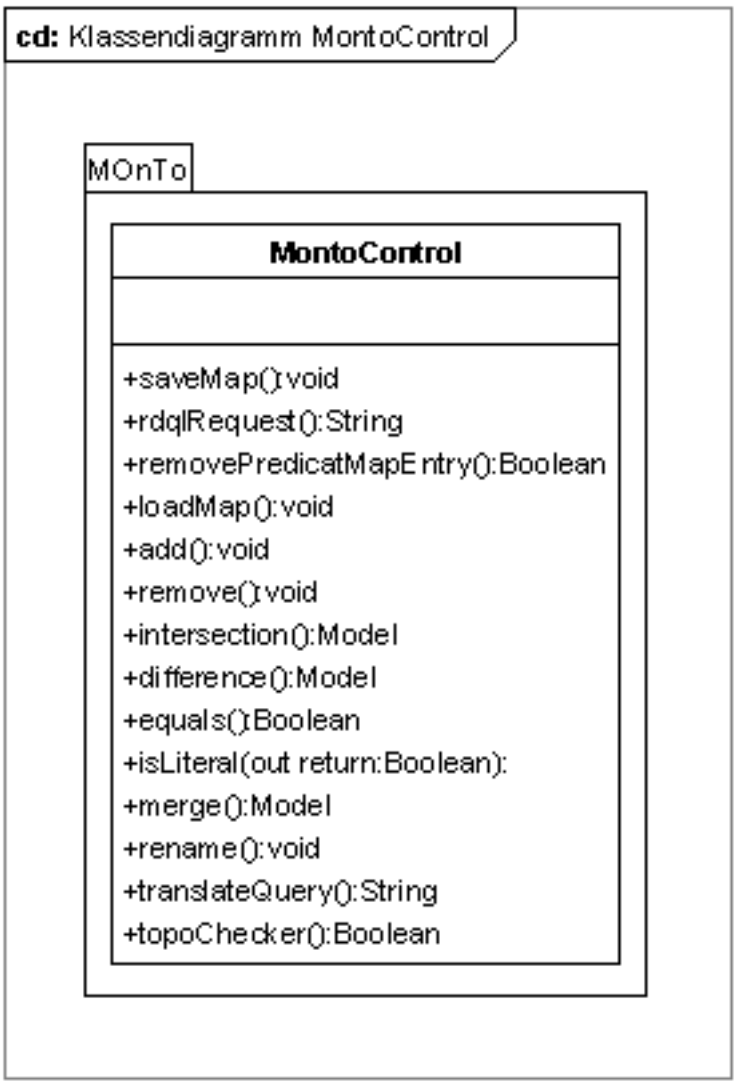


Abbildung 4.19.: MOnTo 0.7 Klassendiagramm MontoControl

### 4.9.3. MOnTo Logikmodul

Das Modul implementiert die Operationen auf Ontologien und Algorithmen wie den Vergleich und den Merge mehrerer Ontologien. Hier werden auch Operationen realisiert, die für das Bearbeiten von Ontologien notwendig sind. Z.B. das Löschen, Einfügen und Umbenennen von einzelnen Slots einer Ontologie. Dabei werden die Ontologie Sprachen RDF [16] und RDF(S) [20] unterstützt. Für die Datenabfrage wird die Sprache RDQL eingesetzt. Die Javaklasse MOnToLogik implementiert die obigen Aspekte.

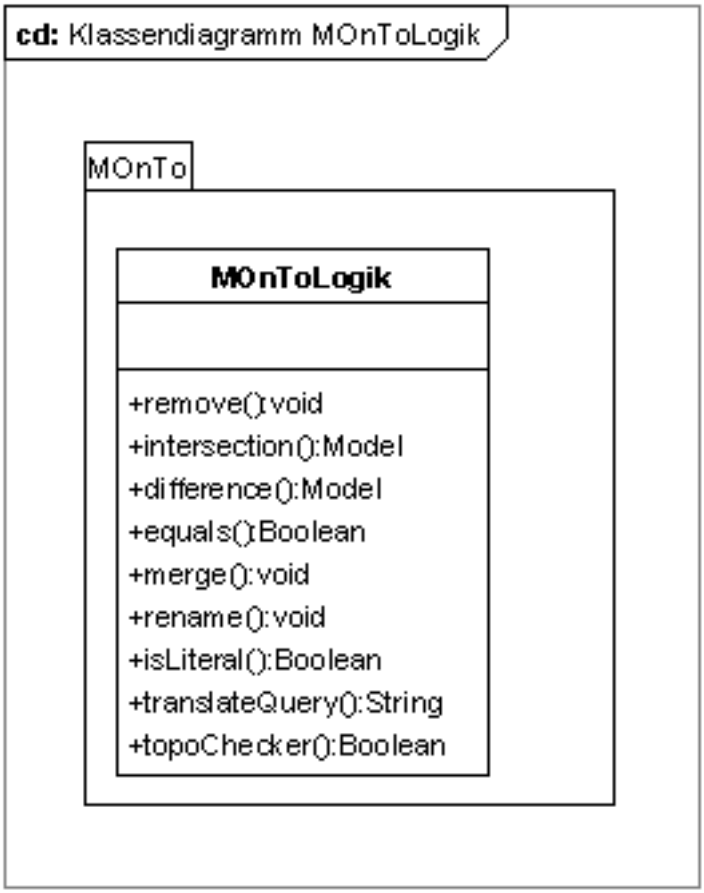


Abbildung 4.20.: MOnTo 0.7 Klassendiagramm MOnToLogik

### 4.9.4. MOnTo Persistenzmodul

Das Modul ist zuständig für die Speicherung und das Laden von Ontologien, das Verwalten der erstellten Merge-Maps, das Bearbeiten von Merge-Maps wie z.B. Löschoperationen und das Einfügen von neuen Regeln. Hier werden die meisten Operationen, die in der GUI (SlotMatchPointFrame) zur Verfügung stehen, implementiert. Realisiert wird das Modul durch die Klasse ModelMap. Alle Daten werden in Dateien auf der Festplatte gespeichert, in dieser Version von MOnTo ist noch keine Anbindung an eine Datenbank vorgesehen.

