FAKULTÄT FÜR !NFORMATIK

Integration von Ontology Alignment Algorithmen in AIViz

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Manuel Gradwohl
Matrikelnummer 0425062

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuerin: Univ.Ass. Dipl.-Ing. Dr.techn. Monika Lanzenberger

Wien, 28.09.2009

_____ (Unterschrift Verfasser)

_____ (Unterschrift Betreuerin)

Bei der Erstellung dieses Dokumentes
wurde LaTeX 2_ε mit TeXnicCenter verwendet.

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, am 28.09.2009

Manuel Gradwohl

Danksagung

An dieser Stelle möchte ich mich bei meiner Betreuerin Dipl.-Ing. Dr.techn. Monika Lanzenberger für ihre Ratschläge und Bemühungen bedanken. Weiters gilt mein Dank meiner Familie, ohne deren entgegengebrachte Unterstützung ich mein Studium nicht hätte absolvieren können. Für jegliche Unterstützung beim Verfassen der Arbeit bedanke ich mich bei Verena Gradwohl, Barbara Mollay, Beate Hemmerlein, Thorsten Guggenberger, Thomas Kaufmann, Michael Pickelbauer und Sabine Eder. Bei Christoph Waystyn möchte ich mich für die LaTeX-Vorlage bedanken.

Kurzfassung

Für die Umsetzung der Visionen des Semantic Webs spielt das Kombinieren von unabhängigen Wissensbasen, die beim Semantic Web durch Ontologien umgesetzt werden, eine zentrale Rolle. Auf Grund der semantischen Heterogenität zwischen den Ontologien ähnlicher Domänen ist es allerdings erforderlich diese gegenseitig abzustimmen, sodass sie konsistent sowie kohährent zueinander sind. Dieser Vorgang wird als Ontology Alignment bezeichnet. Die Kernaufgabe dabei ist die Ähnlichkeiten bzw. Übereinstimmungen - so genannte Alignments - zwischen Ontologien zu ermitteln.

Es gibt bereits eine Menge an Ontology Alignment Algorithmen bzw. Systemen wie z.B. FOAM, RiMOM, Lily usw., welche Ähnlichkeiten basierend auf Domänenwissen, das durch die Ontologien beschrieben wird, (semi-)automatisch feststellen. Die Ergebnisse der Algorithmen sind leider nicht immer zufriedenstellend. Da die Größe von Ontologien ständig wächst, ist es für den bzw. die BenutzerIn sehr schwierig, die vom Algorithmus gefundenen Alignments zu verfizieren.
Hier schafft das Tool AlViz Abhilfe, indem es die Informationen der Ontologien mit den ermittelten Alignments durch Kombination und Verknüpfung mehrerer Visualisierungstechniken graphisch darstellt. Weiters ermöglicht das Tool Änderungen an Alignments vorzunehmen.

Die Arbeit präsentiert den Entwurf sowie die Umsetzung einer Erweiterung von AlViz für die Integration von Ontology Alignment Algorithmen. Dazu wird eine Architektur mit einheitlicher Schnittstelle definiert, die auf der Ontology Alignment API [Euzenat, 2006] basiert. Die eingebundenen Alignment Algorithmen können dadurch direkt aus AlViz gestartet werden. Weiters ermöglicht dies eine schnelle Visualisierung und auch Analyse der Alignment-Ergebnisse in AlViz.
Außerdem können nun auch unterschiedliche Alignment Algorithmen kombiniert werden. Dabei kann das Alignment-Ergebnis eines Algorithmus als Eingabe für den Start des nächsten Algorithmus herangezogen werden. Somit können die Stärken von bestimmten Alignment Algorithmen gezielt eingesetzt sowie kombiniert werden.

Abstract

In order to put visions of the Semantic Web into practice, it's important to combine indepentend knowledge bases. These knowledge bases are represented by ontologies. Due to semantic heterogenity between ontologies of similar domains, it's necessary to align them with each other for them to become consistent and coherent. This process is called ontology alignment, the main task of which it is to find similarities, so called alignments, between the individual ontologies.

There already are numerious ontology alignment algorithms existing for example FOAM, RiMOM, Lily and so on. These algorithms (semi-)automatically find out the similarities based on domain knowledge which is described by ontologies. The results of those algorithms are, however, not always satisfying. Because ontologies are continuously growing in size, it's very difficult for the user to varify the resulting alignments of the algorithm. To assist in analyzing and verifying the resulting alignments of the algorithm the tool AlViz was introduced. It does so by combining and coppling various visualization techniques which represent information taken from the ontologies. Furthermore the tool allows for changing alignments.

This paper presents a concept as well as the realization of an extension of AlViz for the integration of additional ontology alignment algorithms. For this purpose an architecture with a uniform interface which is based on the Ontology Alignment API [Euzenat, 2006] is defined. Thus, the integrated alignment algorithms can be started directly with AlViz. In addition this allows for a quick visualization and analysis of the alignment results in AlViz.
With this extension of AlViz it is also possible to combine different alignment algorithms. Here the alignment result of one algorithm can be used as input for starting the next algorithm. Therefore the strengths of specific alignment algorithms can be implemented and combined selectively.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieses Kapitel gibt einen Überblick über die vorliegende Arbeit. Dabei werden die Motivation sowie die Zielsetzung für diese Arbeit beschrieben und abschließend die einzelnen Kapitel zur Orientierung kurz vorgestellt.

1.1 Motivation

Der Begriff *Semantic Web* ist mittlerweile in aller Munde. Dabei steht die Anreicherung von Webinhalten mit maschinenlesbarer Semantik, was eine automatisierte Verarbeitung der Semantik von Information (z.B. durch Software Agenten) als auch automatisiertes Schließen ermöglicht, im Mittelpunkt. Grundlage dafür bilden dabei Ontologien, mit denen Domänenwissen abgebildet werden kann.
Ein weiterer Aspekt, der in den Visionen des Semantic Webs verankert ist, ist das Kombinieren bzw. Verknüpfen von Wissen aus verschiedenen Quellen. Auf Grund der Heterogenität des Webs, d.h. ähnliches Wissen ist auf verschiedene Weise beschrieben, müssen Wissenbasen bzw. Ontologien aufeinander abgestimmt werden, sodass sie konsistent und koherent zueinander sind. Dieser Vorgang wird als Ontology Alignment bezeichnet, wobei die Kernaufgabe das Ermitteln von Ähnlichkeiten bzw. Übereinstimmungen ist.

Da hier eine Ähnlichkeit zur Behandlung heterogener Daten im Datenbereich besteht und der Autor bereits während seines Studiums Erfahrungen gesammelt hat, wurde das Interesse zur Verfassung einer Arbeit in diese Richtung geweckt. Ein weiterer Ansporn zur Erstellung der vorliegenden Arbeit war die Erweiterung eines bestehenden Tools für visuelles Ontology Alignment (*AlViz*), welches unterschiedliche Visualisierungstechniken verwendet, um die gefundenen Alignments eines Ontology Alignment Algorithmus dem bzw. der BenutzerIn übersichtlich und verständlich zu präsentieren. Auch das oft

komplexe, aber durchaus interessante Gebiet der Visualisierung war Wegbegleiter beim Studium des Autors.

1.2 Zielsetzung

Hauptziel dieser Arbeit ist die Präsentation und Umsetzung einer Architektur, welche die Integration von Ontology Alignment Algorithmen in AlViz ermöglicht. Dabei soll eine einheitliche Schnittstelle definiert werden, die den Austausch von Ähnlichkeiten zwischen den zugrundeliegenden Ontologien, so genannten Alignments, zwischen AlViz und den unterschiedlichen Ontology Alignment Algorithmen unterstützt. Dadurch soll AlViz die Daten von jedem beliebigen Algorithmus interpretieren und für die interne Verarbeitung sowie Visualisierung aufbereiten können.
Daraus ergibt sich der Vorteil, dass z.B. ein Alignment Algorithmus mehrmals hintereinander gestartet werden kann, wobei bei jedem neuerlichen Start die Ähnlichkeiten des vorangegangenen Durchlaufs als Eingabe übergeben werden, mit dem Ziel, dass bisher noch nicht entdeckte Ähnlichkeiten gefunden werden können. In weiterer Folge ergibt sich auch der Vorteil, dass das Ermitteln von Ähnlichkeiten zwischen Ontologien mit unterschiedlichen Algorithmen erfolgen kann, sodass die Stärken einzelner Algorithmen gezielt eingesetzt werden können.

Als weiteres Ziel wird die Anpassung von AlViz gesehen, die die Beseitigung der engen Kopplung an den Ontology Alignment Algorithmus FOAM, der in der Ausgangsversion von AlViz basierend auf [Huber, 2009] vorhanden ist, vornimmt und die Einführung einer losen Kopplung zu den integrierten Alignment Algorithmen beinhaltet.

Bevor allerdings auf diese Architektur im praktischen Teil der vorliegenden Arbeit eingegangen wird, erfolgt eine umfangreiche Einführung zu den Bereichen Semantic Web und Ontology Alignment als auch eine Beschreibung von FOAM und AlViz. Dadurch soll ein besseres Verständnis der Thematik vermittelt, sowie die Zusammenhänge der einzelnen Teile klarer werden.

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich grob in einen theoretischen und einen praktischen Teil sowie in einen Abschlussteil. Zum theoretischen Teil werden die Kapitel 2, 3, 4, 5, 6 und 7

gezählt und zum praktischen Teil die Kapitel 8 und 9. Den Abschlussteil bildet Kapitel 10. Nachfolgend werden die Inhalte der Kapitel überblicksmäßig erläutert.

Kapitel 2 - Semantic Web
Dieses einführende Kapitel skizziert die Probleme bei der Verarbeitung der derzeitigen Webinhalte und zeigt darauf aufbauend die Ziele und Visionen des Semantic Webs sowie dessen Architektur.

Kapitel 3 - Resource Description Framework: RDF
In diesem Kapitel wird das *Resource Description Framework* (RDF), welches eines der Kernkomponenten des Semantic Webs darstellt und die formale Beschreibung von Ressourcen (z.B. Informationen und Objekte) ermöglicht, erläutert. Außerdem werden die Basiskonzepte bzw. Sprachelemente von RDF-Schema erklärt, mit welchem das Vokabular einer Domäne spezifiziert und in RDF verwendet werden kann.

Kapitel 4 - Ontology Web Language: OWL
Auf Grund der oft notwendigen Ausdrucksstärke, die allerdings mit Hilfe von RDF-Schema nicht umgesetzt werden kann, wurde vom World Wide Web Consortium die Ontology Web Language (OWL) definiert.
Zunächst werden im Kapitel die Anforderungen an eine Ontologiesprache und die Grenzen von RDF-Schema aufgezeigt. Danach erfolgt eine Erklärung der wichtigsten Sprachelemente von OWL, worauf abschließend ein Überblick über die Subsprachen von OWL sowie mögliche zukünftige Erweiterungen gegeben wird.

Kapitel 5 - Ontology Alignment
Dieses Kapitel zeigt zunächst die Ebenen der semantischen Heterogenität zwischen Ontologien auf. Daraufhin werden die Grundlagen sowie Anwendungsbereiche zu bzw. von Ontology Alignment erörtert. Nachdem der Alignment Prozess schrittweise erklärt wird, beschäftigt sich der letzte Abschnitt dieses Kapitels mit der Evaluierung von Ontology Alignment Algorithmen.

Kapitel 6 - Framework for Ontology Alignment and Mapping: FOAM
Für den Aufbau von Verständnis in Bezug auf die Arbeitsweise von Alignment Algorithmen werden in diesem Kapitel anhand von FOAM interne Verarbeitungsschritte erläutert.

Kapitel 7 - AlViz - Ein Tool für visuelles Ontology Alignment
In diesem Kapitel wird die Ausgangsversion des Tools AlViz, welches Alignments für den bzw. die BenutzerIn mit Hilfe von verschiedenen Visualisierungstechniken graphisch aufbereitet, im Hinblick auf die durchzuführenden Erweiterungen bzw. Anpassungen

beschrieben. Im letzten Abschnitt dieses Kapitels werden noch offene Aspekte von AlViz aufgeschlüsselt.

Kapitel 8 - Praktischer Teil: Analyse & Design
Im ersten Abschnitt von Kapitel 8 werden die genauen Anforderungen an die Architektur, sowie die Schnittstelle für die Integration von Ontology Alignment Algorithmen bzw. die Einführung einer losen Kopplung zwischen AlViz und Alignment Algorithmen, erörtert. Basierend darauf wird das software-technische Design im zweiten Abschnitt dieses Kapitels erläutert.

Kapitel 9 - Praktischer Teil: Implementierung & Evaluierung
Die Beschreibung der Implementierung zum software-technischen Design aus Kapitel 8 ist in diesem Kapitel zu finden. Dabei werden die Funktionalitäten der einzelnen Klassen kurz umrissen sowie die Erweiterungen der graphischen Benutzeroberfläche von AlViz dargestellt. Außerdem wird anhand von FOAM die Integration in AlViz gezeigt.
Abschließend werden in diesem Kapitel die umgesetzten Erweiterungen bzw. Anpassungen an AlViz informal evaluiert. Nachdem der Testaufbau der Evaluierung beschrieben wurde folgt die Darstellung der daraus erhaltenen Ergebnisse.

Kapitel 10 - Abschluss
Das letzte Kapitel beinhaltet eine Zusammenfassung der davor behandelten Kapitel. Weiters werden die Schlussfolgerungen aus der Arbeit formuliert und abschließend mögliche Erweiterungen im Zusammenhang mit neuen Funktionalitäten von AlViz skizziert.

Kapitel 2

Semantic Web

Dieses Kapitel soll eine Einführung in Semantic Web geben. Zunächst werden die Grundzüge des derzeitigen und zukünftigen World Wide Web kurz umrissen um daraufhin die Visionen bzw. Ziele von Semantic Web anzusprechen.
Der zweite und abschließende Teil dieses Kapitels befasst sich mit der Architektur von Semantic Web. Hier werden überblicksmäßig die zum Einsatz kommenden Technologien beschrieben. Auf einen Teil dieser Technologien wird dann in den darauf folgenden Kapiteln näher eingegangen.

2.1 Das World Wide Web - heute und morgen

Das WWW (World Wide Web) hat sowohl die Art und Weise wie Menschen miteinander kommunizieren als auch die Wirtschaft im Gesamten wesentlich verändert. Man spricht von einer Entwicklung hin zur *Knowledge Economy* oder allgemeiner *Knowledge Society*. [Antoniou und van Harmelen, 2008]

Die Informationen im WWW werden durch natürliche Sprache (Deutsch, Englisch, Französisch, ...), Bilder oder multimediale Elemente dargestellt. Für den Menschen sind diese Darstellungsarten völlig ausreichend, um den Inhalt vollständig zu verstehen. Menschen können Tatsachen auch von nur teilweise verfügbaren Informationen ableiten, geistige Assoziationen herstellen, Informationen kombinieren und auch sensorische Informationen (z.B. Bilder, Sound) verarbeiten.
Allerdings reicht diese Informationsdarstellung für Computer nicht aus, um den Inhalt als auch die Verknüpfungen zwischen verschiedenen Webseiten zu verstehen und interpretieren zu können. Computer haben dabei Probleme mit unvollständigen Daten, mit

dem Kombinieren von Daten und dem Verarbeiten aller Arten von sensorischer Informationen. [Antoniou und van Harmelen, 2008; Berners-Lee u. a., 2001]

Bereits 1999 formulierte Berners-Lee dazu seine Visionen bezüglich dem Web von morgen wie folgt:

> *„I have a dream for the Web ... and it has two parts.*
>
> *In the first part, the Web becomes a much more powerful means for collaboration between people. I have always imagined the information space as something to which everyone has immediate and intuitive access, and not just to brows, but to create. [...] Furthermore, the dream of people-to-people communication through shared knowledge must be possible for groups of all sizes, interacting electronically with as much ease as they do now in person.*
>
> *In the second part of the dream, collaborations extend to computers. Machines become capable of analyzing all the data on the Web - the content, links, and transactions between people and computers. A Semantic Web, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy, and our daily lives will be handled by machines talking to machines, leaving humans to provide the inspiration and intuition.“* [Berners-Lee und Fischetti, 1999]

Für die Umsetzung der Visionen muss das Web mit maschinenverarbeitbarer Semantik erweitert werden. Die Idee dabei ist, dass jede Ressource Informationen in Form von Metadaten über sich selbst bereitstellt. Das Format der Metadaten soll dabei maschinenverständlich (z.B. XML) und das Vokabular der Metadaten wohl definiert sein. Weiters soll Reasoning mit Hilfe der Metadaten möglich sein. [Herman, 2003]

Dies führt uns zu einer Definition von Semantic Web:

> *„The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.“* [Berners-Lee u. a., 2001]

2.2 Architektur

Damit ein besserer Überblick über Semantic Web gegeben wird, beschreiben die folgenden Unterabschnitte die einzelnen Schichten der Architektur (siehe Abbildung 2.1), die für die Realisierung des Semantic Webs notwendig sind. Dabei kann jede Schicht die Funktionalität der darunter liegenden Schicht verwenden bzw. erweitert die darüber liegende Schicht die Funktionalität der darunterliegenden Schicht. [Antoniou und van Harmelen, 2008]

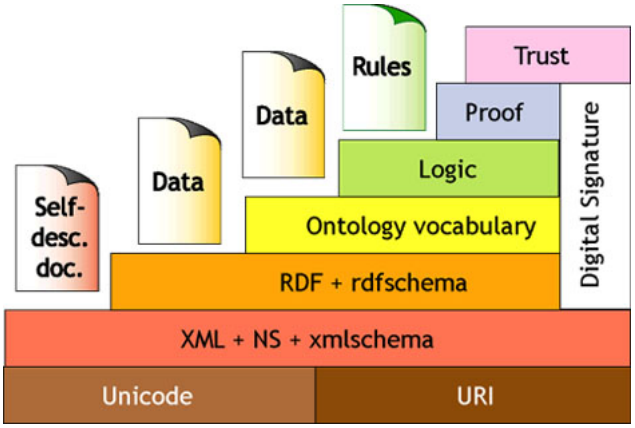

Abbildung 2.1: Architektur des Semantic Webs [Quelle: Berners-Lee, 2000, Folie 10]

2.2.1 Unicode & URI

Die unterste Architekturschicht wird aus den Bereichen *Unicode* und *URI* (Uniform Resource Identifier) gebildet.

Unicode ist ein internationaler Standard, der eine einheitliche Kodierung von Zeichen aus unzähligen Sprachen und Kulturen unterstützt. Dadurch ist es möglich, dass verschiedene Maschinen „alle“ Zeichen lesen bzw. darstellen können. Der Standard wird vom Unicode Consortium gewartet. [Unicode, Inc., 2008]

Ein URI ist eine Zeichenfolge, durch welche physische oder abstrakte Ressourcen identifiziert werden können. Listing 2.1 zeigt die allgemeine Syntax eines URIs.

```
scheme ":" hier-part [ "?" query ] [ "#" fragment ]
```

Listing 2.1: URI Syntax [Quelle: Berners-Lee u. a., 2005]

Scheme gibt den Typ des URIs (z.B. *http*) an, wodurch die Interpretation des darauf folgenden URI-Teiles festgelegt wird.
Der *Hierarchical Part* (*hier-part*) besteht aus dem *Authority*-Teil (siehe Listing 2.2) und dem *Pfad*, wobei der Authority-Teil nur optional anzugeben ist. Der Authority-Teil beinhaltet Benutzerinformationen (z.B. Benutzername und Passwort), *Host* und *Port*. Falls die Benutzerinformationen angegeben werden, so müssen diese vor dem Host mit dem '@'-Zeichen getrennt stehen. Der Host gibt dabei den Domainnamen oder die IP-Adresse eines Servers an. Der Port muss nur angegeben werden, wenn ein anderer als der Standard-Port für das angegebene Scheme verwendet werden soll. Der Pfad identifiziert mit dem optionalen Query-Teil eine Ressource innerhalb des entsprechenden URI Schemes und der Authority.

```
[ userinfo "@" ] host [ ":" port ]
```

Listing 2.2: Authority Syntax [Quelle: Berners-Lee u. a., 2005]

Wie bereits erwähnt, identifiziert der optionale Query-Teil mit der Pfadangabe eine Ressource. Der Query-Teil kann dabei nach dem Pfad und mit einem davor stehenden Fragezeichen angegeben werden.
Der Fragment-Teil wird vom restlichen URI-Teil mit dem #-Zeichen abgetrennt. Er ermöglicht die Referenzierung auf sekundäre Ressourcen, wobei sich die Referenz auf die primäre Ressource bezieht, die durch Scheme, Hierarchical Part und optionaler Query spezifiziert wurde. Ein Beispiel dafür wäre das Anker-Element in HTML. [Berners-Lee u. a., 2005]

```
foo://example.com:8042/over/there?name=ferret#nose

scheme: foo
authority: example.com:8042
path: /over/there
query: name=ferret
fragment: nose
```

Listing 2.3: URI Beispiel mit Authority-, Query- und Fragment-Teil [Quelle: Berners-Lee u. a., 2005]

Die Listings 2.3 und 2.4 zeigen URI-Beispiele, wobei das erste einen Authority-, Query- und Fragment-Teil und das zweite nur einen Pfad zusätzlich zum Scheme enthält.

```
urn:example:animal:ferret:nose

scheme: urn
path: example:animal:ferret:nose
```

Listing 2.4: URI Beispiel nur mit Pfad-Teil [Quelle: Berners-Lee u. a., 2005]

2.2.2 XML + NS + xmlschema

Die zweite Schicht besteht aus der Extensible Markup Language (XML), Namespaces (NS) und XML-Schema.