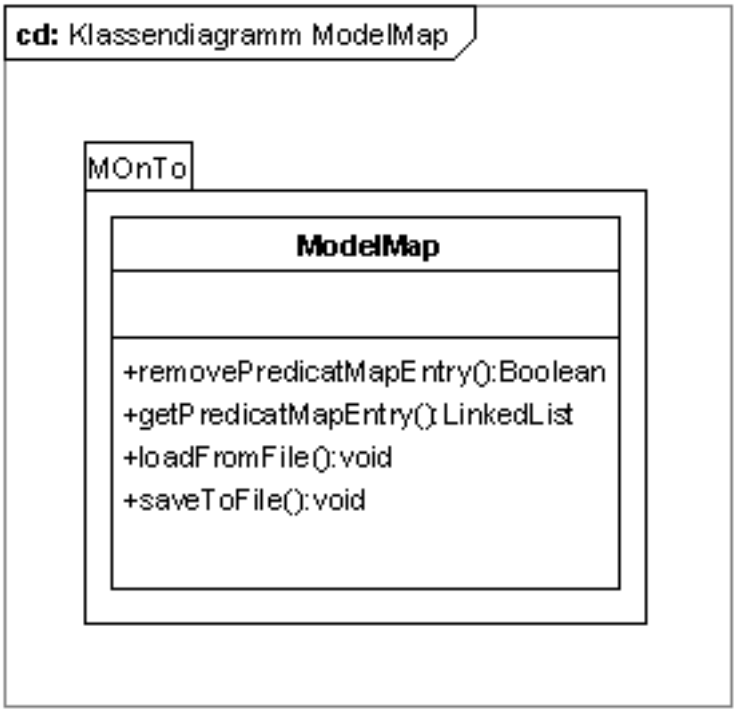


Abbildung 4.21.: MOnTo 0.7 Klassendiagramm ModelMap

### 4.9.5. Sequenzdiagramme

Für die Verdeutlichung der Funktionsweise des Programms sollen hier die einzelnen Systemabläufe mittels Sequenzdiagrammen darstellt werden. Jedes Sequenzdiagramm enthält zusätzlich textuelle Erläuterungen, welche die Verständlichkeit fördern sollen. Aufgrund des Umfangs der Software kann leider nur auf eine Auswahl der Funktionen eingegangen werden. Es werden folgende Abläufe beschrieben:

1. Laden und Anzeigen von Ontologien

2. Bearbeiten einer Ontologie (das Löschen, Umbenennen und Einfügen einzelner Slots)
3. Vergleich zweier Ontologien (Difference, Intersection)
4. Merge zweier Ontologien

*Laden und Anzeigen von Ontologien*

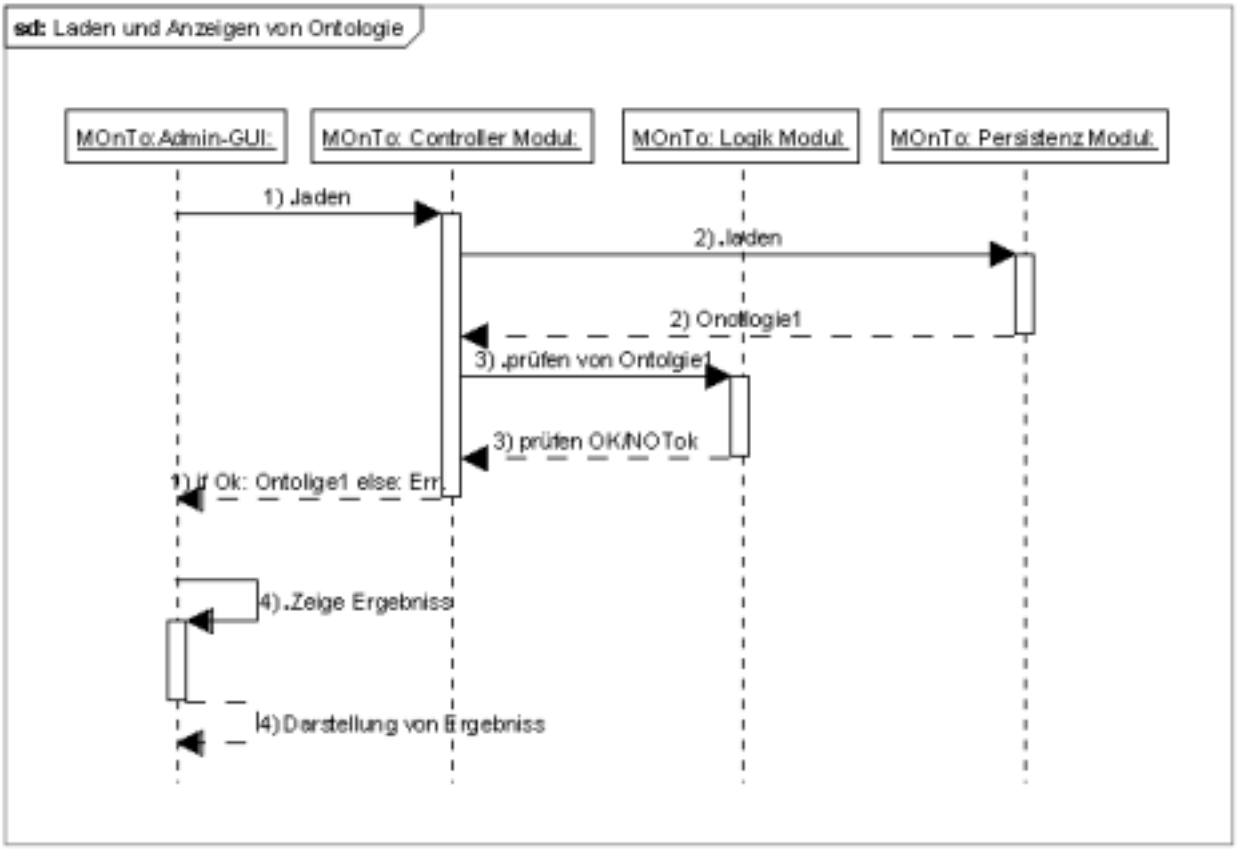


Abbildung 4.22.: MOnTo 0.7 Laden und Anzeigen einer Ontologie

1. Admin-GUI fordert das Laden einer bestimmten Ontologie an und sendet diese Anforderung an den Controller
2. Controller leitet Forderung an das Persistenzmodul weiter
3. Persistenz führt das Laden der gewünschten Ontologie durch. Wenn diese nicht verfügbar ist, so wird eine leere Ontologie zurückgeliefert
4. Controller schickt das Ergebnis zur Überprüfung an das Logik
5. Logik entscheidet ob der Ladevorgang erfolgreich war und meldet dies dem Controller

6. Controller leitet das Ergebnis an Admin-GUI weiter
7. Admin-GUI visualisiert das Ergebnis

*Bearbeitung einer Ontologie (das Löschen, Umbenennen und Einfügen einzelner Slots)*

*Vorbedingung:* Es wird davon ausgegangen, dass die beide Ontologien erfolgreich geladen wurden (siehe Sequenzdiagramm (Laden von Ontologien)) und sich im Speicher befinden.

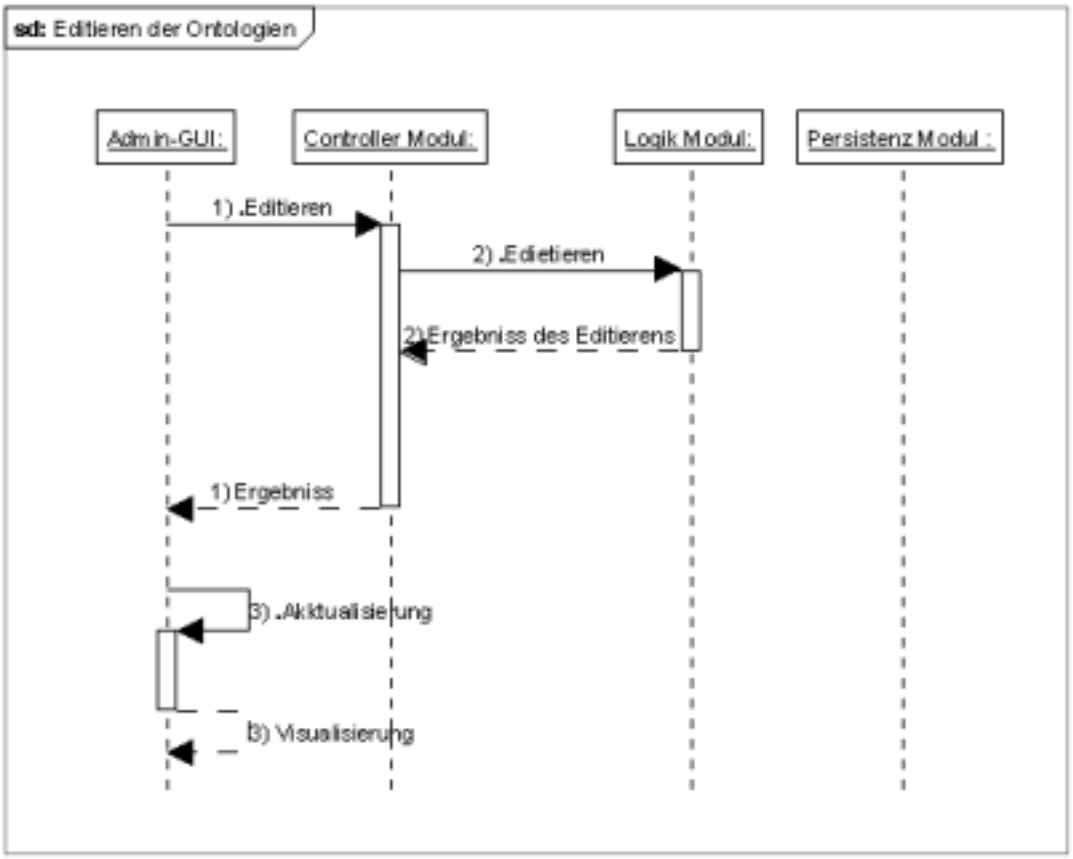


Abbildung 4.23.: MOnTo 0.7 Editieren der Ontologien

1. Admin-GUI schickt eine Anforderung zur Bearbeitung der aktuellen Ontologie an den Controller
2. Controller leitet die Anforderung zusammen mit den notwendigen Parametern an das Logikmodul weiter
3. Logikmodul schickt das Ergebnis zum Controller
4. Controller leitet das Ergebnis an die Admin-GUI weiter