XML wurde vom World Wide Web Consortium (W3C) entwickelt und ist eine Teilmenge der Standard Generalized Markup Language[1] (SGML). XML ermöglicht die Erstellung strukturierter Webdokumente unter Verwendung von benutzerdefiniertem Vokabular. [W3C u. a., 2006b; Antoniou und van Harmelen, 2008]

XML-Namespaces (NS) werden mit Hilfe von URIs definiert. In den Namespaces können XML-Elemente bzw. -Attribute enthalten sein. Durch die Verwendung von XML-NS können Namenskonflikte von XML-Elementen oder -Attributen mit gleichem Namen aus unterschiedlichen Domänen vermieden werden. [W3C u. a., 2006a]

XML-Schema ermöglicht die Beschreibung der Struktur von XML-Dokumenten, d.h. welche Elemente welche Attribute aufweisen, welche Elemente als Kinder von anderen Elementen erlaubt sind und wie oft diese vorkommen dürfen usw. Es wird dabei die XML-Syntax eingesetzt. [W3C u. a., 2004b]

Für eine ausführlichere Erläuterung zu XML, Namespaces, XML-Schema wird an dieser Stelle auf [Antoniou und van Harmelen, 2008, Kapitel 2] verwiesen.

[1] http://www.w3.org/MarkUp/SGML/

2.2.3 RDF + rdfschema

RDF (Resource Description Framework) bietet ein Basisdatenmodell für das Erstellen von Aussagen über Ressourcen (Webseiten, Dinge, Personen, ...) mit einer XML-basierten Syntax. Diese Aussagen bestehen immer aus den Teilen Subjekt, Prädikat und Object: <Subjekt> hat eine Eigenschaft <Prädikat> mit dem Wert <Objekt>. Das Objekt kann wie das Subjekt eine Ressource sein, die über einen URI identifiziert wird. Das Objekt kann alternativ auch ein Literal (z.B. in Form einer Zeichenfolge) sein.
RDF wurde genauso wie XML von W3C entwickelt bzw. vorgeschlagen. [Antoniou und van Harmelen, 2008; Yu, 2007]

RDF-Schema verfügt über Modellierungskomponenten, womit Ressourcen hierarchisch organisiert werden können. Die Schlüsselkomponenten sind dabei *Classes*, *Properties*, *Subclass/Subproperty* Beziehungen, *Domain/Range* Einschränkungen. RDF-Schema basiert auf RDF und kann auch als primitive Sprache für das Beschreiben von Ontologien (siehe Kapitel 2.2.4 und Kapitel 4) angesehen werden. [Antoniou und van Harmelen, 2008]

Kapitel 3 liefert eine genauere Erklärung der Basiskonzepte zu RDF und RDF-Schema.

2.2.4 Ontology Vocabulary

Zunächst stellt sich die Frage: Was ist eine Ontologie?
Ontologien haben Ihren Ursprung in der Philosophie und beschäftigen sich dort vor allem mit dem Begriff des Seins. [Ehrig, 2006]

In der Informatik gibt es für eine Ontologie folgende Definition:

> *„An ontology is an explicit specification of a conceptualization.“* [Gruber, 1995]

Ein Schlüsselbegriff in der Definition ist dabei *conceptualization* (Konzeptualisierung). Damit ist ein Modell von Phänomenen der wirklichen Welt gemeint, das durch Identifikation der relevanten Konzepte der Phänomene entsteht. Beispiele für Konzepte sind Tiere, Personen, Dinge, aber auch Eigenschaften wie Name oder Farbe. Die Konzepte werden eindeutig (*explicit*) spezifiziert. Alle Konzepte zusammen bilden das Vokabular der Ontologie, welches somit (eine bestimmte Sicht auf) eine Domäne beschreibt. Weiters kann durch Einschränkungen zusätzliches Wissen über die Domäne angegeben

werden. [Ding u. a., 2002; Antoniou und van Harmelen, 2008]

Wie bereits erwähnt kann RDF-Schema als primitive Ontologiesprache angesehen werden. Um jedoch auch komplexere Beziehungen zwischen Objekten beschreiben zu können, ist eine mächtigere Sprache als RDF-Schema erforderlich. Deswegen hat W3C OWL (Ontology Web Language), die aus den Ontologiesprachen DAML und OIL hervorkam, entworfen. [Antoniou und van Harmelen, 2008]

Die Konzepte von OWL werden in Kapitel 4 ausführlich besprochen.

2.2.5 Logic

Die Logic-Schicht ermöglicht eine Erweiterung der darunterliegenden Schicht (Ontology Vocabulary), so dass zusätzlich anwendungsspezifische Regeln definiert werden können. [Antoniou und van Harmelen, 2008]

2.2.6 Proof

Diese Schicht beinhaltet die Repräsentation von Beweisen, die Validierung der Beweisführung sowie auch den Schlussfolgerungsprozess. [Antoniou und van Harmelen, 2008]

2.2.7 Trust & Digital Signature

Die Trust-Schicht geht Hand in Hand mit dem Bereich Digital Signature bzw. den Informationen z.B. von vertrauenswürdigen Software Agenten oder Zertifizierungsstellen. Nur wenn die Daten vertrauenswürdig sind, kann das Web sein ganzes Potential ausschöpfen. [Antoniou und van Harmelen, 2008]

Nachdem die Visionen sowie der Überblick über Semantic Web dargestellt wurden, greift das nächste Kapitel die Architekturschicht *RDF + rdfschema* auf und erläutert dazu vor allem Sprachelemente und Verwendung von RDF(S).

Kapitel 3

Resource Description Framework: RDF

XML ist eine universelle Metasprache und ermöglicht die Verwendung von benutzerdefiniertem Vokabular bzw. benutzerdefinierten Tags. Allerdings gibt es bei XML keine Sprachkonstrukte, die über die Bedeutung (*Semantic*) der Daten Auskunft geben. Es liegt an der jeweiligen Applikation, wie sie entsprechende Tags eines XML-Dokuments interpretiert. [Antoniou und van Harmelen, 2008]

RDF stellt ein Datenmodell zur Verfügung, mit dem Objekt-Eigenschaft-Wert Tripple - so genannte Aussagen (*Statements*) - definiert werden können. Dabei kommt die XML Syntax und das XML-Namespace-Konzept zum Einsatz.
RDF selbst ist domänenunabhängig, d.h. die Sprachkonstrukte bzw. die Syntax sind domänenunabhängig definiert. Der bzw. die BenutzerIn kann seine eigene Terminologie mit Hilfe der Schemasprache RDF-Schema (RDFS) bestimmen. Damit können Semantik eingebracht und in weiterer Folge Schlussfolgerungen abgeleitet werden. Im Gegensatz zu XML-Schema, das die Struktur eines XML-Dokumentes festlegt, wird mit RDF-Schema das Vokabular spezifiziert, welches im RDF Datenmodell erlaubt ist. Mit RDFS kann also z.B. definiert werden, welche Eigenschaft in Verbindung mit welchen Arten von Objekten stehen darf oder wie Beziehungen zwischen Objekten sind. [Antoniou und van Harmelen, 2008; Yu, 2007]

Die nachfolgenden Kapitel beschreiben die Basiskonzepte sowie die Syntax zu RDF und RDF-Schema.

3.1 RDF Basiskonzepte

3.1.1 Ressourcen (*Resources*)

Mit Hilfe von RDF können Daten über Objekte spezifiziert werden. Diese Objekte können alles sein: eine Webseite, ein Teil einer Webseite (z.B. ein Wort auf der Seite) oder auch reale Objekte wie z.B. eine Person, ein Tier oder ein Buch. In RDF werden diese Objekte als Ressourcen bezeichnet. D.h. eine Ressource ist ein Objekt, das mit einem RDF-Ausdruck beschrieben wird.
Eine Ressource wird über einen URI identifiziert, welcher dann auch als Name für diese Ressource verwendet wird. [Yu, 2007]

3.1.2 Eigenschaften (*Properties*)

Eigenschaften sind spezielle Ressourcen, die Relationen zwischen Ressourcen beschreiben. Beispiele für Eigenschaften sind *hat Gewicht*, *hat Name*, *geboren am*, *erstellt von*, usw. Da es sich bei Eigenschaften auch um Ressourcen handelt, werden diese - wie bereits im vorigen Kapitel erwähnt - mit einem URI identifiziert. [Antoniou und van Harmelen, 2008]

3.1.3 Aussagen (*Statements*)

Eine RDF-Aussage beschreibt die Eigenschaften von Ressourcen und hat dabei folgendes Format:

Ressource (Subjekt) + Eigenschaft (Prädikat) + Eigenschaftswert (Objekt)

Der Eigenschaftswert kann entweder ein Literal in Form einer Zeichenfolge oder eine Ressource sein. Die Aussage kann wie folgt interpretiert werden:

<Subjekt> hat eine Eigenschaft <Prädikat>, die den Wert <Objekt> hat

[Yu, 2007]

Die Listings 3.1 und 3.2 zeigen Beispiele für RDF-Aussagen. Dabei werden Aussagen über das Erstellungsdatum und den Ersteller einer HTML-Seite formuliert. Bei der ersten Aussage wird als Objekt ein Literal und bei der zweiten Aussage eine Ressource

verwendet. Die beiden Listings unterscheiden sich nur dadurch, dass bei Ersterem keine Namespacedefinitionen genutzt werden, hingegen beim zweiten Listing schon. Wie in Listing 3.2 ersichtlich ist, können die Aussagen mit Namespaces wesentlich kürzer formuliert werden.

```
1 http://www.example.org/index.html hat ein http://www.example.org/terms/creation-date
      mit dem Wert "August 16, 1999"
2
3 http://www.example.org/index.html hat einen http://purl.org/dc/elements/1.1/creator
      mit dem Wert http://www.example.org/staffid/85740
```

Listing 3.1: RDF-Aussagen mit Literal und Ressource als Objekt [Quelle: W3C u. a., 2004c]

```
1 xmlns:ex="http://www.example.org/"
2 xmlns:exterms="http://www.example.org/terms/"
3 xmlns:dc="http://purl.org/dc/elements/1.1/"
4 xmlns:exstaff="http://www.example.org/staffid/"
5
6 ex:index.html hat ein exterms:creation-date mit dem Wert "August 16, 1999"
7
8 ex:index.html hat einen dc:creator mit dem Wert exstaff:85740
```

Listing 3.2: RDF-Aussagen mit Literal und Ressource als Objekt und mit Verwendung von Namespaces [Quelle: W3C u. a., 2004c]

3.1.4 Reification

Reification bezeichnet in RDF die Möglichkeit Aussagen über Aussagen zu definieren. Nachstehendes Beispiel zeigt eine solche Aussage:

> David Billington glaubt, dass John Smith der Ersteller der Seite http://www.example.org/index.html ist

Solche Aussagen können verwendet werden, um Meinungen bzw. Vertrauen über andere Aussagen auszudrücken, was für bestimmte Applikationen wichtig sein kann. Damit eine Aussage in anderen Aussagen verwendet werden kann, ist es notwendig, dass auch den Aussagen URIs zugewiesen werden, auf welche dann referenziert werden kann. [Antoniou und van Harmelen, 2008]

3.1.5 Datentypen

In RDF ist es möglich typisierte Literale festzulegen. Für Applikationen ist es dadurch klar, welchen Datentyp ein entsprechender Literalwert hat. Vorwiegend werden dazu die Datentypen, welche bereits in XML-Schema vordefiniert sind, verwendet. [Antoniou und van Harmelen, 2008]

3.2 RDF Sprachelemente

In Kapitel 3.1.3 wurden bereits Beispiele für RDF-Aussagen angeführt. Allerdings zeigten diese nur den Aufbau von Aussagen und waren weder in ein vollständiges RDF-Dokument eingebettet noch in der RDF-Syntax formuliert. Dies wird nun im folgenden Teil erläutert.
Jedes RDF-Dokument hat genau ein **rdf:RDF**-Tag, das wiederum eine Menge an `rdf:Description`-Tags (siehe nächster Abschnitt) beinhalten kann. Das `rdf:RDF`-Tag kann dazu verwendet werden Namespaces für das aktuelle RDF-Dokument zu spezifizieren. [Antoniou und van Harmelen, 2008]

```
1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3     xmlns:ex="http://www.example.org/"
4     xmlns:exterms="http://www.example.org/terms/"
5     xmlns:dc="http://purl.org/dc/elements/1.1/"
6     xmlns:exstaff="http://www.example.org/staffid/">
7 ...
8 </rdf:RDF>
```

Listing 3.3: Verwendung des `rdf:RDF`-Tags mit Definition von Namespaces [Quelle: W3C u. a., 2004c]

Listing 3.3 zeigt die Verwendung des `rdf:RDF`-Tags und enthält den Standardnamespace für rdf-Tags und vier benutzerdefinierte Namespaces, die auch schon aus den Beispielen in Kapitel 3.1.3 bekannt sind.

Das **rdf:Description**-Tag ermöglicht die Definition einer RDF-Aussage. Als Attribut kann für dieses Tag entweder `rdf:about` oder ein `rdf:ID` eingesetzt werden.
Wird **rdf:about** verwendet, so wird für den Wert dieses Attributs der URI des Subjekts

der RDF-Aussage zugewiesen. Ein Beispiel für die Verwendung des `rdf:Description`-Tags mit dem `rdf:about`-Attribut zeigt Listing 3.4. Im Listing wird eine Aussage über das Erstellungsdatum einer HTML-Seite formuliert.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:exterms="http://www.example.org/terms/"
  <rdf:Description rdf:about="http://www.example.org/index.html">
    <exterms:creation-date>August 16, 1999</exterms:creation-date>
  </rdf:Description>
</rdf:RDF>
```

Listing 3.4: Verwendung des `rdf:Description`-Tags mit dem `rdf:about`-Attribut [Quelle: W3C u. a., 2004c]

Mit dem **`rdf:datatype`**-Attribut kann beim Prädikat-Tag (in Listing 3.4 ist dies `exterms:creation-date`) - wie bereits in Kapitel 3.1.5 angedeutet wurde - noch der Datentyp für das Literal festgelegt werden. Zu beachten ist dabei die Deklaration des XML-Schema Namespaces über das XML-Element `ENTITY`, damit dieser URI als Attributwert dann verwendet werden kann. Listing 3.5 zeigt dazu ein Beispiel.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:exterms="http://www.example.org/terms/">
  <rdf:Description rdf:about="http://www.example.org/index.html">
    <exterms:creation-date
        rdf:datatype="&xsd;date">1999-08-16</exterms:creation-date>
  </rdf:Description>
</rdf:RDF>
```

Listing 3.5: Verwendung des `rdf:Description`-Tags mit dem `rdf:about`-Attribut und dem `rdf:datatype`-Attribut im Prädikat-Tag [Quelle: W3C u. a., 2004c]

Bei der Verwendung von **`rdf:ID`** ist nur mehr der Fragment-Teil des URIs anzugeben (ohne führendes #-Zeichen). Der URI- vor dem Fragment-Teil ergibt sich aus dem Speicherort des RDF-Dokuments oder durch Angabe eines `xml:base`-Attributs im `rdf:RDF`-Tag. Der äquivalente URI des `rdf:ID`-Attributs aus Listing 3.6 für ein `rdf:about`-Attribut

wäre `#item10245`. [W3C u. a., 2004c]

```
<?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:exterms="http://www.example.com/terms/">
  <rdf:Description rdf:ID="item10245">
    ...
  </rdf:Description>
  ...
</rdf:RDF>
```

Listing 3.6: Verwendung des `rdf:Description`-Tags mit dem `rdf:ID`-Attribut [Quelle: W3C u. a., 2004c]

In den bisherigen Beispielen dieses Kapitels wurden nur Literale als Objekt in den RDF-Aussagen verwendet. Soll nun das Objekt eine Ressource sein, so muss das **`rdf:resource`**-Attribut dem Prädikat-Tag hinzugefügt werden. In Listing 3.7 wird - wie schon in Listing 3.5 - das XML-Element `ENTITY` herangezogen, um Namespaces zu definieren. Durch die Verwendung dieses Konstruktes ist es möglich, dass der Namespace eines entsprechenden URIs als Attributwert eingesetzt werden kann. [W3C u. a., 2004c]
Zunächst wird in Listing 3.7 der Name eines Mitarbeiters festgelegt. Der gleiche Mitarbeiter wird daraufhin in der nächsten Aussage als Objektressource verwendet.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY ex "http://www.example.org/">
  <!ENTITY exstaff "http://www.example.org/staffid/">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex="&ex;"
    xmlns:exstaff="&exstaff;"
    xmlns:exterms="http://www.example.org/terms/"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="&exstaff;85740">
    <exterms:name>John Smith</exterms:name>
  </rdf:Description>
  <rdf:Description rdf:about="&ex;index.html">
    <dc:creator rdf:resource="&exstaff;85740" />
  </rdf:Description>
</rdf:RDF>
```

Listing 3.7: Verwendung des `rdf:resource`-Attributs [Quelle: W3C u. a., 2004c]

In RDF kann mit dem **rdf:type**-Tag innerhalb von `rdf:Description` angegeben werden, von welchem Typ das Subjekt der RDF-Aussage ist. [Antoniou und van Harmelen, 2008]
Listing 3.8 zeigt die gleiche RDF-Aussage wie in Listing 3.7, wobei allerdings explizit mit dem `rdf:type`-Element angegeben wird, dass das Subjekt der Aussage vom Typ Mitarbeiter ist.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY ex "http://www.example.org/">
  <!ENTITY exstaff "http://www.example.org/staffid/">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex="&ex;"
    xmlns:exstaff="&exstaff;"
    xmlns:exterms="http://www.example.org/terms/">
  <rdf:Description rdf:about="&exstaff;85740">
    <rdf:type rdf:resource="&ex;staff" />
    <exterms:name>John Smith</exterms:name>
  </rdf:Description>
</rdf:RDF>
```

Listing 3.8: Verwendung des `rdf:type`-Tags [Quelle: W3C u. a., 2004c]

3.2.1 Container-Elemente

Container-Elemente dienen dazu eine Anzahl von Ressourcen oder Literalen zusammenzufassen. Ein Container ist dann notwendig, wenn z.B. über mehrere Objekte als Ganzes betrachtet eine RDF-Aussage formuliert werden soll. Folgende Containertypen sind in RDF verfügbar [Antoniou und van Harmelen, 2008; W3C u. a., 2004c]:

- **`rdf:Bag`**: repräsentiert eine unsortierte Liste, in der gleiche Elemente auch mehrfach vorkommen dürfen
- **`rdf:Seq`**: repräsentiert eine sortierte Liste, in der gleiche Elemente auch mehrfach vorkommen dürfen
- **`rdf:Alt`**: repräsentiert eine Liste unterschiedlicher Elemente, wobei nur ein Element der Liste gewählt werden kann