5. Admin-GUI visualisiert die bearbeitete Ontologie

*Vergleich von zwei Ontologien (Difference, Intersection)*

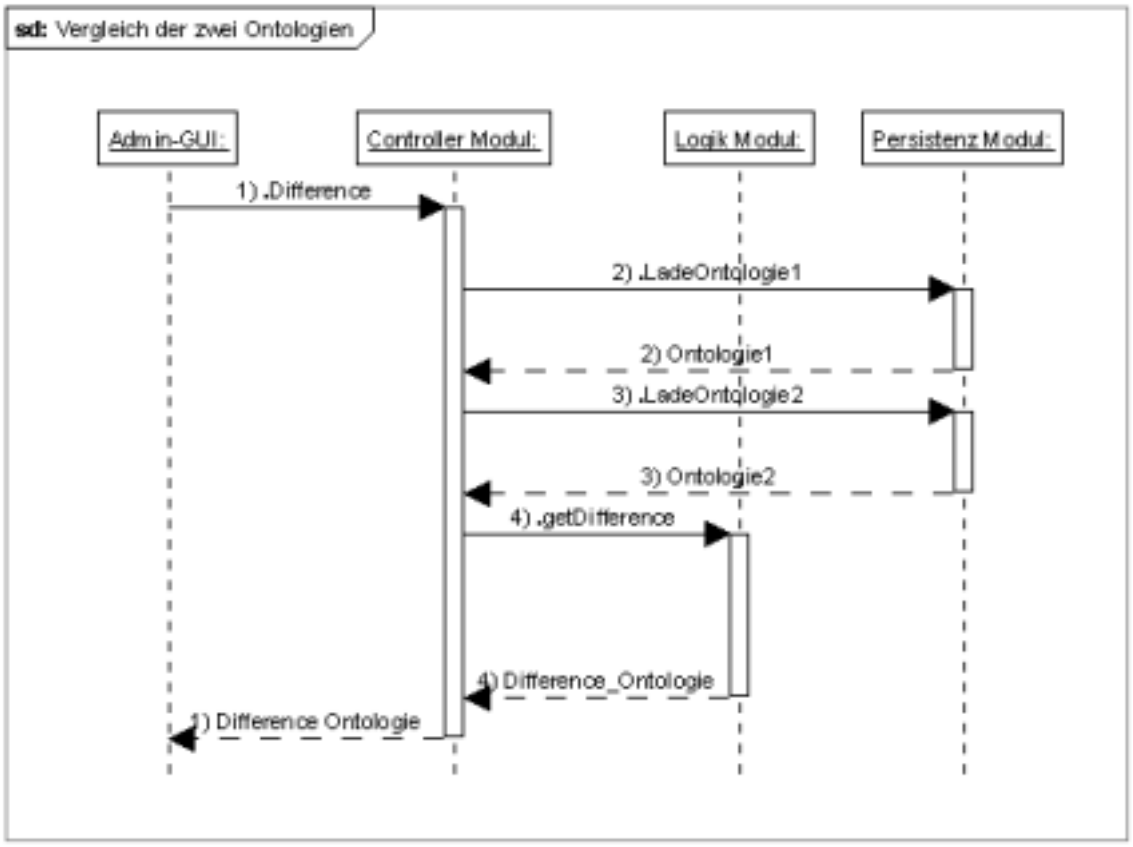


Abbildung 4.24.: MOnTo 0.7 Vergleich der zwei Ontologien

1. Admin-GUI schickt die Anforderungen zusammen mit den erforderlichen Parametern zum Controller
2. Controller steuert das Laden der zwei Ontologien, die verglichen werden sollen
3. Controller fordert das Logikmodul auf, den Vergleich durchzuführen
4. Logikmodul führt den Vergleich durch und schickt das Ergebnis zum Controller zurück
5. Controller leitet das Ergebnis des Vergleichs an die Admin-GUI
6. Admin-GUI visualisiert das Ergebnis