Die Elemente eines Containers werden mit dem **rdf:li**-Tag oder mit den Tags **rdf:_1**, **rdf:_2** usw. definiert.
Listing 3.9 zeigt die Verwendung des `rdf:Bag`-Tags, wobei die Liste an Studierende angegeben wird, die den Kurs mit der URI `http://example.org/courses/6.001` besuchen. In Listing 3.10 wird eine Aussage über die FTP-Verfügbarkeit des X11-Packages formuliert. Die unterschiedlichen FTP-Adressen werden dabei mit dem `rdf:Alt`-Element definiert.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">
   <rdf:Description rdf:about="http://example.org/courses/6.001">
      <s:students>
         <rdf:Bag>
            <rdf:li rdf:resource="http://example.org/students/Amy"/>
            <rdf:li rdf:resource="http://example.org/students/Mohamed"/>
            <rdf:li rdf:resource="http://example.org/students/Johann"/>
            <rdf:li rdf:resource="http://example.org/students/Maria"/>
            <rdf:li rdf:resource="http://example.org/students/Phuong"/>
         </rdf:Bag>
      </s:students>
   </rdf:Description>
</rdf:RDF>
```

Listing 3.9: Verwendung des `rdf:Bag`-Tags [Quelle: W3C u. a., 2004c]

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/packages/vocab#">

   <rdf:Description rdf:about="http://example.org/packages/X11">
      <s:DistributionSite>
         <rdf:Alt>
            <rdf:li rdf:resource="ftp://ftp.example.org"/>
            <rdf:li rdf:resource="ftp://ftp1.example.org"/>
            <rdf:li rdf:resource="ftp://ftp2.example.org"/>
         </rdf:Alt>
      </s:DistributionSite>
   </rdf:Description>
</rdf:RDF>
```

Listing 3.10: Verwendung des `rdf:Alt`-Tags [Quelle: W3C u. a., 2004c]

3.2.2 Reification

Wie bereits im Kapitel 3.1.4 erwähnt, gibt es in RDF die Möglichkeit Aussagen über Aussagen zu spezifizieren. Listing 3.11 zeigt ein Beispiel für Reification. Es wird dabei zunächst eine Aussage innerhalb eines `rdf:Description`-Tags mit den Sprachelementen **`rdf:subject`**, **`rdf:predicate`** und **`rdf:object`** definiert. Explizit wird zusätzlich noch mit dem `rdf:type` angegeben, dass es sich um eine RDF-Aussage handelt. Die formulierte Aussage kann danach über den angegebenen URI bei `rdf:about` in einer neuen Aussage verwendet werden. Konkret wird in der zweiten RDF-Aussage festgelegt, dass das Subjekt mit der URI `http://www.example.org/John` das Tutorial auf der entsprechenden HTML-Seite weiterempfiehlt.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY ex "http://www.example.org/">
  <!ENTITY exterms "http://www.example.com/terms/">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:exterms="http://www.example.com/terms/">
  <rdf:Description rdf:about="#x1">
    <rdf:subject rdf:resource="&ex;index.html">
    <rdf:predicate rdf:resource="&exterms;contains" />
    <rdf:object rdf:resource="&ex;tutorial" />
    <rdf:type rdf:resource="http://www.w3c.org/1999/02/22-rdf-syntax-ns#Statement">
  </rdf:Description>
  <rdf:Description rdf:about="&ex;John"/>
    <xterm:recommends rdf:resource="#x1">
  </rdf:Description>
</rdf:RDF>
```

Listing 3.11: Beispiel für Reification [Quelle: W3C u. a., 2004c]

3.3 RDFS Basiskonzepte

Klassen, Eigenschaften sowie Hierarchien von Klassen und Eigenschaften bilden die Basiskonzepte von RDFS, mit denen das Vokabular einer Domäne zusammengesetzt und in RDF verwendet werden kann. Diese Konzepte sind bereits aus dem Bereich der objektorientierten Programmierung bekannt, allerdings gibt es dabei auch Unterschiede. [Antoniou und van Harmelen, 2008]

Klassen in RDF weisen gegenüber Klassen in der objektorientierten Programmierung grundsätzlich keine Unterschiede auf, d.h. Klassen definieren den Typ von konkreten Objekten - den Instanzen. Die Instanzen werden in RDF als Subjekt oder Objekt in den Aussagen verwendet.
Subklassen sind eine Spezialisierung der Basis- bzw. Superklasse(n). Somit kann eine Teilmenge der Instanzen der Superklasse auch einer oder mehreren Subklassen zugeordnet werden.
Ein Beispiel dazu kann aus dem Universitätsbereich geliefert werden: Die Gesamtheit aller Universitätsangestellten bildet eine Basisklasse. Die Subklassen davon wären Professoren, Universitätsassistenten, Personal des technischen Supports, usw.

Die Eigenschaften verbinden Subjekte und Objekte zu einer vollständigen RDF-Aussage. Damit die Gültigkeit von RDF-Aussagen innerhalb der bestimmten Domäne gewährleistet ist, werden Einschränkungen in Bezug auf die Klassen der Subjekte (*Domain*) sowie der Objekte (*Range*) deklariert.
Im Gegensatz zur objektorientierten Programmierung sind die Eigenschaften bei RDFS global definiert. Somit können neue Eigenschaften für Klassen hinzugefügt werden, ohne dass deren interne Struktur verändert werden muss.
Die Hierarchie von Eigenschaften ist ähnlich zu verstehen wie die von Klassen. Um wieder auf das Universitätsbeispiel zurückzukommen: Die Eigenschaft *wird vorgetragen von* aus der Aussage *Der Lehrveranstaltung A wird vorgetragen von Professor B* ist eine Subeigenschaft von *liegt im Zuständigkeitsbereich von*. Da Professor B den Vortrag zur Lehrveranstaltung hält, ist er auch automatisch im Zuständigkeitsbereich. Es können allerdings auch noch weitere Professoren im Zuständigkeitsbereich dieser Lehrveranstaltung sein, da sie z.B. die Übungsbeispiele zu dieser Lehrveranstaltung ausarbeiten und deswegen Ansprechpartner bzw. zuständig für einen gewissen Teil sind. [Antoniou und van Harmelen, 2008]

Ein weiterer Unterschied zur objektorientierten Programmierung ist, dass bei RDF(S) die *Unique-Name-Assumption* nicht gilt. Dies bedeutet, dass Instanzen mit unterschiedlichen URIs nicht automatisch als unterschiedliche Instanzen angesehen werden. [Antoniou und van Harmelen, 2008]

3.4 RDFS Sprachelemente

Die Elemente `rdfs:Class`, `rdfs:Resource`, `rdfs:Literal` und `rdf:Property` sind Instanzen von `rdfs:Class`, somit ist `rdfs:Class` eine Instanz von sich selbst.

- Mit **rdfs:Class** können Typen bzw. Klassen innerhalb der Domäne definiert werden.
- Alle Dinge, die durch RDF beschrieben werden, sind Ressourcen und daher Instanzen von **rdfs:Resource**. Jede Klasse bzw. jedes Element ist eine Subklasse von rdfs:Resource. Somit stellt rdfs:Resource die Wurzelklasse aller Klassen dar.
- **rdfs:Literal** ist die Klasse von Literalen wie Zeichenfolgen oder Zahlen.
- Mit **rdf:Property** können Eigenschaften der Domäne definiert werden, die dann beim Formulieren der RDF-Aussagen verwendet werden können.

[W3C u. a., 2004a; Yu, 2007]

Die Elemente rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain und rdfs:range sind Instanzen von rdf:Property.

- **rdfs:subClassOf** und **rdfs:subPropertyOf** ermöglichen den Aufbau von Hierarchien von Klassen bzw. Eigenschaften. Das jeweilige Element wird innerhalb von rdfs:Class respektive rdf:Property angegeben. Dadurch spezifiziert es die umgebene Klasse/Eigenschaft als eine Subklasse oder -eigenschaft.
- **rdfs:domain** und **rdfs:range** können bei der Deklaration einer Eigenschaft mit rdf:Property verwendet werden und bestimmen damit von welchem/n Typ(en) das Subjekt (rdfs:domain) oder Objekt (rdfs:range) einer RDF-Aussage mit der entsprechenden Eigenschaft in Relation stehen darf.

[W3C u. a., 2004a]

Mit dem Element **rdfs:comment** können Kommentare, die üblicherweise längere Texte sind, zu Ressourcen hinzugefügt werden. Das Element **rdfs:label** ermöglicht die Festlegung eines benutzerfreundlichen Namens bzw. einer Bezeichnung für Ressourcen. [Antoniou und van Harmelen, 2008]

Listing 3.12 zeigt die Verwendung der wichtigsten RDF-Schema-Elemente. Zu beachten ist dabei, dass nun der Namespace für RDF-Schema dazugekommen ist (xmlns:rdfs) und auch die RDF-Schema-Elemente innerhalb von rdf:RDF definiert werden. Neben der Klasse *Lehrveranstaltung* werden die Klassen *Professor* und *Universitaetsassistent* als Subklassen von *Universitaetsangestellter* definiert. Weiters werden Eigenschaften für die Festlegung des Names eines Universitätsangestellten (*heisst*) sowie für die Bestimmung

der Zuständigkeit und der Vortragenden von Lehrveranstaltungen (*liegtImZustaendigkeitsbereichVon*, *wirdVorgetragenVon*) spezifiziert. Die Eigenschaft *wirdVorgetragenVon* ist dabei eine Subeigenschaft von *liegtImZustaendigkeitsbereichVon*, da der bzw. die vortragende(n) Universitätsangestellte(n) auch für die entsprechende Lehrveranstaltung zuständig ist bzw. sind. Alle Zuständigen sind allerdings nicht zugleich die Vortragenden. Z.B. sind oft Universitätsangestellte nur für einen Übungsteil zur Lehrveranstaltung zuständig, da sie Übungsbeispiele vorbereiten und die Studierenden beim Ausarbeiten betreuen. D.h. sie sind nicht die Vortragenden dieser Lehrveranstaltung.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xml:base="http://www.example.org/rdfs">

   <!-- Klassen -->
   <rdfs:Class rdf:about="#Universitaetsangestellter">
    <rdfs:comment>Die Klasse fuer Universitaetsangestellte</rdfs:comment>
    <rdfs:label>Universitaetsangestellter</rdfs:label>
   </rdfs:Class>
   <rdfs:Class rdf:about="#Professor">
    <rdfs:label>Professor</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Universitaetsangestellter"/>
   </rdfs:Class>
   <rdfs:Class rdf:about="#Universitaetsassistent">
    <rdfs:label>Universitaetsassistent</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Universitaetsangestellter"/>
   </rdfs:Class>

   <rdfs:Class rdf:about="#Lehrveranstaltung">
    <rdfs:label>Lehrveranstalung</rdfs:label>
   </rdfs:Class>

   <!-- Eigenschaften -->
   <rdf:Property rdf:about="#heisst">
    <rdfs:domain rdf:resource="#Universitaetsangestellter"/>
    <rdfs:range rdf:resource="&xsd;string"/>
   </rdf:Property>

   <rdf:Property rdf:about="#liegtImZustaendigkeitsbereichVon">
    <rdfs:domain rdf:resource="#Lehrveranstaltung"/>
```

```
35      <rdfs:range rdf:resource="#Universitaetsangestellter"/>
36    </rdf:Property>
37    <rdf:Property rdf:about="#wirdVorgetragenVon">
38      <rdfs:subPropertyOf rdf:resource="liegtImZustaendigkeitsbereichVon"/>
39      <rdfs:domain rdf:resource="#Lehrveranstaltung"/>
40      <rdfs:range rdf:resource="#Professor"/>
41    </rdf:Property>
42  </rdf:RDF>
```

Listing 3.12: Verwendung von RDF-Schema-Elementen

Die Sprachelemente, die durch RDF(S) zur Verfügung gestellt werden, reichen oft nicht aus, um bestimmte Sachverhalte für das Semantic Web zu formulieren. Im nächsten Kapitel werden daher die Grenzen von RDF(S) erläutert und die Ontology Web Language vorgestellt, die in dieser Hinsicht Verbesserungen mit sich bringt.

Kapitel 4

Ontology Web Language: OWL

Die Web Ontology Working Group des W3C[1] identifizierte Anwendungsfälle für das Semantic Web, bei denen weit mehr Ausdrucksstärke notwendig ist, als es RDF und RDF-Schema bieten. Aus diesem Grund definierten sie ausgehend von DAML+OIL[2] die Ontology Web Language (*OWL*). [Antoniou und van Harmelen, 2008]

In diesem Kapitel werden zunächst die Anforderungen an eine Ontologiesprache aufgelistet und darauf folgend die Grenzen der Ausdrucksstärke von RDF/RDF-Schema erläutert und somit die Notwendigkeit von erweiterten Konzepten aufgezeigt.
Der Hauptteil des Kapitels beschäftigt sich mit den Sprachelementen bzw. der Syntax von OWL. Danach wird die Strukturierung von OWL in die drei vorhandenen Subsprachen besprochen.
Es haben sich bereits notwendige Aspekte für zukünftigen Erweiterungen von OWL herauskristallisiert, welche abschließend kurz erläutert werden.

4.1 Anforderungen an eine Ontologiesprache

Eine Ontologiesprache soll eine formale und eindeutige Beschreibung von Konzepten einer bestimmten Domäne ermöglichen. Die Basisanforderungen an eine Ontologiesprache sind nach [Antoniou und Harmelen, 2003] daher

- eine eindeutig definierte Syntax,
- formale Spezifikation und adäquate Ausdrucksmächtigkeit,

[1] http://www.w3.org/2001/sw/WebOnt
[2] http://www.daml.org/2001/03/daml+oil-index.html

- eine einfache Nutzung und
- die Unterstützung von automatischem Schließen.

Die Anforderung, eine eindeutig definierte Syntax zur Verfügung zu stellen ist bereits aus dem Bereich von Programmiersprachen bekannt und auch die Voraussetzung für die Informationsverarbeitung von Computern.
Durch formale Spezifikationen können die Konzepte einer Domäne genau beschrieben werden, d.h. es ist nicht möglich, dass unterschiedliche Interpretationen durch verschiedene Personen bzw. Computer entstehen können. Die formale Spezifikation bringt auch die Möglichkeit über das formulierte Wissen automatisches Schließen durchzuführen. Dabei können z.B. die Klassenzugehörigkeit von Instanzen und die Äquivalenz von Klassen abgeleitet oder auch die Konsistenz der Ontologie überprüft werden. [Antoniou und van Harmelen, 2008]

Entsprechend dem Schichtenmodell der Semantic Web Architektur (siehe Abbildung 2.1 aus Kapitel 2.2) muss eine Ontologiesprache auch Kompatibilität nach unten gewährleisten. Somit ist die Verwendung der Aspekte der darunterliegenden Schichten wie Unicode, URIs, XML-Namespaces usw. wesentlich.

4.2 Grenzen der Ausdrucksstärke von RDF-Schema

RDF und RDFS ermöglichen die Abbildung von Ontologien. Leider beschränken sich die Konzepte von RDF/RDFS dafür im Groben auf die Definition von Hierarchien von Klassen/Eigenschaften sowie Einschränkungen auf die Typen von Subjekt und Objekt bei RDF-Aussagen. Antoniou und van Harmelen haben daher einige fehlende Aspekte, die für die Abbildung von Wissen im Semantic Web wichtig wären, angeführt [Antoniou und van Harmelen, 2008]:

Lokale Einschränkungen von Eigenschaften: Beim Definieren einer Eigenschaft kann mit `rdfs:range` z.B. keine Einschränkung festgelegt werden, die nur manche Klasse betrifft. Ein Beispiel, wo dies zum Einsatz kommen könnte, wäre mit der Eigenschaft *ernährt sich von*: Eine Kuh ernährt sich von Pflanzen, während sich andere Tiere auch von Fleisch ernähren.

Disjunkte Klassen: In RDFS ist es nicht möglich, disjunkte Klasse zu definieren. Z.B. kann *männlich* und *weiblich* nur als Subklasse von Person definiert werden. Es

ist nicht möglich, dass die Instanzen der männlichen Personen disjunkt mit den Instanzen der weiblichen Personen deklariert werden.

Boolesche Kombinationen mit Klassen: Klassen können mit RDFS nicht mit Hilfe von Vereinigung, Durchschnitt oder Kompliment aus anderen Klassen gebildet werden.

Einschränkungen von Kardinalitäten: RDFS bietet keine Möglichkeit, dass eine Eigenschaft nur eine bestimmte Anzahl an Werten annehmen darf bzw. muss. Z.B.: Eine Person hat genau zwei Elternteile.

Spezielle Charakteristika von Eigenschaften: Oft ist es notwendig, dass bestimmte Eigenschaften als transitiv (z.B. die Eigenschaft *ist größer als*), als eindeutig (*unique* - z.B. die Eigenschaft *ist Mutter von*) oder als invers (z.B. die Eigenschaften *isst* und *wird gegessen von*) zu anderen Eigenschaften definiert werden. Diese Möglichkeit wird von RFDS nicht zur Verfügung gestellt.

4.3 OWL Sprachelemente

OWL verwendet die XML-basierte Syntax von RDF bzw. auch einen Teil der Sprachelemente von RDF/RDFS.
Es wurden für OWL bereits auch alternative syntaktische Formen definiert, damit die erstellten Ontologien für den Menschen leichter lesbar sind. Dazu zählen unter anderem eine Form mit XML-basierter Syntax[3], die allerdings nicht den RDF-Aufbau folgt, sowie eine graphische Form basierend auf der Unified Modeling Language[4]. Diese Formen werden allerdings in dieser Arbeit nicht verwendet und somit auch nicht näher erläutert. [Antoniou und van Harmelen, 2008]

Das Wurzelelement einer OWL Ontologie stellt das `rdf:RDF`-Tag dar, welches bereits aus RDF/RDFS bekannt ist.
Die OWL Ontologie beinhaltet als erstes Element oft das optionale **`owl:Ontology`**-Tag, welches Metadaten betreffend der aktuellen Ontologie enthält. Das **`owl:imports`**-Element ermöglicht das Importieren von anderen Ontologien, die damit Teil der aktuellen Ontologie werden und darin verwendet werden können. Listing 4.1 zeigt die Basisstruktur eines OWL Dokuments. Zu beachten ist neben den bisher besprochenen OWL-Sprachelementen der neu hinzugekommene OWL-Namespace.

[3] http://www.w3.org/TR/owl-xmlsyntax/
[4] http://www.uml.org/

In allen folgenden OWL-Beispielen wird diese Basisstruktur vorausgesetzt und nur mehr die Elemente innerhalb des `rdf:RDF`-Tags bzw. nach `owl:Ontology` angegeben.

```
<rdf:RDF
  xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
  <owl:Ontology rdf:about="">
    <rdfs:comment>An example OWL ontology</rdfs:comment>
    <owl:imports rdf:resource="http://www.mydomain.org/persons"/>
    <rdfs:label>University Ontology</rdfs:label>
  </owl:Ontology>
  ...
</rdf:RDF>
```

Listing 4.1: Basisstruktur einer OWL Ontologie [Quelle: Antoniou und van Harmelen, 2008]

4.3.1 Sprachelemente für Klassen

Klassen werden in OWL mit **`owl:Class`** definiert. Will man die Äquivalenz einer Klasse zu einer anderen ausdrücken, so kommt das Element **`owl:equivalentClass`** zum Einsatz. Mit **`owl:disjointWith`** können hingegen disjunkte Klassen definiert werden. [Antoniou und van Harmelen, 2008]

Listing 4.2 zeigt ein Beispiel für die Verwendung der bisher beschriebenen Sprachelemente für Klassen. Für die Bildung einer Hierarchie wird auf das bekannte `rdfs:subClassOf` aus RDFS zurückgegriffen. Im Beispiel werden die Klassen *academicStaffMember*, *faculty*, *professor*, *assistantProfessor* und *associateProfessor* definiert. Weiters wird festgelegt, dass *faculty* die äquivalente Klasse zu *academicStaffMember* ist und die Klassen *professor*, *assistantProfessor* und *associateProfessor* Subklassen von *academicStaffMember* sind. Außerdem ist die Klasse *associateProfessor* disjunkt mit *professor* und *assistantProfessor*, d.h. eine Instanz von *associateProfessor* darf nicht gleichzeitig eine Instanz von *professor* oder *assistantProfessor* sein.