*Merge von zwei Ontologien*

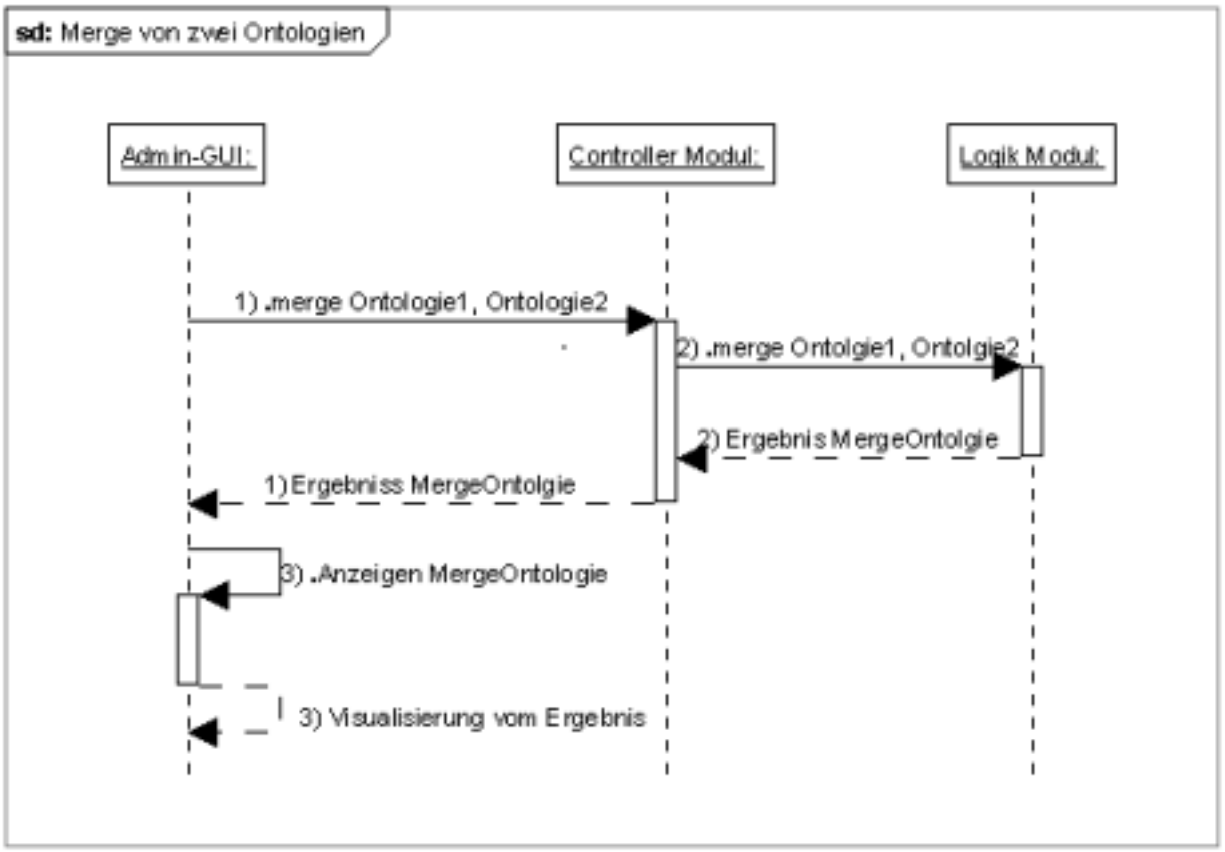


Abbildung 4.25.: MOnTo 0.7 Merge von zwei Ontologien.png

1. Admin-GUI schickt die Anforderung zusammen mit den erforderlichen Parametern zum Controller
2. Controller leitet die Anforderung weiter an Logik
3. Logik führt den Merge durch und schickt das Ergebnis an den Controller zurück
4. Controller gibt das Ergebnis der Operation an die Admin-GUI zurück
5. Admin-GUI visualisiert das Ergebnis

*Vorbedingung:* Es wird davon ausgegangen, dass die beiden Ontologien erfolgreich geladen wurden (siehe Sequenzdiagramm (Laden von Ontologien)) und sich im Speicher befinden.

### 4.9.6. Systemanforderungen

Die technischen Anforderungen an das System:

1. Windows XP (Sp1/2)
2. JDK 1.5
3. Intel P 4 (3.2 MHz)
4. 512 MB RAM

# 5. Zusammenfassung

Das Hauptziel dieser Arbeit war die Untersuchung von Möglichkeiten für die Beseitigung der Heterogenität zwischen semantisch annotierten Modellen. Es wurde sich mit der Frage nach einem System beschäftigt, welches unterschiedliche Modelle der Welt für alle Beteiligten verständlich machen kann. Ein sehr wichtiger Aspekt dabei war, dass die schon bestehenden Informationsbestände nicht in das neue gemeinsame Modell transformiert werden müssen und trotzdem nutzbar bleiben. Ein weiteres Ziel war die Automatisierung dieser Lösung und der Versuch die Rolle des Menschen bei diesem Integrationsprozess zu minimieren. Die Modelle selbst sollten nicht nur über die Struktur der Daten Aussagen liefern, sondern auch die semantische Beschreibung der Daten zur Verfügung stellen können. Der Autor ist der Meinung, dass die semantische Beschreibung der Daten eine wichtige Rolle in der Informationswelt der Zukunft spielen wird. Man kann schon heute zahlreiche Technologien und Anwendungen finden, welche die Vorteile der semantischen Beschreibung voll ausnutzen und die Möglichkeiten des modernen Internets für die Nutzer erweitern. Ein Beispiel ist die Technologie des Semantik Web und Web 2.0 http://twozero.uni-koeln.de/content/e14/index_ger.html. In dieser Arbeit wurden Ontologien für die Modellierung und die Beschreibung verwendet. Während dieser Arbeit wurde ein Entwurf für die Architektur eines semantischen Transparentssystems entwickelt. Es soll die Aufgaben der Integration von neuen Mitgliedern in semantisch annotierte Systeme ermöglichen. Es wurden Anwendungsfälle und die dazugehörigen Workflows definiert und beschrieben sowie der Entwurf des Systems fertiggestellt. Des weiteren wurde untersucht, in wie weit sich der Integrationsprozess automatisieren lässt. Zur Veranschaulichung des in dieser Arbeit vorgestellten Ansatzes wurde ein Prototyp mit dem Namen MOnTo in Version 0.7 entwickelt. Bei diesem Prototypen handelt es sich um eine Software, deren Aufgabe es ist, in erster Linie die Arbeit mit Ontologien zu ermöglichen. Die Hauptfunktionalitäten versetzen den Benutzer dieser Software in die Lage zwei Ontologien auf Ähnlichkeit zu untersuchen, Ontologien zu verändern sowie einen Merge zwischen mehreren Ontologien durchzuführen. Des weiteren realisiert der Prototyp eine Transparenzplattform, für unterschiedliche Modelle, welche sich in ihrer Struktur aber nicht in ihrer Semantik unterscheiden. Die genaue Beschreibung dieser Software findet sich in Kapitel 4 (Design und Realisierung). Es folgt eine Auflistung der mit MOnTo 0.7 zu lösenden Aufgaben.

1. Das Laden und die strukturelle Darstellung von Ontologien

2. Verändern von Ontologien mit Hilfe der grafischen Benutzerschnittstelle von MOnTo 0.7. Dies sind u.a. die Änderung von Namen sowie das Löschen und Einfügen von neuen Elementen in die geladene Ontologie.
3. Die Bestimmung der strukturellen Ähnlichkeiten zwischen Ontologien
4. Bestimmung der semantischen Ähnlichkeiten zwischen Ontologien
5. Abfrage von Instanzen der geladenen Ontologien durch die Abfragesprache RDQL
6. Erstellung von transparenten Abfragen für zwei unterschiedlich modellierte Ontologien, die mit Hilfe der Ontologiesprache RDF(S) erstellt wurden.
7. Erstellung von Abfrageergebnissen in vollständiger RDF(S)-Form

Mit dieser Arbeit schlägt der Autor ein weiteres Konzept vor, wie sich eine Lösung für die Nutzung der so verschiedenen aber gleichzeitig so ähnlichen Beschreibungen von Artefakten der realen Welt darstellen kann. Der Autor hat ein System entwickelt, welches die Kommunikation zwischen den unterschiedlichen Weltvorstellungen und Weltbeschreibungen ermöglichen soll. Diese Arbeit hatte das Ziel zu zeigen, dass die heutzutage verfügbaren Technologien des Semantik Web endlich in der Lage sind der Vielfältigkeit der Ideen von Menschen der realen Welt und ihre ständige Veränderung wiederzuspiegeln und füreinander verständlich zu machen. Nach Meinung des Autors liegt die Schönheit in der Vielfalt, diese muss allerdings verstanden werden.