OWL enthält bereits zwei vordefinierte Klassen, nämlich **`owl:Thing`** und **`owl:Nothing`**. Ersteres stellt die allgemeinste Klasse dar und enthält jedes Element. Somit ist jedes Element bzw. jede Klasse eine Subklasse von `owl:Thing`. `owl:Nothing` ist dagegen die

leere Klasse, d.h. jede Klasse ist eine Superklasse davon. [Antoniou und van Harmelen, 2008]

```
<owl:Class rdf:ID="academicStaffMember"/>
<owl:Class rdf:ID="faculty">
  <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>

<owl:Class rdf:ID="professor">
  <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
</owl:Class>
<owl:Class rdf:ID="assistantProfessor">
  <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
</owl:Class>

<owl:Class rdf:ID="associateProfessor">
  <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
  <owl:disjointWith rdf:resource="#professor"/>
  <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>
```

Listing 4.2: Verwendung von `owl:Class`, `owl:equivalentClass` und `owl:disjointWith` [Quelle: Antoniou und van Harmelen, 2008]

Klassen können auch aus Booleschen Kombinationen wie Vereinigung, Durchschnitt und Kompliment gebildet werden. Dazu werden die Konstrukte **`owl:unionOf`**, **`owl:intersectionOf`** und **`owl:complementOf`** verwendet. [Antoniou und van Harmelen, 2008]

```
<owl:Class rdf:ID="noStaffMember">
  <owl:complementOf rdf:resource="#staffMember"/>
</owl:Class>

<owl:Class rdf:ID="peopleAtUni">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#staffMember"/>
    <owl:Class rdf:about="#student"/>
  </owl:unionOf>
</owl:Class>
```

Listing 4.3: Verwendung von Booleschen Kombinationen zur Klassenbildung [Quelle: Antoniou und van Harmelen, 2008]

Listing 4.3 zeigt die Verwendung der Sprachelemente zur Klassenbildung aus Booleschen Kombinationen. Im oberen Teil des Listings wird eine neue Klasse *noStaffMember* als Komplement der Klasse *staffMember* definiert. D.h. eine Instanz von *noStaffMember* kann nicht gleichzeitig eine Instanz von *staffMember* sein. Zu beachten ist, dass der selbe Sachverhalt auch mit dem `owl:disjointWith`-Element ausgedrückt werden könnte. Der untere Teil des Listings bildet eine Klasse *peopleAtUni* aus der Vereinigung der bereits existierenden Klassen *staffMember* und *student*. Somit sind alle Instanzen von *staffMember* und *student* auch in *peopleAtUni* enthalten.

4.3.2 Sprachelemente für Eigenschaften

OWL unterscheidet zwei Arten von Eigenschaften:

Objekteigenschaften (`owl:ObjectProperty`): stellen eine Beziehung zwischen zwei Objekten her (z.B. die Eigenschaft *wird unterrichtet von*).

Datentyp-Eigenschaften (`owl:DatatypeProperty`): stellen eine Beziehung zwischen einem Objekt und einem Datentypwert her (z.B. die Eigenschaft Name, Alter usw.). So wie RDF/RDFS verwendet auch OWL die Datentypen von XML-Schema.

Genauso wie bei Klassen gibt es auch bei Eigenschaften die Möglichkeit, diese als äquivalent zu anderen Eigenschaften auszuzeichnen. Dazu wird das Sprachelement **`owl:equivalentProperty`** herangezogen.
Weiters ist es bei Eigenschaften möglich die Inversität zu anderen Eigenschaften anzugeben und zwar mit **`owl:inverseOf`**. [Antoniou und van Harmelen, 2008]
Ein Beispiel für die Verwendung der hier beschriebenen Sprachelemente für Eigenschaften wird in Listing 4.4 gezeigt. Zunächst wird im Listing eine Datentyp-Eigenschaft mit dem Namen *age* spezifiziert, wobei nur positive ganze Zahlen als Wert erlaubt sind. Da dabei kein `rdfs:domain`-Tag angegeben wurde, kann diese Eigenschaft von allen Klassen verwendet werden. Weiters wird die Objekt-Eigenschaft *involves* definiert, die nur in Verbindung mit Instanzen der Klasse *course* als Domain und mit Instanzen von *academicStaffMember* verwendet werden darf. Die Eigenschaft *isTaughtBy* wird als Subeigenschaft von *involves* sowie mit den gleichen Domain- und Range-Angaben definiert. Jede Instanz von *course*, die die Eigenschaft *isTaughtBy* aufweist, hat somit auch die Eigenschaft *involves*, aber nicht umgekehrt. Im unteren Teil des Listings wird die Objekt-Eigenschaft *teaches* definiert, wobei im Vergleich zu den Eigenschaften *involves* und *isTaughtBy* die Konzepte bei Domain und Range vertauscht sind. Es wird dazu noch festgelegt, dass es sich dabei um die inverse Eigenschaft zu *isTaughtBy* handelt.

Schließlich wird noch die Eigenschaft *lecturesIn* als äquivalente Eigenschaft zu *teaches* spezifiziert.

```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="involves">
  <rdfs:domain rdf:resource="#course"/>
  <rdfs:range rdf:resource="#academicStaffMember"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isTaughtBy">
  <rdfs:domain rdf:resource="#course"/>
  <rdfs:range rdf:resource="#academicStaffMember"/>
  <rdfs:subPropertyOf rdf:resource="#involves"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="teaches">
  <rdfs:domain rdf:resource="#academicStaffMember"/>
  <rdfs:range rdf:resource="#course"/>
  <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="lecturesIn">
  <owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>
```

Listing 4.4: Verwendung von `owl:ObjectProperty`, `owl:DatatypeProperty`, `owl:equivalentProperty` und `owl:inverseOf` [Quelle: Antoniou und van Harmelen, 2008]

4.3.2.1 Einschränkungen

Angenommen es soll eine Klasse C deklariert werden, die aus Instanzen einer Klasse C' besteht, welche bestimmte Eigenschaften erfüllt. In OWL wird solch eine Konstellation so gelöst, dass C als Subklasse von C' deklariert wird, wobei die Instanzen von C' die gewünschte(n) Eigenschaft(en) aufweisen. Üblicherweise bildet dabei C' eine anonyme Klasse, d.h. es wird C' kein expliziter Name bzw. keine explizite ID zugeordnet. [Antoniou und van Harmelen, 2008]
Das Listing 4.5 zeigt die Definition einer solchen Klasse mit Einschränkung bezüglich

einer Eigenschaft. Im Listing wird die Klasse *firstYearCourse* definiert, die nur Instanzen von Kursen aus dem ersten Jahr enthält. Solche Kurse werden nur von Professoren unterrichtet.

```
<owl:Class rdf:about="#firstYearCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:allValuesFrom rdf:resource="#Professor"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 4.5: Verwendung von `owl:Restriction`, `owl:onProperty` und `owl:allValuesFrom` [Quelle: Antoniou und van Harmelen, 2008]

Neue Elemente in Listing 4.5 sind `owl:Restriction`, `owl:onProperty` und `owl:allValuesFrom`.
Mit dem **`owl:Restriction`**-Element wird eine Einschränkung definiert und es enthält genau ein **`owl:onProperty`**-Element, welches auf die entsprechende Eigenschaft referenziert. Nach `owl:onProperty` wird spezifiziert, welche Werte die Eigenschaft annehmen darf. Das können z.B. die Elemente `owl:allValuesFrom`, `owl:someValuesFrom`, `owl:hasValue`, `owl:minCardinality` und `owl:maxCardinality` sein. [Antoniou und van Harmelen, 2008]

`owl:allValuesFrom` besagt, dass alle Werte oder Instanzen der referenzierten Klasse - im Listing 4.5 betrifft dies Instanzen der Klasse *Professor* - miteinbezogen werden, die als Objekt in einer Aussage bezüglich der entsprechenden Eigenschaft existieren. In der Logik wird das Verhalten von `owl:allValuesFrom` als Allquantor bezeichnet.
`owl:someValuesFrom` ist im Gegensatz dazu ein Existenzquantor. D.h. es muss mindestens ein Wert (Literal oder Instanz einer Klasse) - und nicht alle - existieren, der als Objekt in einer Aussage bezüglich der entsprechenden Eigenschaft definiert wurde.
`owl:hasValue` erlaubt einen bestimmten Objektwert (Literal oder Instanz einer Klasse) anzugeben, der in einer Aussage bezüglich der entsprechenden Eigenschaft deklariert worden sein muss. [Antoniou und van Harmelen, 2008]

Ein Beispiel zu `owl:someValuesFrom` und `owl:hasValue` zeigt Listing 4.6. Im oberen Teil des Listings wird die Klasse *academicStaffMember* definiert. Durch die dabei angegebene Einschränkung, darf diese Klasse nur Instanzen entalten, die die Eigenschaft

teaches (siehe Listing 4.4) mit mindestens einer Instanz aus der Klasse *undergraduateCourse* als Range-Wert haben. Bei der unteren Definition wird eine Einschränkung zur Klasse *mathCourse* festgelegt, die besagt, dass sie nur Instanzen enthalten darf, die bezüglich der Eigenschaft *isTaughtBy* (siehe Listing 4.4) beim Range-Wert auf die Instanz *949318* verweist. Dies ist eine Instanz der Klasse *academiceStaffMember*.

```
<owl:Class rdf:about="#academicStaffMember">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#teaches"/>
      <owl:someValuesFrom rdf:resource="#undergraduateCourse"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#mathCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:hasValue rdf:resource="#949318"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 4.6: Verwendung von `owl:someValuesFrom` und `owl:hasValue` [Quelle: Antoniou und van Harmelen, 2008]

Kardinalitätseinschränkungen sind in OWL mit den Elementen **`owl:minCardinality`** und **`owl:maxCardinality`** möglich. In Listing 4.7 wird die Verwendung dieser Elemente veranschaulicht. Dabei wird festgelegt, dass einer Abteilung mindestens 10 und maximal 30 Personen zugeteilt sein müssen.

```
<owl:Class rdf:about="#department">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
      10
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
```

```
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
      30
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 4.7: Verwendung von `owl:minCardinality` und `owl:maxCardinality` [Quelle: Antoniou und van Harmelen, 2008]

4.3.2.2 Spezielle Charakteristika von Eigenschaften

Manchmal ist es notwendig eine Eigenschaft z.B. als transitiv zu deklarieren. In OWL sind folgende Charakteristika-Elemente für Eigenschaften verfügbar [Antoniou und van Harmelen, 2008]:

`owl:TransitiveProperty`: definiert eine transitive Eigenschaft, wie z.B. *ist größer als*

`owl:SymmetricProperty`: definiert eine symmetrische Eigenschaft, wie z.B. *ist das Geschwisterteil von*

`owl:FunctionalProperty`: definiert eine Eigenschaft, die genau einen Wert für eine Instanz hat, wie z.B. *ist im Alter von*, *hat eine Körpergröße von*

`owl:InverseFunctionalProperty`: definiert eine Eigenschaft, bei der zwei unterschiedliche Instanzen nicht den gleichen Wert haben dürfen bzw. bei der ein Wert genau einer Instanz zugeordnet ist, wie z.B. *ist die Sozialversicherungsnummer von*

```
<owl:ObjectProperty rdf:ID="hasSameGradeAs">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
  <rdfs:domain rdf:resource="#student" />
  <rdfs:range rdf:resource="#student" />
</owl:ObjectProperty>
```

Listing 4.8: Verwendung von `owl:TransitiveProperty` und `owl:SymmetricProperty` [Quelle: Antoniou und van Harmelen, 2008]

In Listing 4.8 wird die Eigenschaft *hasSameGradeAs* als symmetrisch und transitiv definiert.

4.3.3 Enumerationen

Für die Definition einer Enumeration wird in OWL das Element **owl:oneOf** verwendet. Listing 4.9 zeigt die Verwendung des owl:oneOf-Tags. Dabei wird die Klasse *weekdays* definiert, welche die Instanzen *Monday*, *Tuesday*, *Wednesday*, *Thursday*, *Friday*, *Saturday* und *Sunday* enthält. Zu Beachten ist, dass hier in Verbindung mit dem owl:oneOf-Element auch das **rdf:parseType**-Element zum Einsatz kommt, um eine Enumeration anzudeuten. [Antoniou und van Harmelen, 2008; Yu, 2007]

```
<owl:Class rdf:ID="weekdays">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Monday"/>
    <owl:Thing rdf:about="#Tuesday"/>
    <owl:Thing rdf:about="#Wednesday"/>
    <owl:Thing rdf:about="#Thursday"/>
    <owl:Thing rdf:about="#Friday"/>
    <owl:Thing rdf:about="#Saturday"/>
    <owl:Thing rdf:about="#Sunday"/>
  </owl:oneOf>
</owl:Class>
```

Listing 4.9: Definition einer Enumeration mit OWL [Quelle: Antoniou und van Harmelen, 2008]

4.3.4 Instanzen

Instanzen von OWL-Klassen werden wie in RDF deklariert. Listing 4.10 zeigt wie eine Instanz der Klasse *academicStaffMember* auf zwei unterschiedliche Arten definiert werden kann. Diese Instanz wurde bereits in Listing 4.6 verwendet. [Antoniou und van Harmelen, 2008]

```
<rdf:Description rdf:ID="949352">
  <rdf:type rdf:resource="#academicStaffMember"/>
</rdf:Description>

<academicStaffMember rdf:ID="949352"/>
```

Listing 4.10: Instanzen von OWL-Klassen erzeugen [Quelle: Antoniou und van Harmelen, 2008]

OWL verfolgt derzeit nicht die *Unique-Names Assumption*, was bedeutet, dass in OWL zwei Instanzen mit unterschiedlichen Namen oder IDs keine unterschiedlichen Instanzen darstellen. Es muss mit **owl:differentFrom** explizit angegeben werden, dass eine Instanz unterschiedlich zu einer anderen ist. Ein Beispiel dazu zeigt Listing 4.11, wo die Instanz *949318* der Klasse *lecturer* definiert und zugleich festgelegt wird, dass diese Instanz unterschiedlich zur Instanz *949352* ist. [Antoniou und van Harmelen, 2008]

```
1 <lecturer rdf:ID="949318">
2   <owl:differentFrom rdf:resource="#949352"/>
3 </lecturer>
```

Listing 4.11: Verwendung von `owl:differentFrom` [Quelle: Antoniou und van Harmelen, 2008]

4.4 Subsprachen von OWL

Um einen Ausgleich zwischen Ausdrucksmächtigkeit und effizienter Unterstützung für automatisches Schließen zu erreichen, wurde OWL in Subsprachen unterteilt, die nachfolgend erläutert werden.

4.4.1 OWL Full

OWL Full beinhaltet alle Sprachelemente von OWL. Es erlaubt auch die Kombination mit RDF/RDFS.
Der Vorteil von OWL Full ist, dass es sowohl semantisch als auch syntaktisch zu RDF abwärtskompatibel ist. Der Nachteil dabei ist, dass die Sprache auf Grund ihrer Mächtigkeit unentscheidbar ist, wodurch vollständiges bzw. effizientes Schließen nicht mehr möglich ist. [Antoniou und van Harmelen, 2008]

4.4.2 OWL DL

OWL DL (Description Logic) unterliegt gegenüber OWL Full folgenden Einschränkungen:

- Vokabular-Trennung: Eine Ressource darf entweder nur eine Klasse, ein Datentyp, eine Datentyp-Eigenschaft, eine Objekteigenschaft, eine Instanz, ein Datenwert oder Teil des vordefinierten Vokabulars sein und nicht mehr.
- Explizite Typisierung: Eine Klasse muss explizit definiert werden, obwohl sie bereits implizit durch die Verwendung von `rdfs:subClassOf` definiert wurde.
- Eigenschaften-Trennung: Objekteigenschaften und Datentypeigenschaften müssen disjunkt sein. Dies impliziert, dass `owl:inverseOf`, `owl:FunctionalProperty`, `owl:InverseFunctionalProperty` und `owl:SymmetricProperty` in Verbindung mit Datentypeigenschaften nicht verwendet werden dürfen.
- Kardinalitätseinschränkungen auf transitive Eigenschaften sind nicht erlaubt.
- Anonyme Klassen sind nur erlaubt als *Domain* und *Range*, entweder bei `owl:equivalentClass` oder `owl:disjointWith`. Und als *Range* (und nicht als *Domain*) bei `rdfs:subClassOf`.

Der Vorteil hier ist, dass die Unterstützung für automatisches effizientes Schließen wieder gewährleistet werden kann, allerdings geht die vollständige Kompatibilität mit RDF verloren. [Antoniou und van Harmelen, 2008]

4.4.3 OWL Lite

Eine OWL Lite Ontologie muss eine OWL DL Ontologie sein und unterliegt außerdem noch nachstehenden Einschränkungen:

- `owl:oneOf`, `owl:disjointWith`, `owl:unionOf`, `owl:complementOf` und `owl:hasValue` sind nicht erlaubt.
- Kardinalitätseinschränkungen dürfen nur mit den Werten 0 und 1 angegeben werden.
- `owl:equivalentClass` darf nicht zwischen anonymen Klassen zum Einsatz kommen.

Der Vorteil dieser Subsprache ist, dass sie durch die Einschränkung an Sprachkonstrukten nicht so komplex ist, wodurch die Implementierung von Softwareprodukten für die

Erstellung bzw. Verarbeitung von OWL Lite Ontologien einfach ist. D.h. es gibt genügend Softwareprodukte, die die Sprachkonstrukte von OWL Lite vollständig unterstützen. Natürlich leidet darunter jedoch die Ausdrucksstärke. [Antoniou und van Harmelen, 2008]

Folgende aufsteigende Kompatibilitäten zwischen den drei Subsprachen sind gegeben [Antoniou und van Harmelen, 2008]:

- Jede gültige OWL Lite Ontologie ist eine gültige OWL DL Ontologie.
- Jede gültige OWL DL Ontologie ist eine gültige OWL Full Ontologie.
- Jede gültige OWL Lite Aussage ist eine gültige OWL DL Aussage.
- Jede gültige OWL DL Aussage ist eine gültige OWL FULL Aussage.

4.5 Zukünftige Erweiterungen

Der derzeitige Sprachumfang von OWL soll keineswegs der Endpunkt im Bereich Ontologiesprachen für Semantic Web sein. Einige neue Erweiterungen wurden bereits im OWL Anforderungsdokument zusammengefasst, andere Aspekte werden diskutiert. Ein Ausschnitt von Aspekten, die verbesserungswürdig sind, wird nun kurz erläutert [Antoniou und van Harmelen, 2008]:

Import und Module: Derzeit wird beim Import von Ontologien die vollständige Ontologie hinzugefügt. Es ist nicht möglich nur eine Teilmenge davon zu importieren. Ein weiteres Bestreben dabei ist, die aus Programmiersprachen bekannte modulare Struktur mit *Information Hiding* als Basis für den Importmechanismus zu verwenden.

Closed-World Assumption: OWL verfolgt derzeit die *Open-World Assumption*. Diese besagt, dass nicht automatisch angenommen werden kann, dass etwas nicht existiert, nur weil es nicht definiert wurde. Anders ausgedrückt kann nicht automatisch angenommen werden, dass eine Behauptung wahr ist, wenn das Gegenteil dieser Behauptung nicht bewiesen wurde. Auf Grund der oft nur teilweise verfügbaren Informationen im World Wide Web ist diese These notwendig. Allerdings kann auch die *Closed-World Assumption* (alles was nicht definiert wurde, ist falsch) für bestimmte Anwendungen nützlich sein.

Unique-Names Assumption: Üblicherweise werden Instanzen mit verschiedenen Namen oder IDs auch als unterschiedlich angesehen. In OWL wird die *Non-Unique-Names Assumption* angewendet, wodurch unterschiedlich benannte bzw. mit unterschiedlichen IDs versehene Instanzen nicht automatisch als unterschiedlich angesehen werden. Die Unterschiedlichkeit muss explizit festgelegt werden (siehe Listing 4.11). Für das WWW ist dies auch plausibel. Jedoch kann es in manchen Situationen trotzdem notwendig sein auf *Unique-Name Assumption* umzuschalten.

Ein weiterer Schritt in die Richtung der Visionen von Semantic Web ist das Kombinieren von Wissen, welches durch Ontologien beschrieben ist. Diesem umfangreichen Gebiet widmet sich das nächste Kapitel und legt den Fokus vor allem auf Ontology Alignment.

Kapitel 5

Ontology Alignment

Dieser Abschnitt liefert zunächst eine formale Definition einer Ontologie, die für das Verständnis des weiteren Verlaufs dieser Arbeit notwendig ist. Daraufhin wird angeführt, auf welchen Ebenen semantische Heterogenität beim Verknüpfen von Wissen aus verschiedenen Ontologien auftreten kann. Nachdem Begriffserklärungen im Bereich von Ontology Mediation erläutert wurden, erfolgt die Beschreibung der Grundlagen zu Ontology Alignment. Dabei wird zuerst ein Ausgangsbeispiel für die Durchführung des Ontology Alignments vorgestellt, dass in den weiteren Abschnitten der Arbeit für Erklärungen herangezogen wird. Die nächsten Unterabschnitte erläutern Ontology Similarity und das Schichtenmodell für Ähnlichkeitsfunktionen. Die Grundlagen zu Ontology Alignment werden mit einem Auszug aus vorhandenen Ähnlichkeitsfunktionen abgeschlossen. Die darauf folgenden Unterkapitel beschreiben Anwendungsbereiche von Ontology Alignment, den allgemeinen Ontology Alignment Prozess sowie Aspekte zur Evaluierung von Ontology Alignment Algorithmen.

5.1 Ontologie

In den vorangegangenen Kapiteln wurde bereits in rein textueller Form eine Ontologie definiert sowie auch die Verwendung von Beschreibungssprachen für Ontologien mit Beispielen erläutert. In [Ehrig und Sure, 2004] wird eine Ontologie O in formaler Form mit folgendem Tupel definiert:

$$O := (C, H_C, R_C, H_R, I, R_I, A)$$

Die Ontologie O besteht aus Konzepten C, welche Objekte der realen Welt darstellen. Die

Konzepte sind dabei in einer bestimmten Hierarchie H_C strukturiert. Weiters bestehen Relationen R_C zwischen Konzepten, die wiederum in einer Hierarchie H_R aufgebaut sind. Die konkreten Instanzen I sind über Instanzen von Relationen R_I verbunden. A repräsentiert Axiome, die herangezogen werden können, um weiteres Wissen über die abgebildete Domäne abzuleiten. [Ehrig und Sure, 2004]

Im weiteren Verlauf dieser Arbeit bezeichnet E die Menge der Entitäten bzw. das Vokabular einer Ontologie. Eine Entität (*Entity*) $e \in E$ im Sprachgebrauch einer Ontologie O ist entweder ein Konzept, eine Relation oder eine Instanz:

$$e_{|O} \in C \cup R \cup I$$

Ist auf Grund des Kontextes klar auf welche Ontologie sich die Entität bezieht, so wird einfach e statt $e_{|O}$ geschrieben.

5.2 Semantische Heterogenität - Ontology Mismatch

Damit die in Semantic Web gelegten Hoffnungen (siehe Kapitel 2) zumindest teilweise erfüllt werden können, ist das Kombinieren bzw. die Verknüpfung von Metadaten und Wissen aus verschiedenen Quellen notwendig, um im weiteren Schritt Schlussfolgerungen ableiten und automatisierte Datenverarbeitung durchführen zu können. Für Semantic Web spielt somit das Kombinieren von unabhängig erstellten Ontologien eine zentrale Rolle. [Ding u. a., 2002] Auf Grund von semantischer Heterogenität ergeben sich dabei allerdings Probleme, die zu bewältigen sind. Diese Probleme entstehen, wenn z.B. die vorliegenden Ontologien, basierend auf unterschiedlichen Sichten, auf die abzubildende Domäne erstellt wurden, oder auch wenn der Detailgrad der Wissensbeschreibung zwischen den Ontologien variiert. [Bouquet u. a., 2004]
In [Klein, 2001] wird dieses Problem der semantischen Heterogenität zwischen den Ontologien aufgegriffen und unter dem Begriff *Ontology Mismatch* zusammengefasst. Ontology Mismatch wird dabei weiter in Mismatch auf Sprachebene und Mismatch auf Ontologieebene unterteilt.

5.2.1 Mismatch auf Sprachebene

Auf dieser Ebene werden Mismatches zusammengefasst, die auf Grund unterschiedlicher Wissensrepräsentationssprachen der entsprechenden Ontologien entstehen. [Su, 2004]

Die Sprachebene beinhaltet vier Typen von Mismatches [Klein, 2001]:

- **Syntax:** Es liegt eine syntaktische Heterogenität vor, wenn die gleichen Konzepte mit unterschiedlichen syntaktischen Konstrukten beschrieben werden. Z.B. kann in RDFS eine Klasse mit `rdfs:Class` und in OWL mit `owl:Class` definiert werden. [Klein, 2001]
- **Logische Repräsentation:** Abhängig von vorhandenen Konstrukten der verwendeten Sprache müssen gewisse Sachverhalte auf verschiedene Art und Weise umgesetzt werden. Beispielsweise kann in bestimmten Sprachen die Disjunktheit von zwei Klassen explizit angegeben werden (`A ist disjunkt mit B`), wohingegen in anderen Sprachen dies nur mit dem Umweg über Subklassen möglich ist (`A ist eine Subklasse von NOT B` und `B ist eine Subklasse von NOT A)`.
  [Klein, 2001]
- **Bedeutung von primitiven Sprachelementen:** Es kann vorkommen, dass in zwei verschiedenen Sprachen gleiche Namen für Sprachkonstrukte definiert sind, deren Bedeutung sich allerdings unterscheidet. In der OIL RDF Schema Syntax werden z.B. mehrere `<rdfs:domain>`-Tags bezüglich einer Eigenschaft so interpretiert, dass diese Eigenschaft nur einen Wert aus dem *Durchschnitt* der angegebenen Domänen aufweisen darf. RDF-Schema interpretiert hingegen dieses Sprachkonstrukt so, dass der Wert aus der *Vereinigung* der Domänen sein muss. [Klein, 2001]
- **Sprachausdrucksstärke:** Einige Wissensrepräsentationssprachen enthalten Konstrukte, mit denen die Negation ausgedrückt werden kann. Mit anderen Sprachen kann die Negation widerum nicht ausgedrückt werden. Weitere Beispiele für solche Unterschiede in der Ausdrucksstärke wären z.B. die Unterstützung zur Definition von Listen, Mengen und Standardwerten. [Klein, 2001]

5.2.2 Mismatch auf Ontologieebene

Die Ontologieebene beinhaltet Mismatches, die beim Kombinieren von Ontologien auch dann auftreten können, wenn die zugrunde liegenden Ontologien mit der gleichen Repräsentationssprache verfasst wurden [Klein, 2001]:

- **Konzeptualisierung:** Man spricht im Englischen von *Scope Mismatch*, wenn es zunächst aussieht als würden zwei Klassen dasselbe Konzept repräsentieren, allerdings nicht exakt die gleichen Instanzen haben. Ein Beispiel dazu wäre die Klasse

Mitarbeiter. Oft werden leicht unterschiedliche Konzepte von *Mitarbeiter* abgebildet.
Auf Grund unterschiedlicher Sichtweisen der Ontologieersteller auf die abzubildende Domäne kann sich der Abdeckungsgrad von Konzepten oder auch der Detailgrad von Konzeptbeschreibungen wesentlich unterscheiden. Ein Beispiel, wann Unterschiede bezüglich dem ersten Aspekt auftreten könnten, wäre, wenn eine Ontologie aus der Sicht eines Universitätsmitarbeiters und eine Ontologie aus der Sicht eines Studenten erstellt werden. Der als Zweites angesprochene Unterschied liegt z.B. dann vor, wenn die erste Ontologie ein Konzept *Person* aufweist und die zweite Ontologie zwischen den Konzepten *Kinder*, *Erwachsene* und *Senioren* unterscheidet. [Klein, 2001; de Bruijn u. a., 2006]

- **Modellierungsstil:** Zeit, Aktivitäten, Abläufe, Pläne usw. können durch unterschiedliche Paradigmen repräsentiert werden. Z.B. kann in einer Ontologie die Zeit durch ein Zeitintervall und in der anderen Ontologie durch einen konkreten Zeitpunkt angegeben sein.
  Ein weiterer Aspekt, der dem Modellierungsstil zugeteilt wird, ist die Heterogenität, die aus unterschiedlichen Entscheidungen für die Modellierung von Konzepten resultiert. Ein Beispiel hierfür wäre, dass für die Erstellung einer neuen Klasse entweder eine seperate Klasse eingeführt wird oder eine bestehende Klasse, um ein spezielles Attribut erweitert wird, mit dem die Zugehörigkeit zu Typ A oder Typ B ausgedrückt werden kann. [Klein, 2001]

- **Terminologie:** In den vorliegenden Ontologien können gleiche Konzepte mit unterschiedlichen Bezeichnungen definiert werden. Man spricht von *Synonymen*. Ein Beispiel dafür wäre *Car* und *Automobile*. Es kann natürlich auch der umgekehrte Fall vorkommen, d.h. dass Konzepte mit gleichen Bezeichnungen in den verschiedenen Ontologien festgelegt wurden, aber auf Grund des Kontextes in der jeweiligen Ontologie eine andere Bedeudeutung haben. Man spricht dann von *Homonymen*. Im Englischen wäre dazu der *Conductor* ein Beispiel, der im musikalischen Kontext als Dirigent und im elektronischen Kontext als Leitung übersetzt wird. [Klein, 2001]

- **Kodierung:** Als letzter Aspekt für die Unheinheitlichkeit zwischen Ontologien wird in [Klein, 2001] die Kodierung von Werten angeführt. Z.B. kann ein Datum im Format „dd/mm/yyy“ oder auch im Format „mm-dd-yy“ vorliegen. Diese Heterogenität lässt sich aber relativ leicht auflösen. [Klein, 2001]

5.3 Ontology Mediation

Das Abgleichen von Unterschieden zwischen heterogenen Ontologien bildet ein breites Forschungsgebiet, welches unter dem Begriff *Ontology Mediation* zusammengefasst wird. Dies beinhaltet das Ermitteln von korrespondierenden Entitäten aus den Ontologien sowie die Spezifikation der Abbildung von der Entität der einen Ontologie auf die korrespondierende Entität der anderen Ontologie. Weiters zählt dazu auch die Durchführung und Verwendung dieser Abbildungen, damit applikationsspezifische Aufgaben erledigt werden können. Zu solchen Aufgaben zählen z.B. automatisierte Kombination von Flugbuchungs- und Hotelreservierungssystemen, sowie automatisierte Terminvereinbarung zwischen Geschäftspartnern mit Hilfe von Software Agenten. Weitere Beispiele werden in Kapitel 5.5 erläutert. Außerdem beinhaltet Ontology Mediation auch das Zusammenfügen von Ontologien. [de Bruijn u. a., 2005; Ehrig, 2006]
Ontology Mediation wird in folgende Bereiche unterteilt [Ehrig, 2006]:

- **Ontology Merging:** Ziel dabei ist, dass das Wissen aus mehreren verschiedenen Ontologien zusammengefügt und daraus eine neue Ontologie gebildet wird. Die Domäne der Quellontologien und der neuen Ontologie ist dabei gleich. Auf Grund des oft unterschiedlichen Detailgrades der Konzeptbeschreibungen in den Quellontologien steht man vor der Herausforderung die taxonomischen Strukturen in der neuen Ontologie zu vereinheitlichen. Wenn nicht alle Wissenüberlappungen zwischen den zugrunde liegenden Ontologien gefunden werden, besteht die Gefahr, dass das abgebildete Wissen in der Ergebnisontologie redundant, nicht koherent sowie nicht konsistent ist. [Pinto u. a., 1999]