## 5.1. Ausblick

Der Autor ist der Meinung, dass in wenigen Jahren das Thema der Verarbeitung von Daten mit semantischer Annotierung immer aktueller werden wird. Dabei wird die Unterstützung für die Aufbereitung solcher Daten von großer Bedeutung sein. Besonders das Problem der ergonomischen Gestaltung der Werkzeuge für die Arbeit mit großen Ontologien wird eine wichtige Rolle spielen. Es ist schon heute möglich viele Probleme der technischen Umsetzung zu lösen, wie in dieser Arbeit gezeigt wurde. Die Hauptschwierigkeit liegt allerdings darin diese komplexen technischen Abläufe so zu gestalten, dass jeder Benutzer ohne spezielle theoretische Kenntnisse in der Lage sein wird, mit ihnen zu Arbeiten. Erst wenn diese Herausforderung gelöst wird ist nach Meinung des Autors ein nächster Schritt in Richtung der Vision des Semantik Webs möglich.

Z.B. liegt eine der Schwierigkeiten im Bereich der Verarbeitung von großen Ontologien. Allein eine übersichtliche Darstellung solcher Ontologien mit Hilfe einer grafischen Benutzerschnittstelle ist ein Thema, welches sich vom Umfang für eine solide wissenschaftliche Arbeit eignet. Es wird eine Lösung benötigt, die einerseits eine Unterstützung für die Entwickler und

andererseits eine Unterstützung für Benutzer sein kann. Jeder Benutzer sollte in der Lage sein, durch eine einfache Handhabung bestehende Dateninhalte semantisch zu annotieren und in verständlicher Form den anderen Kommunikationspartnern zur Verfügung zu stellen. Erste Einsätze findet man schon heute im Bereich des Bloggings, wobei Autoren ihre Einträge mit einfachen RDF-Tags versehen können.

Eine weitere Herausforderung ist eine komplette Automatisierung der Übersetzung zwischen verschiedenen semantischen Modellen in dem Web der Zukunft. An dieser Stelle kann die Einrichtung eines globalen Übersetzungssystems eine Lösung sein, genauso wie man heute im Web Wörterbücher für die natürliche Sprache nutzt, könnten Dienste für die Übersetzung zwischen diversen semantischen Modellen eingerichtet werden. Die Lösungen, die während dieser Arbeit entwickelt wurden, können dabei eine wesentliche Rolle spielen. Die semantischen Wörterbücher werden eine Rolle als Vermittler zwischen den Kommunikationspartnern übernehmen und Zugriff auf eine globale Wissensdatenbank haben.

Der Autor ist der Meinung, dass die Welt sobald diese Probleme gelöst wurden, nicht mehr das sein wird, was sie vorher war. Maschinen werden in der Lage sein auf Grund derselben Informationen, die den Menschen zur Verfügung stehen, Entscheidungen zu treffen. Dies macht sie zu würdigen Partnern und Helfern der Menschheit, allerdings auch zu ihren gefährlichsten Gegnern. Es kann sein, dass in wenigen Jahrzehnten die Szenarien der heutigen Sience Fiction Filme nicht mehr die Phantasien ihrer Autoren bleiben werden. Die Frage ob es eine glückliche Geschichte oder ein Horrorfilm wird bleibt offen und die Antwort darauf soll die Menschheit selbst wählen.

# Literaturverzeichnis


[1] HP-Lab Bristol. – URL http://jena.sourceforge.net/

[2] *Methadaten*. Universität Götingen. – URL http://www2.sub.uni-goettingen.de/intrometa.html

[3] *Protege 2000*. Stanford University. – URL http://protege.stanford.edu/doc/users_guide/index.html

[4] BERNERS-LEE, Tim: *W3*. URL http://www.w3.org/DesignIssues/Semantic.html, 1998

[5] GARSHOL, Lars M.: http://www.xml.com/, 2001

[6] GOMEZ-PEREZ, Asuncion: *Ontological Engineering*. Springer London, 2005. – ISBN 1-85233-551-3

[7] HAW-HAMBURG: URL http://www.informatik.haw-hamburg.de/wiki/inf-master/index.php/

[8] JM: http://www.lexitron.de/, 2001. – URL http://www.lexitron.de/main.php?detail=true&eintrag=1179