- **Ontology Integration:** Im Gegensatz zu Ontology Merging wird hier eine neue Ontologie für die Domäne D erstellt, die sich aus der Kombination von Ontologien aus unterschiedlichen Domänen $D_1, D_2, ...D_n$ zusammensetzt. Obwohl die Domänen der Quellontologien als auch die Domäne der neuen Ontologie verschieden sind, kann es Beziehungen zwischen den Ontologie geben, die im Zuge von Ontology Integration gefunden werden müssen. Unter anderem auf Grund von möglicherweise gleicher Terminologie mehrerer Ontologien, aber anderer Bedeutung in den zugrundeliegenden Domänen, ist es hier besonders komplex diese Beziehungen zwischen den Ontologien zu erfassen. Auch bei Ontology Integration müssen Probleme wie z.B. variierender Detailgrad an Konzeptbeschreibungen der Quellontologien und Konsistenz der Ergebnisontologie bewältigt werden. [Pinto u. a., 1999]
- **Ontology Mapping:** Für jede Entität in der ersten Ontologie wird versucht eine korrespondierende Entität mit gleicher oder sehr ähnlicher Bedeutung in der zwei-

ten Ontologie zu finden. Ein Mapping repräsentiert somit die semantische Relation zwischen Ontologien. Die zugrundeliegenden Ontologien bleiben dabei unverändert und es werden Mapping Axiome, welche die Relation zwischen den Entitäten beschreiben, separat von der Ontologie gespeichert. Üblicherweise können die ermittelten Mappings nur in eine Richtung angewandt werden. Z.B. die Instanzen von Konzept C_1 Ontologie O_1 können Instanzen des Konzepts C_2 in Ontologie O_2 sein, aber nicht umgekehrt. [Ehrig und Sure, 2004; Ehrig, 2006]

- **Ontology Alignment:** Zwei Ontologien werden gegenseitig abgestimmt, sodass sie zueinander konsistent und kohärent sind. Dies kann eine Transformation der zugrundeliegenden Ontologien inkludieren. D.h. überflüssige Informationen werden entfernt oder fehlende Informationen müssen hinzugefügt werden. Während sich Ontology Mapping lediglich mit der Ermittlung von Relationen zwischen ähnlichen bzw. übereinstimmenden Entitäten aus den vorliegenden Ontologien befasst, liegt der Fokus bei Ontology Alignment auch bei der Repräsentation und Anwendung von gefundenen Abbildungen für bestimmte Aufgaben. Somit wird Ontology Mapping als Subschritt von Ontology Alignment gesehen. [Klein, 2001; Ehrig, 2006; Lanzenberger und Sampson, 2006a]

Existieren zwischen den vorliegenden Ontologien Heterogenitäten auf der Sprachebene, so werden diese oft durch den *Normalisierungsprozess* (*Normalization*) beseitigt, bevor die korrespondierenden Entitäten ermittelt werden. D.h. Ontology Mapping bzw. Ontology Alignment behandelt üblicherweise nur die Heterogenität auf der Ontologieebene. [Noy, 2004]

Auf Grund des Fokus' dieser Arbeit wird im weiteren Verlauf nur der Bereich um Ontology Alignment behandelt.

5.4 Grundlagen zu Ontology Alignment

Eine Kernaufgabe von Ontology Alignment ist es Ähnlichkeiten bzw. Übereinstimmungen zwischen Ontologien zu finden. Die Ähnlichkeiten können dabei von unterschiedlicher Ausprägung sein. Z.B. Gleichheit, Subklassen (z.B. definiert durch *is-a*-Beziehungen), Objekte als Teil eines komplexen Objektes usw. [van Hage, 2008] Ontology Alignment ist daher für die Vision der semantischen Interoperatibilität, damit ist die gemeinsame Nutzung von Services sowie Austausch von digitalem Wissen gemeint, äußerst wichtig. [Hughes und Ashpole, 2005]

Oft enthalten Ontologien ein Vokabular mit mehr als 10000 Konzepten und unzähligen Relationen zueinander. Manuelles Ontology Alignment würde dafür Jahre dauern. Daher will man natürlich automatisierte Ontology Alignment Algorithmen einsetzen. [van Hage, 2008] In [Li u. a., 2007] werden dazu folgende Herausforderungen angeführt:

- korrekte Ähnlichkeiten finden
- effizientes Auffinden von korrespondierenden Entitäten
- ausreichende Benutzerinteraktion beim Ontology Alignment Vorgang einbringen, weil ein rein automatisiertes Alignment oft nicht möglich ist
- automatische Anpassung der Strategien für das Finden von Übereinstimmungen bei bestimmten Aufgaben bzw. Domänen, da dabei die Charakteristika der vorliegenden Ontologien unterschiedlich sein kann
- einfache Parameterisierung, weil die Genauigkeit der Ergebnisse mit verschiedenen Parametern, wie z.B. Gewichtungen von Ähnlichkeiten und Schwellwerten, variiert.

Der Ontology Alignment Prozess aus einem abstrakten Blickwinkel betrachtet, wird in [Ehrig u. a., 2006a] bzw. [Ehrig, 2006] als eine Funktion gesehen, die wie folgt definiert ist.

Definition 1 (Ontology Alignment Function)**.** *An ontology alignment function, align, based on the vocabulary, E, of all entities $e \in E$ and based on the set of possible ontologies, O, is a partial function*

$$align : E \times O \times O \rightharpoonup E,$$
$$with\ \forall e \in O_1(\exists f \in O_2 : align(e, O_1, O_2) = f \vee align(e, O_1, O_2) = \perp).$$

Anstelle von $align(e, O_1, O_2)$ kann auch $align_{O_1,O_2}(e, f)$ geschrieben werden. Wurde zur Entität e aus O_1 eine (teilweise) korrespondierende Entität f in O_2 gefunden, so sagt man im Englischen „*entity e is aligned with entity f*“, d.h. $align_{O_1,O_2}(e, f) \Leftrightarrow align_{O_1,O_2}(e) = f$. Die tief stehende Angabe der Ontologien kann wieder weggelassen werden, sofern auf Grund des Kontextes auf die betroffenen Ontologien geschlossen werden kann. Ein Entitätspaar (e, f), für das noch kein Abgleich stattgefunden hat bzw. wenn dieses Paar gegenüber entsprechenden Alignmentkriterien nocht nicht getestet wurde, wird als *Candidate Alignment* bezeichnet. [Ehrig u. a., 2006a; Ehrig, 2006]

Als Ergebnis des Ontology Alignments erhält man eine Menge von Tupeln $\{(id, e, e', n, R)\}$, was in der Literatur oft als *Alignment* bezeichnet wird. id ist ein eindeutiger Bezeichner

für die gefundene Übereinstimmung, e und e' sind Entitäten aus zwei unterschiedlichen Ontologien, n ist ein Konfidenzmaß in Form einer reellen Zahl zwischen 0 und 1, welches die mathematische Wahrscheinlichkeit für die Richtigkeit der gefundenen Übereinstimmung angibt. R stellt die Relationsart der korrespondierenden Entitäten dar. [Shvaiko und Euzenat, 2005; Martínez-Gil u. a., 2008]

Der Ontology Alignment Prozess aus abstrakter Sichtweise wird in [Bouquet u. a., 2004] ebenfalls als Funktion definiert. Dabei werden allerdings auch andere Eingabedimensionen für den Prozess angeführt:

$$A' = f(o, o', A, p, r)$$

Die grafische Darstellung zu dieser Funktion wird in Abbildung 5.1 veranschaulicht.

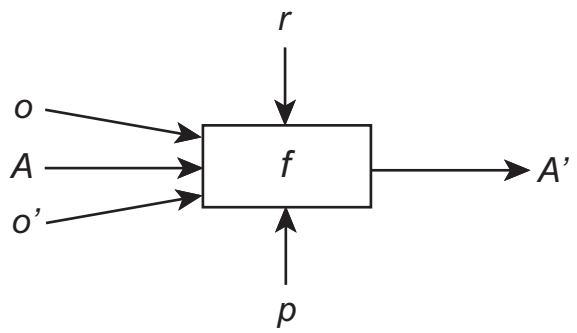

Abbildung 5.1: Ontology Alignment aus abstrakter Sichtweise als Funktion dargestellt [Quelle: Bouquet u. a., 2004]

- f repräsentiert die Ontology Alignment Funktion.
- o und o' sind die beiden Ontologien, die den Ontology Alignment Prozess durchlaufen sollen.
- A ist ein optionaler Eingabeparameter und repräsentiert eine Menge von Alignments, die erweitert bzw. verfeinert werden sollen und die entweder aus einem vorherigen Ontology Alignment Prozess resultierten oder von BenutzerInnen definiert wurden.
- p stellt Parameter für den Prozess dar. Diese können Gewichtungen für bestimmte Ähnlichkeitsarten, Schwellwerte usw. festlegen.

- r sind Ressourcen, die beim Ontology Alignment verwendet werden wie z.B. Wörterbücher.
- A' ist das Ergebnis des Ontology Alignments und repräsentiert die Menge der gefundenen Alignments.

5.4.1 Ontology Alignment Beispiel

Damit im weiteren Verlauf dieser Arbeit die Methoden des Ontology Alignments verständlicher erklärt werden können, werden zwei Ontolgoien aus der Automobil-Domäne verwendet. Abbildung 5.2 zeigt neben der Erklärung zu den grafischen Elementen der Ontologieabbildungen (im oberen linken Rechteck) die beiden Ontologien (in den Elipsen) vor der Durchführung des Ontology Alignments.
Konzepte bzw. Klassen sind durch gelbe Rechtecke, Relationen bzw. Eigenschaften durch blaue Sechsecke und Instanzen durch orange Elipsen dargestellt. Eine Beziehung zwischen einem Super- und Subkonzept ist durch einen Pfeil mit durchgezogener Linie vom Sub- zum Superkonzept visualisiert. Eine Relation hat von der Domain einen eingehenden Pfeil mit strichlierter Linie und einen Pfeil ebenfalls mit strichlierter Linie zur Range. Eine Konzept- sowie eine Relationsinstanz ist durch eine punktiert Linie mit einem Pfeil dargestellt.

Ontologe 1 beinhaltet die Konzepte *Object*, *Vehicle*, *Boat*, *Car*, *Owner* und *Speed*, die Relationen *belongsTo* und *hasSpeed* sowie die Instanzen *Marc*, *Porsche KA-123* und *300 km/h*. Zwischen *Object*, *Vehicle* und *Boat* bzw. *Car* bestehen *is-a*-Beziehungen. D.h. *Vehicle* ist ein *Object*, *Boat* ist ein *Vehicle* als auch ein *Object* und *Car* ist ein *Vehicle* sowie auch ein *Object*. Jedes *Vehicle* hat einen Besitzer und eine Maximalgeschwindigkeit. Betrachtet man die *Ontologe 1* auf Instanzebene, so sieht man, dass *Marc* den *Porsche KA-123* besitzt und dieses Auto eine Maximalgeschwindigkeit von *300 km/h* aufweist.

Ontologe 2 enthält die Konzepte *Thing*, *Vehicle*, *Automobile*, *Volkswagen*, *Porsche*, *Motor* und *Speed*, die Relationen *hasMotor* und *hasProperty* als auch die Instanzen *Marc's Porsche*, *Motor123456* und *fast*. Zwischen *Thing*, *Vehicle*, *Automobile* und *Volkswagen* bzw. *Porsche* als auch zwischen *Thing* und *Motor* sind *is-a*-Beziehungen vorhanden. Somit ist *Vehicle* ein *Thing*, *Automobile* ein *Vehicle* und ein *Thing*, *Motor* ein *Thing* usw. Jedes *Automobile* hat einen *Motor*, wobei beiden Konzepten eine Eigenschaft bezüglich der Gewischwindigkeit zugewiesen wird. Auf Instanzebene hat *Marc's Porsche* den Motor *Motor123456*. Dabei ist *Marc's Porsche* und auch dem Motor *Motor123456*

die Geschwindigkeits-Eigenschaft *fast* zugeordnet.

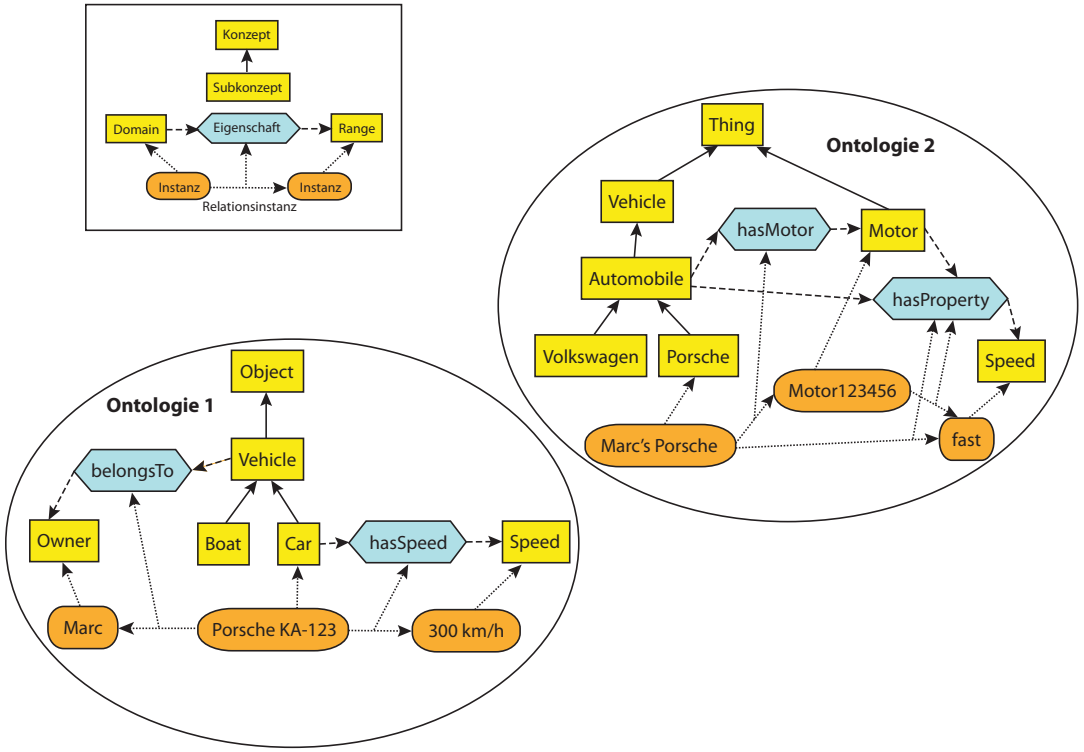

Abbildung 5.2: Zwei Ontologien vor dem Ontology Alignment [Quelle: Ehrig, 2006]

5.4.2 Ontology Similarity

Wie bereits beschrieben, wird im Zuge des Ontology Alignments versucht Ähnlichkeiten zwischen Entitäten aus verschiedenen Ontologien zu finden. Um diese Ähnlichkeiten zu ermitteln werden so genannte *Ähnlichkeitsfunktionen* (im Englischen *Similarity Functions*) verwendet. [Ehrig, 2006] Nach [Ehrig, 2006] wird eine Ähnlichkeitsfunktion wie folgt definiert.

Definition 2 (Similarity Function)**.** *A similarity function*

$$sim : \mathfrak{P}(E) \times \mathfrak{P}(E) \times O \times O \rightarrow [0, 1]$$

is a function that maps a pair of entity sets (expressed through the power set $\mathfrak{P}(E)$ of entities) and their corresponding ontologies O to a real number expressing the similarity between two sets such that

- $\forall e, f \in \mathfrak{P}(E),\ O_1, O_2 \in O,\ sim(e, f, O_1, O_2) \geq 0$ *(positiveness)*
- $\forall e, f, g \in \mathfrak{P}(E),\ O_1, O_2 \in O,\ sim(e, e, O_1, O_2) \geq sim(f, g)$ *(maximality)*
- $\forall e, f \in \mathfrak{P}(E),\ O_1, O_2 \in O,\ sim(e, f, O_1, O_2) = sim(f, e, O_1, O_2)$ *(symmetry)*
- $\forall e, f \in \mathfrak{P}(E),\ O_1, O_2 \in O,\ sim(e, f, O_1, O_2) = 1 \Leftrightarrow e = f$: *Two entity sets are identical.*
- $\forall e, f \in \mathfrak{P}(E),\ O_1, O_2 \in O,\ sim(e, f, O_1, O_2) < 1$: *Two entity sets are similar/ different to a certain degree.*
- $\forall e, f \in \mathfrak{P}(E),\ O_1, O_2 \in O,\ sim(e, f, O_1, O_2) = 0 \Leftrightarrow e \neq f$: *Two entity sets are different and have no common characteristics.*

Diese Definition sagt aus, dass eine Ähnlichkeitsfunktion eine Menge an Entitäten der ersten Ontologie mit einer Menge an Entitäten der zweiten Ontologie vergleicht und die Ähnlichkeit der Mengen durch eine reelle Zahl zwischen 0 und 1 ausdrückt. In dieser Arbeit beinhalten die beiden Mengen bei der Erläuterung von Methoden bzw. Ähnlichkeitsfunktionen des Ontology Alignments fast ausschließlich jeweils nur eine Entität.
Die von der Funktion erhaltene Ähnlichkeitsmaßzahl ist immer positiv, gleichgültig welche Entitätsmengen verglichen werden. Weiters ist die Maßzahl beim Vergleich einer Menge mit sich selbst immer größer oder gleich der Maßzahl, die aus einem Vergleich zwischen zwei beliebigen Entitätsmengen resultiert. Außerdem gilt für Ähnlichkeitsfunktionen, dass sie symmetrische Funktionen sind. Ergibt sich der Ähnlichkeitswert 1, so bedeutet dies, dass die beiden Entitätsmengen, bezogen auf die entsprechenden Ähnlichkeitsfunktion, als identisch angesehen werden. Bei einem Wert kleiner als 1 sind sie zu einem gewissen Grad ähnlich bzw. unterschiedlich und bei einem Wert von 0 gelten die Entitätsmengen als ganz unterschiedlich bzw. haben keine Gemeinsamkeiten. [Ehrig, 2006]

Für die Unterscheidung von verschiedenen Ähnlichkeitsfunktionen wird eine tiefgestellte Indexvariable verwendet: $sim_k(e, f, O_1, O_2)$. Auch hier kann die Angabe der Ontologien entfallen, wenn die betroffenen Ontologien auf Grund des Kontextes klar sind. [Ehrig, 2006]

5.4.3 Schichtenmodell für Ähnlichkeitsfunktionen

In Kapitel 5.2 wurden bereits die Arten von Heterogenitäten auf der Ontologieebene erläutert. Für die Berechnung der Ähnlichkeiten bedeutet dies somit, dass es nicht ausreicht, lediglich lexikalische Vergleiche zwischen Elementnamen oder Tagnamen aus den Ontologien durchzuführen. Es ist auch erforderlich, dass die Struktur bzw. die Relationen der Konzepte sowie deren Bedeutung und der Anwendungsbereich der Ontologien in Betracht gezogen werden. [Hughes und Ashpole, 2005] In [Ehrig u. a., 2005a] wurde dazu ein Framework definiert, welches Ähnlichkeitsfunktionen in Schichten strukturiert. Es besteht aus den horizontal orientierten Schichten *Daten*, *Ontolgie* und *Kontext*, die aufeinander aufbauen. Zu diesen drei Schichten liegt vertikal jene Schicht, die das *Domänenwissen* repräsentiert. Innerhalb der Ontologieschicht wurde in [Ehrig, 2006] die semantische Komplexität noch weiter in Unterschichten, die von der Semantic Web Architektur aus [Berners-Lee, 2000] abgeleitet wurden, aufgesplittet (siehe Abbildung 5.3).

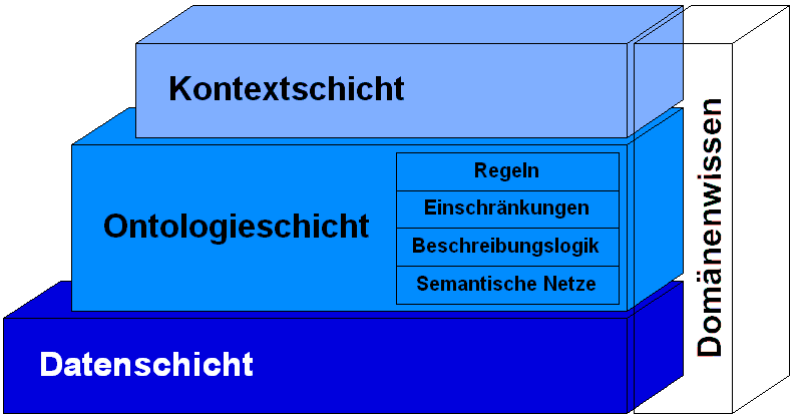

Abbildung 5.3: Schichtenmodell für Ähnlichkeitsfunktionen [Quelle: Ehrig, 2006]

Datenschicht: Diese Schicht beinhaltet den Vergleich von Entitäten durch das reine Betrachten der Datenwerte. Die Datenwerte können dabei einen einfachen (Zeichenketten oder Zahlen) oder komplexen Datentyp haben. Für den Vergleich der Datenwerte werden generische Ähnlichkeitsfunktionen wie etwa die Edit-Distanz[1] für Zeichenketten oder die relative Distanz für Zahlen verwendet. Für komplexe Datentypen, die sich aus einfachen Datentypen zusammensetzen, sind natürlich

[1] auch Levenshtein-Distanz oder Editierdistanz genannt

umfangreichere Funktionen notwendig, die aber oft auf Ähnlichkeitsfunktionen für einfache Datentypen aufbauen. [Ehrig u. a., 2005a]

Ontologieschicht: In der zweiten Schicht werden semantische Relationen zwischen Entitäten untersucht. *Semantische Netze* betrachten die Ontologie als Graph aus Konzepten und Relationen. Zwei Entitäten werden dabei als gleich angesehen, wenn z.B. beide dieselben Eigenschaften haben. Mit Hilfe der *Beschreibungslogik* kann eine Hierarchie von Konzepten aufgebaut werden, wo Subkonzepte die Relationen der Superkonzepte erben. Eine starke Ähnlichkeit von zwei Konzepten kann festgestellt werden, wenn z.B. beide die gleichen Superkonzepte aufweisen. [Ehrig und Sure, 2004; Ehrig u. a., 2005a] Listing 5.1 zeigt ein solches Beispiel.

```
<owl:Class rdf:ID="automobile"/>
<owl:Class rdf:ID="car"/>
<owl:Class rdf:ID="porsche">
  <rdfs:subClassOf rdf:resource="#automobile"/>
  <rdfs:subClassOf rdf:resource="#car"/>
</owl:Class>
<owl:Class rdf:ID="mercedes">
  <rdfs:subClassOf rdf:resource="#automobile"/>
  <rdfs:subClassOf rdf:resource="#car"/>
</owl:Class>
```

Listing 5.1: Zwei Klassen mit gleichen Subkonzepten [Quelle: Ehrig und Sure, 2004]

Einschränkungen ermöglichen eine weitere Art und Weise wie Ähnlichkeiten abgeleitet werden können. Dies kann erfolgen durch die Festlegung von Charakteristika für Eigenschaften (`owl:TransitiveProperty`, `owl:SymmetricProperty`, ...), Werteinschränkungen (`owl:allValuesFrom`, `owl:minCardinality`, ...), Äquivalenzen (`owl:equivalentClass`, ...), Enumerationen (`owl:oneOf`) und Disjunktheit (`owl:disjointWith`). Auch die oberste Subschicht *Regeln* kann für Ontology Similarity interessant sein. Vor allem wenn die gleichen Regeln für Entitäten definiert sind, werden diese Entitäten als ähnlich betrachtet. Jedoch wurde bis jetzt diese Subschicht generell im Semantic Web Bereich noch nicht ausreichend erforscht bzw. fehlt noch die praktische Unterstützung im Sinne von Standards, Frameworks usw. Dies gilt somit bei dieser Subschicht auch für Ontology Similarity. [Ehrig und Sure, 2004; Ehrig u. a., 2005a]

Kontextschicht: Diese Schicht betrachtet die Verwendung der Entitäten der Ontologien in bestimmten Kontexten einer Applikation. Die Ähnlichkeit zwischen Enti-

täten lässt sich hier feststellen, indem dessen Verwendung innerhalb einer ontologiebasierten Anwendung verglichen wird. Als Beispiel sei hier das Verkaufsportal amazon.com genannt. Mit Hilfe von Informationen zum Bücherkauf der Kunden kann die Ähnlichkeit von Büchern in unterschiedlichen Kontexten ermittelt werden. Das Hauptproblem dabei ist natürlich wie bestimmte Verwendungsmuster der verschiedenen Entitäten für die Ähnlichkeitsberechnung definiert bzw. gefunden werden. [Ehrig u. a., 2005a]

Domänenwissen: Eine Ontologie beinhalten oft domänenspezifisches Vokabular. Sofern dieses Vokabular bzw. Wissen klar definiert ist, kann es zusätzlich in allen bisher beschriebenen Schichten - daher auch die vertikale Orientierung in Abbildung 5.3 - eingesetzt werden. [Ehrig und Sure, 2004] Ein domänenspezifisches Wissen wird z.B. durch die Dublin Core Metadata Initiative[2] (DCMI) in Form von internationalen Standards für die Beschreibung von Ressourcen wie Dokumente abgebildet. [DCMI, 2009]

5.4.4 Ähnlichkeitsfunktionen

Dieses Unterkapitel bietet einen Auszug aus konkreten Ähnlichkeitsfunktionen, die beim Ontology Alignment zum Einsatz kommen können. Die einzelnen Funktionen werden dabei nach dem Schichtenmodell aus dem Kapitel 5.4.3 gegliedert. Die Auswahl der Ähnlichkeitsfunktionen basiert auf [Ehrig, 2006]. Für eine umfangreichere Sammlung an Funktionen für Ontology Alignment wird an dieser Stelle auf [Euzenat u. a., 2004] verwiesen.

5.4.4.1 Datenschicht

Wie bereits im Kapitel 5.4.3 beschrieben, erfolgt auf dieser Schicht der Vergleich von Datenwerten. Dies können z.B. Werte zu Datentyp-Eigenschaften von Instanzen oder auch URIs sein. [Ehrig und Sure, 2004]

Gleichheit:
In bestimmten Fällen gelten Datenwerte von Entitäten nur dann als ähnlich, wenn sie

[2] http://dublincore.org/

auch ganz gleich sind. Ein Beispiel dazu wären Datenwerte, die als eindeutige Bezeichner verwendet werden. Folgende Funktion lässt sich daher definieren, wobei v_1 und v_2 Datenwerte sind [Ehrig, 2006]:

$$sim_{equality}(v_1, v_2) := \begin{cases} 1 & \text{if } v_1 = v_2 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Syntaktische Ähnlichkeit für Zeichenfolgen:
Für den Vergleich von Zeichenketten kommt oft die in [Levenshtein, 1966] vorgestellte Edit-Distanz zum Einsatz. Es wird die minimale Anzahl an Einfüge-/Lösch- und Ersetzoperationen ermittelt, die notwendig ist, um die eine Zeichenkette in die andere - unter Verwendung eines dynamischen Algorithmus - zu transformieren. Z.B. die Edit-Distanz für die Zeichenketten „TopHotel“ und „Top Hotel“ ist 1. [Maedche und Staab, 2002]
Basierend auf der Edit-Distanz *ed* kann eine Funktion für die Bestimmung der syntaktischen Ähnlichkeit definiert werden, wobei v_1 und v_2 Zeichenfolgen sind [Ehrig, 2006]:

$$sim_{syntactic}(v_1, v_2) := max\left(0,\ \frac{min(|v_1|, |v_2|) - ed(v_1, v_2)}{min(|v_1|, |v_2|)}\right) \tag{5.2}$$

In der Formel wird zunächst die Zeichenanzahl der Zeichenketten betrachtet und die kleinere Zahl davon herangezogen, um die Edit-Distanz der beiden Wörter davon zu subtrahieren. Zur Normalisierung wird das Ergebnis der Subtraktion durch die kleinere Zeichenanzahl, die bereits vorher ermittelt wurde, dividiert. Ist dieser Wert größer als 0, so entspricht er dem Ergebnis dieser Ähnlichkeitsfunktion, anderfalls ist das Ergebnis 0.

Distanzbasierte Ähnlichkeit für numerische Werte:
Beim Vergleich von numerischen Werten mit bestimmten Wertebereichen (z.B. eine Teilmenge von Ganzzahlen oder Gleitkommazahlen) ist es ratsam, dass die zugrundeliegende Ähnlichkeitsfunktion auf der arithmetischen Differenz der numerischen Werte basiert. Ein generischer Ansatz dafür wäre folgender [Ehrig, 2006]:

$$sim_{diff}(v_1, v_2) := 1 - \left(\frac{|v_1 - v_2|}{maxdiff}\right)^{\gamma} \tag{5.3}$$

Dabei sind $v_1, v_2 \in [min_{v \in V}(V), max_{v \in V}(V)]$ und $maxdiff = max_{v \in V}(V) - min_{v \in V}(V)$ für einen Datentyp V.

Nachdem bei dieser Formel der Betrag aus der Differenz von v_1 und v_2 ermittelt wurde, wird er durch $maxdiff$ dividert. Das erhaltene Zwischenergebnis wird mit dem Parameter $\gamma \in \mathbb{R}^+$ potenziert, wodurch der Einfluss des ansteigenden Unterschieds zwischen den Werten v_1 und v_2 für die Ähnlichkeitsbewertung entweder abgeschwächt oder verstärkt werden kann. Schließlich ergibt sich das Ergebnis dieser Ähnlichkeitsfunktion durch das Subtrahieren des Wertes aus der vorherigen Potenzberechung von 1. [Ehrig, 2006]

5.4.4.2 Objekte

Bis jetzt wurden nur Funktionen für den Vergleich konkreter Datenwerte erläutert. In diesem Unterabschnitt werden nun Ähnlichkeitsfunktionen vorgestellt, die Objekte bzw. Entitäten noch ohne Beachtung der Bedeutung vergleichen. Im Schichtenmodell wird dies schon in der Ontologieschicht angesiedelt. [Ehrig, 2006]

Gleicheit von Objekten:
Basierend auf logischen Aussagen kann die Gleichheit von Objekten festgestellt werden. Logische Aussagen können Definitionen in der Ontologie selbst sein z.B. mit `owl:sameAs` oder aus manuellen bzw. automatischen Methoden hervorgehen. Z.B. durch einen vorangegangenen Alignment-Prozess. [Ehrig u. a., 2006a] In [Ehrig, 2006] wird daher nachstehende Ähnlichkeitsfunktion definiert:

$$sim_{object}(e, f) := \begin{cases} 1 & align(e) = f \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Dice Koeffizient:
Zwei Mengen von Entitäten E und F können, basierend auf deren Mengen an Instanzen ($e \in E,\ f \in F$), verglichen werden. [Castano u. a., 1998] Allerdings ist es erforderlich, dass alle Instanzen eindeutige Bezeichner aufweisen, wobei gleiche Instanzen den gleichen Bezeichner - auch über mehrere Ontologien hinweg - haben müssen [Ehrig, 2006]:

$$sim_{dice}(E, F) := \frac{2 \cdot |E \cap F|}{|E| + |F|} \tag{5.5}$$

Bei dieser Formel wird das Verhältnis zwischen der zweifachen Anzahl an überschneidenden Entitäten der Mengen und der Summe aus der Anzahl an Entitäten der jeweiligen Mengen berechnet.

Jacquard Koeffizient:
Dieser Koeffizient berechnet das Verhältnis zwischen der Anzahl an überschneidenden Entitäten der Mengen und der Anzahl an vorhandenen Entitäten [Ehrig, 2006]:

$$sim_{dice}(E, F) := \frac{|E \cap F|}{|E \cup F|} \tag{5.6}$$

Single Linkage:
Für den Vergleich von zwei Mengen an Entitäten (E, F) wird auch oft der maximale Ähnlichkeitswert zwischen zwei Entitäten aus den zugrundeliegenden Mengen verwendet. Somit wird hier vorausgesetzt, dass die Ähnlichkeitsberechnung zwischen einzelnen Entitäten der Mengen bereits durchgeführt wurde [Ehrig, 2006]:

$$sim_{single}(E, F) := \max_{(e,f)|e \in E, f \in F}(sim(e, f)) \tag{5.7}$$

Dieses Verknüpfungsmaß wird üblicherweise auch bei *Data Mining* z.B. *Clustering* eingesetzt. [Ehrig, 2006]

Average Linkage:
Es ist auch möglich, dass der Durchschnitt über alle Ähnlichkeiten zwischen den Entitäten der beiden Mengen verwendet wird [Ehrig, 2006]:

$$sim_{complete}(E, F) := \frac{\sum_{\forall(e,f)|e \in E, f \in F} sim(e, f)}{|E| \cdot |F|} \tag{5.8}$$

Hier werden zunächst alle Entitäten der ersten Menge mit allen Entitäten der zweiten Menge durch Verwendung einer festgelegten Ähnlichkeitsfunktion verglichen und die daraus resultierenden Ähnlichkeitswerte aufsummiert. Das Endergebnis ergibt sich dann durch die vorhin ermittelte Summe dividiert durch das Produkt aus der Anzahl der ersten Menge und der Anzahl der zweiten Menge.

Multi Similarity:
Bei diesem Ähnlichkeitsmaß wird eine Technik verwendet, die aus der Statistik als Multidimensionale Skalierung bekannt ist. Es wird jede Entität durch einen Vektor repräsentiert, dessen Elemente dem Ähnlichkeitswert zu allen anderen Entitäten der beiden Entitätsmengen (E, F) entsprechen. Für beide Mengen kann ein respräsentativer Vektor

durch Durchschnittsberechnung aus allen Entitätsvektoren gebildet werden. Zu beachten ist dabei, dass durch den Durchschnitt nicht mehr darauf geschlossen werden kann, welche zwei Entitäten der Mengen welchen Ähnlichkeitswert haben. Das Ergebnis der Ähnlichkeitsfunktion ergibt sich schließlich aus dem Skalarprodukt der Vektoren der beiden Mengen, was auch dem Kosinus des Zwischenwinkels der Vektoren entspricht. [Ehrig u. a., 2005a; Ehrig, 2006]

$$sim_{multi}(E,F) := \frac{\sum_{e\in E}\vec{e}}{|\sum_{e\in E}\vec{e}|} \cdot \frac{\sum_{f\in F}\vec{f}}{|\sum_{f\in F}\vec{f}|} \tag{5.9}$$

Dabei ist $\vec{e} = (sim(e,e_1), sim(e,e_2), ..., sim(e,f_1), sim(e,f_2), ...)$. $\vec{f}$ ist analog dazu definiert.

5.4.4.3 Ontologieschicht

Dieser Unterabschnitt behandelt Funktionen, welche die Ähnlichkeit von Entitäten auf Grund der Ontologiestruktur ableiten.

Ähnlichkeit basierend auf Labels:
Den einzelnen Entitäten der Ontologien kann mit dem `rdfs:label`-Tag ein Label bzw. ein Bezeichner zugewiesen werden. Um die Entitäten anhand der Labels zu vergleichen, kann auf die bereits beschriebene Ähnlichkeitsfunktion für Zeichenfolgen zurückgegriffen werden [Ehrig, 2006]:

$$sim_{label}(e,f) := sim_{syntactic}(label(e), label(f)) \tag{5.10}$$

Um den Vergleich von Bezeichnern, die in verschiedenen Sprachen vorliegen, zu unterstützen oder auch Abkürzungen und Synonyme zu finden, wird auch auf Wörterbücher wie z.B. WordNet[3] zurückgegriffen. [Ehrig und Sure, 2004]

Listing 5.2 zeigt ein Beispiel mit Klassen, die ein ähnliches Label aufweisen.

[3] `http://wornet.princeton.edu/`