[9] NOY, Natasha: URL http://protege.stanford.edu/plugins/prompt/prompt.html

[10] RDF.ORG: URL http://www.openrdf.org/doc/sesame/users/ch06.html

[11] RDF.ORG: URL http://www.openrdf.org/

[12] SQL: URL http://www.torsten-horn.de/techdocs/sql.htm/

[13] UDDI: URL http://www.uddi.org/

[14] W3C: URL http://www.w3.org/TR/1998/REC-xml-19980210

[15] W3C: URL http://www.w3.org/TR/html4/intro/sgmltut.html

[16] W3C: URL http://www.w3.org/RDF/

[17] W3C: URL http://www.w3.org/TR/rdf-syntax-grammar/

[18] W3C: URL http://www.w3.org/TR/rdf-schema/

[19] W3C: URL http://www.w3.org/TR/rdf-schema/

[20] W3C: URL http://www.w3.org/TR/rdf-schema/

[21] W3C: URL http://www.w3.org/2004/OWL/

[22] W3C: URL http://www.w3.org/Submission/2001/12/

[23] W3C: URL http://www.w3.org/TR/owl-ref/#Sublanguage-def

[24] W3C: URL http://www.w3.org/Submission/RDQL/

[25] W3C: URL http://www.w3.org/TR/rdf-sparql-query/

[26] W3C: URL http://www.w3.org/

[27] W3C: URL http://www.w3.org/TR/wsdl

[28] W3C: URL http://www.w3.org/Submission/OWL-S/

[29] WIKIPEDIA: URL http://de.wikipedia.org/wiki/Uniform_Resource_Identifier

[30] WIKIPEDIA: URL http://de.wikipedia.org/wiki/RDF-Schema

[31] WIKIPEDIA: URL http://de.wikipedia.org/wiki/Business-To-Business

[32] WIKIPEDIA: URL http://de.wikipedia.org/wiki/Business-To-Consumer

[33] WIKIPEDIA: URL http://de.wikipedia.org/wiki/Unified_Modeling_Language

# A. Anhang

```
<?xml version="1.0"?> <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/MietautoClub.owl#"
 xml:base="http://www.owl-ontologies.com/MietautoClub.owl">


 <owl:Class rdf:ID="Autobus">
   <rdfs:subClassOf>
     <owl:Class rdf:ID="LKW"/>
   </rdfs:subClassOf>
   <owl:disjointWith>
     <owl:Class rdf:ID="Zweitonner"/>
   </owl:disjointWith>
   <owl:disjointWith>
     <owl:Class rdf:ID="Dreitonner"/>
   </owl:disjointWith>
 </owl:Class>
 <owl:Class rdf:ID="Van">
   <rdfs:subClassOf>
     <owl:Class rdf:ID="PKW"/>
   </rdfs:subClassOf>
   <owl:disjointWith>
     <owl:Class rdf:ID="Cabriolets"/>
   </owl:disjointWith>
   <owl:disjointWith>
     <owl:Class rdf:ID="UntereMittelklasse"/>
   </owl:disjointWith>
   <owl:disjointWith>
```

```
      <owl:Class rdf:ID="Mittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="ObereMittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Kleinwagen"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Minis"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Gelendewagen"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Oberklasse"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="Fahrzeugklasse">
    <owl:disjointWith>
      <owl:Class rdf:ID="Mietfahrzeug"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="Moped">
    <owl:disjointWith>
      <owl:Class rdf:ID="Motorrad"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Zweirad"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:ID="Roller"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#Gelendewagen">
    <owl:disjointWith>
      <owl:Class rdf:about="#ObereMittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Mittelklasse"/>
```

```
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Oberklasse"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PKW"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#Minis"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Van"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#UntereMittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Kleinwagen"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Cabriolets"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#Mittelklasse">
    <owl:disjointWith>
      <owl:Class rdf:about="#Cabriolets"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Oberklasse"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Van"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Kleinwagen"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PKW"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#ObereMittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Gelendewagen"/>
    <owl:disjointWith>
```

```
      <owl:Class rdf:about="#UntereMittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Minis"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#Zweitonner">
    <owl:disjointWith rdf:resource="#Autobus"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Dreitonner"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#LKW"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#LKW">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Vierrad"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#PKW"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#UntereMittelklasse">
    <owl:disjointWith>
      <owl:Class rdf:about="#Cabriolets"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Minis"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PKW"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#ObereMittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Van"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Kleinwagen"/>
    </owl:disjointWith>
```

```
    <owl:disjointWith rdf:resource="#Mittelklasse"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Oberklasse"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Gelendewagen"/>
  </owl:Class>
  <owl:Class rdf:about="#Mietfahrzeug">
    <owl:disjointWith rdf:resource="#Fahrzeugklasse"/>
  </owl:Class>
  <owl:Class rdf:about="#Dreitonner">
    <owl:disjointWith rdf:resource="#Autobus"/>
    <rdfs:subClassOf rdf:resource="#LKW"/>
    <owl:disjointWith rdf:resource="#Zweitonner"/>
  </owl:Class>
  <owl:Class rdf:about="#Motorrad">
    <owl:disjointWith rdf:resource="#Moped"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Roller"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Zweirad"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#PKW">
    <owl:disjointWith rdf:resource="#LKW"/>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Vierrad"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Kleinwagen">
    <rdfs:subClassOf rdf:resource="#PKW"/>
    <owl:disjointWith rdf:resource="#Van"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Oberklasse"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Mittelklasse"/>
    <owl:disjointWith rdf:resource="#UntereMittelklasse"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Minis"/>
    </owl:disjointWith>
```

```
    <owl:disjointWith rdf:resource="#Gelendewagen"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#ObereMittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Cabriolets"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#Oberklasse">
    <owl:disjointWith>
      <owl:Class rdf:about="#ObereMittelklasse"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Cabriolets"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Van"/>
    <owl:disjointWith rdf:resource="#Mittelklasse"/>
    <owl:disjointWith rdf:resource="#Kleinwagen"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Minis"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#UntereMittelklasse"/>
    <owl:disjointWith rdf:resource="#Gelendewagen"/>
    <rdfs:subClassOf rdf:resource="#PKW"/>
  </owl:Class>
  <owl:Class rdf:about="#Zweirad">
    <owl:disjointWith>
      <owl:Class rdf:about="#Vierrad"/>
    </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#Fahrzeugklasse"/>
  </owl:Class>
  <owl:Class rdf:about="#ObereMittelklasse">
    <owl:disjointWith rdf:resource="#Kleinwagen"/>
    <owl:disjointWith rdf:resource="#Mittelklasse"/>
    <owl:disjointWith rdf:resource="#UntereMittelklasse"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Minis"/>
    </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#PKW"/>
    <owl:disjointWith rdf:resource="#Gelendewagen"/>
```

```
    <owl:disjointWith>
      <owl:Class rdf:about="#Cabriolets"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Oberklasse"/>
    <owl:disjointWith rdf:resource="#Van"/>
  </owl:Class>
  <owl:Class rdf:about="#Cabriolets">
    <owl:disjointWith rdf:resource="#Kleinwagen"/>
    <owl:disjointWith rdf:resource="#Van"/>
    <owl:disjointWith rdf:resource="#Mittelklasse"/>
    <owl:disjointWith rdf:resource="#UntereMittelklasse"/>
    <rdfs:subClassOf rdf:resource="#PKW"/>
    <owl:disjointWith rdf:resource="#ObereMittelklasse"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Minis"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Gelendewagen"/>
    <owl:disjointWith rdf:resource="#Oberklasse"/>
  </owl:Class>
  <owl:Class rdf:about="#Minis">
    <owl:disjointWith rdf:resource="#Kleinwagen"/>
    <owl:disjointWith rdf:resource="#Cabriolets"/>
    <owl:disjointWith rdf:resource="#Van"/>
    <rdfs:subClassOf rdf:resource="#PKW"/>
    <owl:disjointWith rdf:resource="#ObereMittelklasse"/>
    <owl:disjointWith rdf:resource="#Gelendewagen"/>
    <owl:disjointWith rdf:resource="#Mittelklasse"/>
    <owl:disjointWith rdf:resource="#UntereMittelklasse"/>
    <owl:disjointWith rdf:resource="#Oberklasse"/>
  </owl:Class>
  <owl:Class rdf:about="#Roller">
    <owl:disjointWith rdf:resource="#Moped"/>
    <rdfs:subClassOf rdf:resource="#Zweirad"/>
    <owl:disjointWith rdf:resource="#Motorrad"/>
  </owl:Class>
  <owl:Class rdf:about="#Vierrad">
    <owl:disjointWith rdf:resource="#Zweirad"/>
    <rdfs:subClassOf rdf:resource="#Fahrzeugklasse"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="hasA"/>
```

```
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.1, Build 284)
http://protege.stanford.edu -->
```

# B. Glossar

**HTML** Die Hypertext Markup Language (HTML), oft auch kurz als Hypertext bezeichnet, ist eine textbasierte Auszeichnungssprache zur Darstellung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten.

**Instanzen** Instanzen repräsentieren Objekte in der Ontologie und stellen das zur Verfügung stehende Wissen dar. Diese werden anhand vorher definierter Konzepte erzeugt und auch als Individuals bezeichnet (z.B. München oder Deutschland).

**JDK** *Java Development Kit*, abgekürzt JDK oder im neueren Fall J2SDK, sind die Java-Entwicklungswerkzeuge.

**KI** Künstliche Intelligenz (kurz: KI; englisch: Artificial Intelligence, kurz: AI) ist ein Teilgebiet der Informatik mit interdisziplinärem Charakter. Ziel der KI-Forschung ist die Entwicklung von Maschinen mit intelligentem Verhalten.

**Konzept** Die Beschreibung gemeinsamer Eigenschaften wird als Konzept definiert. Konzepte werden auch als Klassen bezeichnet. Diese können in einer Klassenstruktur mit Über-und Unterklasse angeordnet werden (z.B. das Konzept Stadt oder Land).

**ODBC** (Open DataBase Connectivity - *Offene Datenbank-Verbindungsfähigkeit* ) ist eine standardisierte Datenbankschnittstelle, die SQL als Datenbanksprache verwendet. ODBC bietet also eine Programmierschnittstelle (API), die es einem Programmierer erlaubt, seine Anwendung relativ unabhängig vom verwendeten Datenbankmanagementsystem (DBMS) zu entwickeln, wenn dafür ein ODBC-Treiber existiert.

**OKBC** Die Open Knowledge Base Connectivity (OKBC) ist ein Protokoll zum Zugriff auf Wissensdatenbanken (Knowledge Base), die in Knowledge Representation Systems (KRS) gespeichert sind. OKBC ist also eine allgemeine Schnittstelle auf Ontologien, ähnlich wie ein ODBC-Treiber Zugriff auf verschiedene Datenbanken ermöglicht.

**Prolog** Prolog ( *Programming in Logic* , auch frankophon *Prologue* ) ist eine Programmiersprache, die maßgeblich von Alain Colmerauer, einem französischen Informatiker, Anfang der 1970er Jahre entwickelt wurde und zur Familie der deklarativen Programmierung zählt. Sie ist eine Vertreterin der logischen Programmiersprachen. Ursprüngliche Prologvarianten arbeiten vollständig auf Basis des Edinburgh Standards, bei aktuelleren Versionen wird dies aber nicht mehr konsequent durchgehalten.

**Relationale Datenbanken** Eine relationale Datenbank ist eine Datenbank, die auf dem relationalen Datenbankmodell basiert, das von Edgar F. Codd 1970 erstmals vorgeschlagen wurde; darin ist eine Relation ein im streng mathematischen Sinn wohldefinierter Begriff, der im Wesentlichen ein mathematisches Modell für eine Tabelle beschreibt.

**Swing** Bei Swing handelt es sich um ein API zum Programmieren von grafischen Benutzeroberflächen. Swing wurde von Sun Microsystems für die Programmiersprache Java entwickelt. Seit Java-Version 1.2 (1998) ist es Bestandteil der Java-Runtime.

**WWW** das über das Internet abrufbare weltweite Hypertext-System World Wide Web.

Für die Erstellung von Glossar wurden die Definitionen aus www.wikipedia.de verwendet.

# Abbildungsverzeichnis

# Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 11. Dezember 2006
Ort, Datum

Unterschrift