```
<owl:Class rdf:ID="id1">
  <rdfs:label>telephone number</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="id2">
  <rdfs:label>phone number</rdfs:label>
</owl:Class>
```

Listing 5.2: Zwei Klassen mit ähnlichem Label [Quelle: Ehrig und Sure, 2004]

Ähnlichkeit basierend auf der Hierarchie von Konzepten:
In [Rada u. a., 1989] wurde eine generische Methode erläutert, die die Ermittlung der Ähnlichkeit von Konzepten C einer Ontologie in einer Hierarchie von Konzepten H_C ermittelt [Ehrig, 2006]:

$$sim_{taxonomic}(c_1, c_2) := \begin{cases} e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} & \text{if } c_1 \neq c_2 \\ 1 & \text{otherwise} \end{cases} \tag{5.11}$$

Zu beachten ist, dass hier e nicht einer Entität einer Ontologie, sondern der eulerschen Zahl entspricht. Weiters sind $\alpha \geq 0$ und $\beta \geq 0$ Parameter, mit denen die Verteilung der kürzesten Pfadlänge l und Tiefe h in der Konzepthierarchie skaliert werden kann. Unter dem kürzesten Pfad wird hier die Distanz zwischen den Konzepten c_1 und c_2 verstanden. Die Tiefe der Hierarchieebene wird deswegen in der Berechnung verwendet, da Konzepte, die sich weiter oben in der Hierarchieebene befinden, allgemeiner und semantisch weniger ähnlich sind als jene, die auf einer unteren Ebene angesiedelt sind. Diese Ähnlichkeitsfunktion kann analog für den Vergleich von Relationen basierend auf einer Relationshierarchie verwendet werden. [Ehrig u. a., 2005a]

Ähnlichkeit von Konzepten basierend auf Instanzen:
Die Ähnlichkeit von Konzepten kann nicht nur mit Hilfe ihrer Eigenschaften bestimmt werden, sondern auch durch den Vergleich der Mengen von Konzeptinstanzen. [Ehrig, 2006]
In [Ehrig, 2006] wird dazu nachstehende Ähnlichkeitsfunktion definiert, die die oberhalb beschriebene Multi Similarity verwendet:

$$sim_{extension}(c_1, c_2) := sim_{multi}(\iota_C(c_1), \iota_C(c_2)) \tag{5.12}$$

Dabei entspricht ι_C der Funktionen $C \rightarrow \mathfrak{P}(I)$, die als Konzeptinstantiierung bezeichnet wird.

Nach dieser Formel sind somit zwei Konzepte ähnlich, wenn auch die Instanzen dieser beiden Konzepte nach Anwendung der Multi Similarity ähnlich sind. [Ehrig, 2006]

Ähnlichkeit von Relationen basierend auf Domain und Range:
Folgende Funktion wird in [Ehrig, 2006] für die Ermittlung der Ähnlichkeiten von Relationen $r \in R$ basierend auf deren Domain und Range definiert:

$$sim_{domRan}(r_1, r_2) := 0.5 \cdot (sim_{object}(ran(r_1), ran(r_2)) + sim_{object}(dom(r_1), dom(r_2))) \tag{5.13}$$

Die Ähnlichkeit der Relationen ergibt sich nach dieser Formel aus der Addition des Ähnlichkeitswertes der Range-Konzepte und des Ähnlichkeitswertes der Domain-Konzepte beider Relationen und anschließender Multiplikation mit 0.5. [Ehrig, 2006]

Ähnlichkeit von Instanzen basierend auf gleichen Konzepten:
Die Instanzen $i \in I$ haben eine gewisse Ähnlichkeit, sofern sie Instanzen von gleichen bzw. ähnlichen Konzepten sind [Ehrig, 2006]:

$$sim_{parent}(i_1, i_2) := sim_{object}(c_1, c_2) \; with \; i_1 \in \iota_C(c_1), i_2 \in \iota_C(c_2) \tag{5.14}$$

5.4.4.4 Kontextschicht

Die Bestimmung der Ähnlichkeit von zwei Entitäten unter Berücksichtigung eines bestimmten Kontextes ist in [Ehrig, 2006] wie folgt definiert:

$$sim_{use}(e, f) := sim_{diff}(Usage(e, con), Usage(f, con)) \tag{5.15}$$

Dabei sind $e, f \in E$ und $Usage(e, con)$ entspricht der Verwendungshäufigkeit der Entität e im Kontext *con*. Zu beachten ist, dass die Funktionsdefinition ganz allgemein gehalten ist und bei entsprechenden Anwendungskontexten einfach angepassst bzw. erweitert werden kann.
Diese Formel würde zum Beispiel folgende Ähnlichkeit ermitteln: Zwei Bücher sind ähnlich, wenn deren Autoren in Kooperation zusammen mehrere wissenschaftliche Publikationen verfasst haben. [Ehrig, 2006]

5.5 Anwendungsbereiche

Dieser Abschnitt skizziert drei Anwendungsbereiche aus dem Semantic Web, wo Probleme mit heterogenen Daten auf Grund des Austausches von ontologiebasiertem Wissen bzw. Ressourcen auftreten können und wo Ontology Alignment Methoden nützlich sind. Natürlich ist der Einsatzbereich von Ontology Alignment Methoden nicht nur auf diese drei Bereiche beschränkt.

5.5.1 Kommunikation zwischen Software Agenten

Software Agenten sind autonome Computerprogramme, die bis zu einem gewissen Grad mit anderen Softwareanwendungen bzw. Software Agenten oder auch Menschen kommunizieren können. [Breitman u. a., 2007; Shvaiko und Euzenat, 2005] Für die Kommunikation wird eine Sprache verwendet, mit deren Hilfe die Nachricht zunächst einem bestimmten Interaktionskontext zugeordnet werden kann. Der tatsächliche Inhalt der Nachricht wird in einer Wissensrepräsentationssprache formuliert und referenziert auch oft auf Ontologien. Bei der Kommunikation zwischen unabhängig erstellten Agenten werden diese Agenten nur bedingt die ausgetauschten Nachrichten des jeweils Anderen interpretieren können, wenn die beteiligten Agenten unterschiedliche Sprachen und Ontologien verwenden. Eine Lösung dafür wäre die Verwendung eines Ontology Alignment Protokolls, wie es z.B. in [Euzenat u. a., 2004] beschrieben wird.
Das Protokoll sollte hierbei die Kommunikation mit einer Alignment Service Schicht ermöglichen, um

- eine Alignment-Bibliothek zu durchsuchen,
- ein (Teil-)Alignment basierend auf Ontologien anzufragen,
- die Übersetzung einer Nachricht bei vorhandenem Alignment anzufragen,
- ein Übersetzungsprogramm bei vorhandenem Alignment anzufragen,
- die Vervollständigung eines Teil-Alignments anzufragen und
- ausgehend von einer Nachricht basierend auf einer privaten Ontologie für eine Übersetzung in eine öffentliche Ontologie anzufragen.

Ein Dialog zwischen Software Agenten unter Verwendung einer Alignment Services Schicht mit dem Protokoll könnte wie folgt aussehen [Euzenat u. a., 2004]:

Agent A sendet eine mit der Ontologie *o* formulierte Nachricht *m*.
Agent B fragt nach einer Übersetzung von *m* in Bezug auf die öffentliche Ontologie *o'*.
Agent A sendet die Nachricht *m'*, die nun in Bezug auf *o'* formuliert wurde.
Agent B sendet an das **Alignment Service (AS)** eine Anforderung, dass zwischen Ontologie *o'* und *o"* ein Alignment in Bezug auf die Nachricht *m* erfolgen soll.
AS antwortet mit dem Teil-Alignment *a*.
Agent B fragt beim **AS** um die Übersetzung von *m'* mit Hilfe von *a* an.
AS antwortet mit *m"*.
Agent B fragt beim **AS** um eine Vervollständigung des Teil-Alignments *a* mit den Begriffen in Nachricht *n* an.
AS antwortet mit Alignment *a'*.
Agent B fragt beim **AS** um ein Übersetzungsprogramm, welches von Alignment *a'* abgeleitet ist, an.
AS gibt das Programm *p* als Antwort zurück.
Agent B wendet *p* auf *n* an und sendet das Resultat als Antwort an **Agent A**.

5.5.2 Integration von Web Services

Web Services sind Software Anwendungen, die über ein Interface, welches über das Web zugänglich ist, von BenutzerInnen aufgerufen werden können. Semantic Web Services ermöglichen im Gegensatz zu Standard Web Services durch die Verwendung von Wissensrepräsentationsprachen bzw. Ontologien eine mächtigere und genauere Beschreibung der Services. Es kann nicht davon ausgegangen werden, dass für die Beschreibung verschiedener Semantic Web Services die gleiche Ontologie verwendet wird. Für die Erfüllung einer bestimmten Anforderung ist es daher notwendig, dass für das Auffinden sowie auch für die Integration mehrerer Web Services die korrespondierenden Entitäten zwischen den Ontologien der Web Service Beschreibungen ermittelt werden. Liefert z.B. ein Service die Beschreibung seiner Ausgabe in Form einer Ontologie und das nachfolgend zu verwendende Web Service die Beschreibung der Eingabe in einer anderen Ontologie, so muss hier ein Abgleich der beiden Ontologien erfolgen, um

- die gelieferte Ausgabe mit der geforderten Eingabe des nächsten Services abzugleichen,
- die Vorbedingungen des zweiten Web Services zu überprüfen und

- eine Zwischenschicht zu generieren, die die Ausgabe des ersten Service in das richtige Eingabeformat des zweiten Service transformiert.

[Shvaiko und Euzenat, 2005]

5.5.3 e-Commerce

Viele e-Commerce Anwendungen basieren auf der Veröffentlichung von elektronischen Produktkatalogen. Die Kataloge sind baumstrukturiert aufgebaut und enthalten die zum Verkauf stehenden Produkte mit deren Eigenschaften. Ein Beispiel für solche Anwendungen wären so genannte e-Marketplaces, wo verschiedenste Anbieter ihre Produkte zum Verkauf anbieten können. Ein Problem dabei ist, dass jeder Verkäufer über seinen eigenen lokalen elektronischen Produktkatalog verfügt, der entsprechend den internen Geschäftsprozessen aufgebaut ist. Auf Grund der einheitlichen Schnittstelle beim e-Marketplace gegenüber den Kunden ist es notwendig, dass die internen anbieterspezifischen Produktkataloge in einen allgemeinen Katalog übersetzt werden. Man muss allerdings bedenken, dass ein Verkäufer bei mehreren e-Marketplaces vertreten sein kann, wo jeweils in einheitliche Kataloge übersetzt werden muss. Es ergibt sich somit ein sehr hoher Aufwand für die Anbieter. An dieser Stelle wäre ein Alignment der internen Kataloge mit den einheitlichen Katalogen sehr nützlich, vor allem wenn der schlussendliche Mapping-Prozess semi- oder noch besser vollautomisch erfolgen soll. [Euzenat u. a., 2004; Shvaiko und Euzenat, 2005]

5.6 Der Alignment Prozess

Der in diesem Kapitel vorgestellte Alignment Prozess beschreibt - ausgehenden von zwei oder mehreren Ontologien, Parametern, Ressourcen und vorhandenen Alignments, die optional angegeben werden können - die Schritte, welche beim Ontology Alignment durchlaufen werden, bis eine Ausgabe in Form eines Alignments vorliegt. Viele Ontology Alignment Algorithmen wie z.B. FOAM [Ehrig u. a., 2006a] gehen nach diesen Schritten vor, wobei allerdings manche Schritte zusammengefügt sind oder die Ablaufreihenfolge variiert. Die Grundelemente bleiben aber gleich. Reasoning-basierte Alignment Algorithmen wie z.B. S-Match [Giunchiglia u. a., 2004] bilden dagegen eine Ausnahme. Sie folgen nur teilweise diesen Schritten. [Ehrig, 2006]
Wie in Abbildung 5.4 ersichtlich, besteht der Alignment Prozess aus den Schritten

Merkmalskonstruktion, *Suchraumbestimmung*, *Ähnlichkeitsberechnung*, *Ähnlichkeitsaufsummierung* und *Interpretation*. Der Schritt *Iteration* zeigt an, dass der Prozess iterativ durchgeführt werden kann bis z.B. die Veränderungen der Ausgabe der aktuellen Iteration gegenüber der Ausgabe der vorangegangenen Iteration unter einem festgelegten Schwellwert liegt.

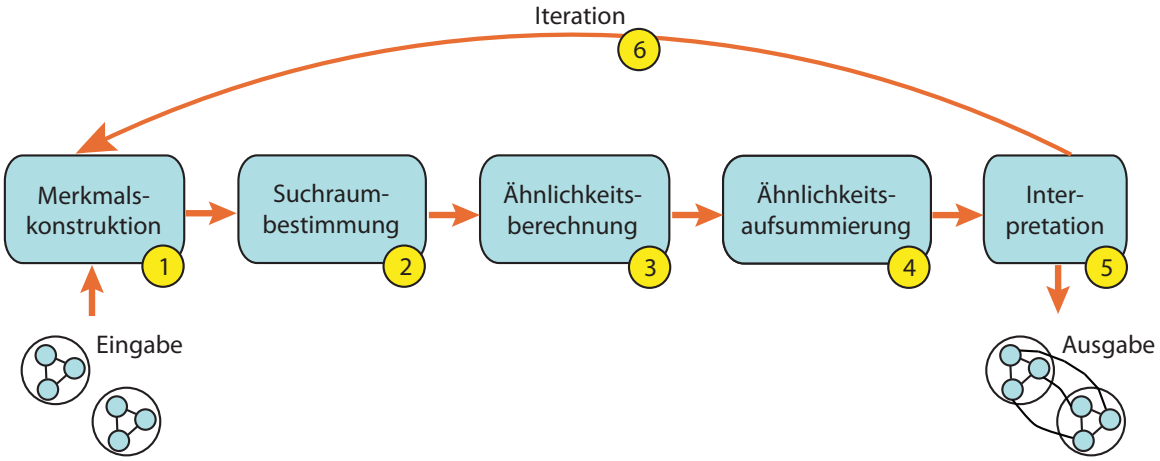

Abbildung 5.4: Der Alignment Prozess [Quelle: Ehrig, 2006]

In den folgenden Unterabschnitten werden die einzelnen Elemente des Alignment Prozesses basierend auf [Ehrig, 2006], [Ehrig u. a., 2006a] bzw. [Ehrig u. a., 2006b] beschrieben.

5.6.1 Merkmalskonstruktion

Im ersten Schritt werden Merkmale der jeweiligen Entitäten ausgewählt, durch welche sie treffend beschrieben werden. Zum einen können diese Merkmale einfach Bezeichner für die Entitäten z.B. in Form von URIs und RDF-Labels sein oder basierend auf Sprachelementen aus RDFS und OWL sein. Z.B. für Subklassen bzw. Subeigenschaften, Gleichheit und Vereinigung. Zum anderen ist es auch erforderlich Merkmale, die nicht direkt in der Ontologie modelliert sind, sondern daraus erst abgeleitet werden müssen, in Betracht zu ziehen. Z.B. sind die Geschwister der Instanz A alle Instanzen vom Konzept B, von der auch A abstammt. Ein weiteres Beispiel dazu wäre folgendes: Aus Ontologie

O_1 ist bekannt, dass ein schnelles Auto einen starken Motor hat und dazu noch zur Klasse der Straßenfahrzeuge gehört. Aus Ontologie O_2 wissen wir, dass man mit einem Sportauto auf der Straße fahren kann und ein Sportauto einen starken Motor hat. Es kann nun darauf geschlossen werden, dass man mit Sportautos schnell fahren kann.
Oft wird Ontology Alignment für bestimmte Anwendungsdomänen durchgeführt. Für das Auffinden der Alignments sollte bei der Merkmalsauswahl auch auf domänenspezifische Aspekte ein Augenmerk gelegt werden. Weiters können externe Merkmale, das sind jegliche Informationen, die nicht in der Ontologie modelliert sind, oder auf Grund schwacher Ausdrucksstärke aus der Ontologie nicht abgeleitet werden können, notwendig sein. Die externen Merkmale können z.B. eine Menge von Wörtern in einem Dokument sein, die eine bestimmte Entität in einer Ontologie beschreiben. Mit Hilfe der Wörter in dieser Beschreibung kann die korrespondierende Entität in der anderen Ontologie gefunden werden.

5.6.2 Suchraumbestimmung

Bei der Suchraumbestimmung wird ermittelt, welche Entitätspaare aus den zugrundeliegenden Ontologien schlussendlich verglichen werden. Übliche Ansätze sind dabei entweder Paare aus allen Entitäten der ersten Ontologie mit denen der zweiten Ontologie zu verbinden oder Paare aus Entitäten mit gleichem Typ (Konzept, Relation oder Instanz) aus der ersten und zweiten Ontologie zu bilden.

5.6.3 Ähnlichkeitsberechnung

Im dritten Schritt kommen die in Kapitel 5.4.4 vorgestellten Ähnlichkeitsfunktionen zum Einsatz. Es wird die Ähnlichkeit pro Merkmal bzw. Merkmalgruppe der Entitäten für alle fest gelegten Entitätspaare berechnet. Die Ähnlichkeitsfunktionen werden entsprechend den Merkmalen gewählt. In Bezug auf das Beispiel in Kapitel 5.4.1 könnte z.B. für das Entitätspaar $(O_1 : car, O_2 : automobile)$ folgender Auszug an Ähnlichkeitsberechnungen durchgeführt werden [Ehrig, 2006]:

$sim_{label}(O_1 : car, O_2 : automobile) = sim_{syntactic}(\text{„car“}, \text{„automobile“}) = 0.0$
$sim_{superconcept}(O_1 : car, O_2 : automobile) =$
$sim_{multi}(\{O_1 : vehicle\}, \{O_2 : vehicle\}) = 1.0$
$sim_{relation}(O_1 : car, O_2 : automobile) =$
$sim_{multi}(\{O_1 : hasSpeed\}, \{O_2 : hasProperty, O_2 : hasMotor\}) = 0.5^2$

5.6.4 Ähnlichkeitsaufsummierung

Bei diesem Schritt werden die vorhin pro Merkmal ermittelten Ähnlichkeitswerte jeweils für die Entitätspaare aggregiert, wobei es manchmal auch vorkommen kann, dass auf Grund hoher Korrelation zwischen den Ähnlichkeitsfunktionen nicht alle Werte aggregiert werden. Allgemein kann die Ähnlichkeitsaufsummierung mit folgender Formel ausgedrückt werden:

$$sim_{agg}(e, f) = agg(sim_1(e, f), ...sim_k(e, f)) \tag{5.16}$$

(e, f) entspricht einem Candidate Alignment und agg einer Funktion, die die resultierenden Werte der Ähnlichkeitsfunktionen (sim_1 bis sim_k) aggregiert. D.h. es erfolgt eine Abbildung $[0, 1]^k$ auf $[0, 1]$. Ein Beispiel für eine gewichtete Aggregierung zeigt nachfolgende Formel:

$$sim_{agg}(e, f) = \frac{\sum_{k=1..n} w_k \cdot adj_k(sim_k(e, f))}{\sum_{k=1...n} w_k} \tag{5.17}$$

w_k stellt dabei die Gewichtung des Ähnlichkeitsmaßes und adj_k stellt eine Funktion dar, die den berechneten Ähnlichkeitswert transformiert bzw. anpasst ($adj : [0, 1] \rightarrow [0, 1]$). Die Aufsummierung der Ähnlichkeitswerte unter Verwendung der soeben vorgestellten Gleichung kann z.B. mit einer der folgenden Varianten erfolgen:

- *Durchschnittsberechnung:* $w_k = 1$, $adj_k(x)$ entspricht der Identitätsfunktion $id(x)$, d.h. der Ähnlichkeitswert bleibt unverändert.
- *Lineare Summierung:* $adj_k(x) = id(x)$ und die Gewichte w_k werden manuell oder durch Lernvorgänge bestimmt.

- *Lineare Summierung mit negativen Gewichten:* Es können auch negative Gewichte verwendet werden, um anzuzeigen, dass bestimmte Ähnlichkeitsmaße im aktuellen Fall kein Indiz für die Ähnlichkeit der Enitäten sind.

- *Summierung unter Verwendung der Sigmoid Funktion:* Eine ausgefeiltere Methode für die Aufsummierung der Ähnlichkeitswerte ist mit der Sigmoid Funktion *sig* (siehe Abbildung 5.5) möglich. Diese Funktion ist durch $sig_k(x) = \frac{1}{1+e^{a_k \cdot x}}$ definiert, wobei mit dem Parameter a_k die Steigung der Funktion beeinflusst werden kann. Es werden somit hier nicht nur einfach die Ähnlichkeitswerte gewichtet, sondern eine Funktion auf die Werte angewandt. Die Sigmoid Funktion wird für die Transformation der Ähnlichkeitswerte verwendet, wodurch in diesem Fall die Definition $adj_k(x) = sig_k(x - 0.5)$ gilt.
  Die grundlegende Idee ist hier, dass hohe Ähnlichkeitswerte verstärkt und niedrige geschwächt werden. Z.B. ist die Wahrscheinlichkeit sehr hoch, dass es sich um die gleichen Entitäten handelt, wenn sich die entsprechenden Bezeichnungen nur durch ein oder zwei Zeichen unterscheiden. Diese Unterschiede können sich durch Tippfehler oder Variation der grammatikalischen Formen ergeben. Stimmen hingegen nur sehr wenige Zeichen der beiden Bezeichnungen überein, so ist es ein Anzeichen dafür, dass diese Entitäten nicht ähnlich sind. Die Aufsummierung dieser niedrigen Ähnlichkeitswerte, die sich im zweiten Fall ergeben würden, sollten nicht zu einem Alignment führen. Aus diesem Grund werden die Ähnlichkeitswerte aus dem ersten Fall verstärkt und aus dem zweiten Fall abgeschwächt.

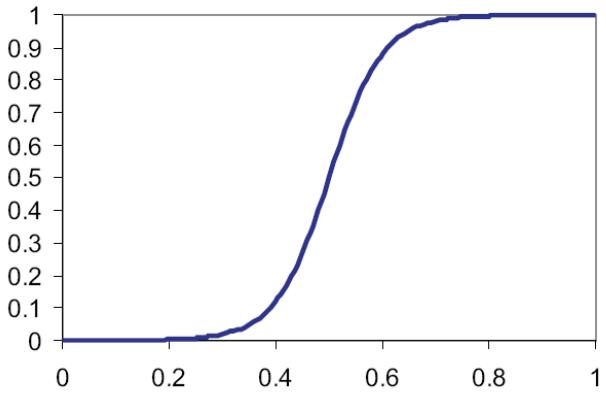

Abbildung 5.5: Sigmoid Funktion [Quelle: Ehrig, 2006]

Nachstehendes Beispiel aus [Ehrig, 2006] zeigt die Verwendung der Durchschnittsbe-

rechnung mit dem Bezug auf das vorgestellte Ontologiebeispiel in Kapitel 5.4.1. Es wird dabei angenommen, dass für die Ähnlichkeitsberechnung der Konzepte $O_1 : car$ und $O_2 : automobile$ 10 Ähnlichkeitsfunktionen herangezogen wurden. Daher wird auch die Summe aus den resultierenden Ähnlichkeitswerten, die durch die Gewichte $w_k = 1$ unverändert bleiben, durch 10 dividiert.

$$\begin{aligned} sim_{agg}(O_1 : car, O_2 : automobile) = (& \\ & 1.0 \cdot sim_{label}(O_1 : car, O_2 : automobile) \\ & +1.0 \cdot sim_{superconcept}(O_1 : car, O_2 : automobile) \\ & +1.0 \cdot sim_{relation}(O_1 : car, O_2 : automobile) \\ & +1.0 \cdot sim_{instance}(O_1 : car, O_2 : automobile) + ... \\ &)/10 = 0.5 \end{aligned}$$

5.6.5 Interpretation

Die aggregierten Ähnlichkeitswerte werden nun mit einem Schwellwert θ verglichen. Liegt der Wert über θ, so wurde eine Abbildung der Entität(en) aus der einen Ontologie auf die Entität(en) der anderen Ontologie gefunden. Diese gefundene Abbildung wird mit den dazugehörigen Entitäten aus den beiden Ontologien mit dem aggregierten Ähnlichkeitswert zur Alignment-Tabelle hinzugefügt. Die Entitätspaare, deren aggregierter Ähnlichkeitswert unter θ liegt, werden innerhalb der aktuellen Iteration nicht weiter betrachtet.
Für die Bestimmung des Schwellwertes sind verschiedene Methoden möglich:

- *Konstanter Wert:* Ein konstanter Wert wird für θ gewählt.

$$\theta = const.$$

- *Delta Method:* Der Schwellwert ergibt sich aus der Differenz des größten aggregierten Ähnlichkeitswertes und eines fixen Wertes.

$$\theta = \max_{e \in O_1, f \in O_2} (sim_{agg}(e, f)) - const.$$

- *N Percent:* Bei dieser Methode wird vom größten aggregierten Ähnlichkeitswert ein bestimmter Prozentsatz abgezogen.

$$\theta = \max_{e \in O_1, f \in O_2} (sim_{agg}(e, f))(1 - p)$$

Es wird in weiterer Folge entschieden, in wie vielen gefundenen Alignments eine Entität enthalten sein darf bzw. welche Alignments - bei mehrfachen Vorkommen von Alignments mit einer bestimmten Entität - in der Ergebnis-Alignment-Tabelle übrig bleiben:

- *One Alignment Link:* Das Ziel dabei ist, nur jene Alignments in der Alignment-Tabelle zwischen Entitätspaaren zu behalten, welche die höchsten Ähnlichkeitswerte aufweisen. Wenn es mehrere Alignments mit einer bestimmten Entität gibt, wird hier also davon ausgegangen, dass es nur ein korrektes Alignment gibt, nämlich jenes mit dem höchsten Ähnlichkeitswert. Alle anderen Alignments werden als potentielle Fehler angesehen.
  Dazu wird die Alignment-Tabelle zunächst absteigend nach dem aggregierten Ähnlichkeitswert sortiert. Die Einträge werden nun von oben nach unten durchgegangen. Ein Eintrag wird aus der Alignment-Tabelle entfernt, wenn er eine Entität enthält, die bereits bei einem vorangegangenen Eintrag mit einem größeren aggregierten Ähnlichkeitswert vorgekommen ist.
- *Multiple Alignment Link:* Oft kann es sinnvoll sein mehrere Alignments, die alle die gleiche Entität enthalten, in der Alignment-Tabelle zu behalten und nicht zu entfernen. Z.B. dann wenn ein bzw. eine BenutzerIn die Alignments manuell überprüft.

5.6.6 Iteration

Da die Ähnlichkeiten von Entitätspaaren durch die Ähnlichkeiten von benachbarten Entitäten in der Ontologiestruktur beeinflusst werden, durchlaufen vielen Ontology Alignment Algorithmen die Schritte *Merkmalskonstruktion*, *Suchraumbestimmung*, *Ähnlichkeitsberechnung*, *Ähnlichkeitsaufsummierung* und *Interpretation* mehrmals. Im ersten Durchlauf werden nur grundlegende Vergleichsmethoden wie z.B. der Vergleich von Zeichenfolgen angewandt oder eine Menge an vordefinierter Alignments verwendet. Bei den weiteren Iterationen können dann strukturbezogene Ähnlichkeitsmaße herangezogen werden. Die Anzahl der Iterationen ist oft vom entsprechenden Anwendungsszenario abhängig und kann durch

- eine fixe Anzahl an Iterationen,
- eine fixe Zeitbeschränkung oder
- den Vergleich der Anzahl an gefundenen Alignments gegenüber der vorherigen Iteration mit einem Schwellwert

ermittelt werden.

Nach mehreren Iterationen könnte - unter der Verwendung der Ontologien aus Kapitel 5.4.1 - ein mögliches Ergebnis so wie in Abbildung 5.6 aussehen. Die gefundenen Alignments sind dabei durch graue Linien dargestellt. Die dazugehörige Alignment Tabelle mit aggregierten Ähnlichkeitswerten zeigt Tabelle 5.1. Bei einem konstanten Schwellwert von 0.5 ist die Ähnlichkeit zwischen *Motor* und *Owner* zu niedrig, womit dieses Entitätspaar nicht zur Alignmentmenge hinzgefügt wird.

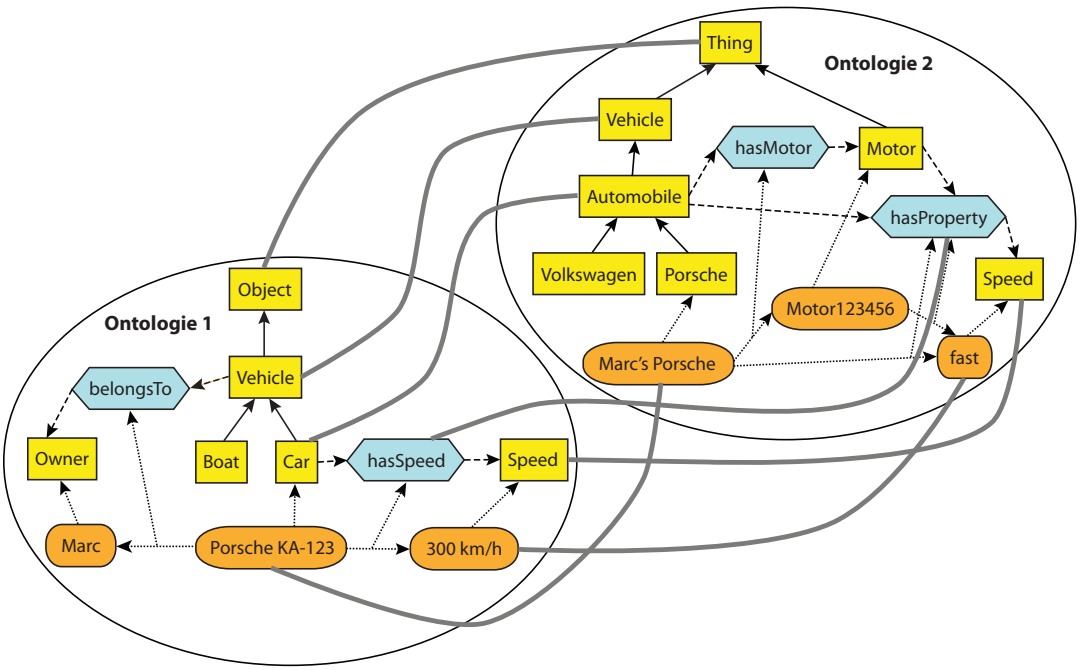

Abbildung 5.6: Die Ontologie aus Kapitel 5.4.1 nach dem Ontology Alignment [Quelle: Ehrig, 2006]

5.7 Evaluierung von Ontology Alignment Algorithmen

Es gibt bereits einige Ontology Alignment Algorithmen und Frameworks wie z.B. ASMOV [Jean-Mary und Kabuka], RiMOM [Li u. a., 2007], Lily [Wang und Xu, 2008], AROMA [David u. a., 2007] und FOAM [Ehrig u. a., 2006a].

Ontologie O_1	Ontologie O_2	Ähnlichkeitswert	Alignment
Object	Thing	0.95	ja
Vehicle	Vehicle	0.9	ja
Car	Automobile	0.85	ja
Speed	Speed	0.8	ja
hasSpeed	hasProperty	0.75	ja
Porsche KA-123	Marc's Porsche	0.75	ja
300 km/h	fast	0.6	ja
Motor	Owner	0.3	nein

Tabelle 5.1: Alignment Tabelle zu Abbildung 5.6 [Quelle: Ehrig, 2006]

Um die Qualität der Ergebnisse dieser Algorithmen festzustellen, wird üblicherweise eine Menge mit Referenz-Alignments (R) basierend auf den Ontologien durch Experten erstellt und mit den Ergebnissen (A) der Ontology Alignment Algorithmen verglichen. Die gängigsten Maße dafür, welche auch im Information Retrieval Bereich zum Einsatz kommen, sind *Precision* und *Recall*. [Euzenat u. a., 2005a] Diese beiden Maße sind in [Euzenat u. a., 2005a] wie folgt definiert.

Definition 3 (Precision)**.** *Bei einer Menge von gegebenen Referenz-Alignments R ist die Precision einer Menge von gefundenen Alignments A definiert durch*

$$P(A, R) = \frac{|R \cap A|}{|A|}$$

Definition 4 (Recall)**.** *Bei einer Menge von gegebenen Referenz-Alignments R ist der Recall einer Menge von gefundenen Alignments A definiert durch*

$$R(A, R) = \frac{|R \cap A|}{|R|}$$

Precision berechnet also das Verhältnis zwischen der Anzahl der gefundenen Alignments, die auch in den Referenz-Alignments enthalten sind (in der Literatur wird oft von *true positives* gesprochen), und der Anzahl der gefundenen Alignments. Recall berechnet das Verhältnis zwischen der Anzahl der true positives und der Anzahl der Referenz-Alignments. [Do u. a., 2002; Euzenat u. a., 2005a; van Hage, 2008]

Ein weiteres Maß, welches als *Fallout* bezeichnet wird, liefert den Anteil an gefundenen Alignments, die allerdings nicht in den Referenz-Alignments enthalten sind (in der Literatur wird oft von *false positives* gesprochen). [Do u. a., 2002; Euzenat u. a., 2005a] Wie

in der nachstehenden Definition aus [Euzenat u. a., 2005a] ersichtlich, kann die Berechnung auf zwei Arten erfolgen. Entweder man subtrahiert von der Elementanzahl von A die Anzahl jener Alignments, die sowohl in A als auch R enthalten sind, und dividiert dann durch die Elementanzahl von A oder man ermittelt einfach direkt die Anzahl der Alignments aus A, die allerdings nicht in R enthalten sind und dividiert dann ebenfalls durch die Elementanzahl von A.

Definition 5 (Fallout)**.** *Bei einer Menge von gegebenen Referenz-Alignments R ist der Fallout einer Menge von gefundenen Alignments A definiert durch*

$$F(A, R) = \frac{|A| - |A \cap R|}{|A|} = \frac{|A \setminus R|}{|A|}$$

Oft ist es erwünscht, dass beim Vergleich von Alignment Algorithmen nur eine Maßzahl verwendet wird. Aus diesem Grund wird manchmal das F-Maß verwendet, das die Ergebnisse aus Precision und Recall zusammenfügt. [Euzenat u. a., 2005a] Folgende Definition ist dazu in [Euzenat u. a., 2005a] angeführt.

Definition 6 (F-Maß)**.** *Bei einer Menge von gegebenen Referenz-Alignments R und einer Zahl α zwischen 0 und 1 ist das F-Maß einer Menge von gefundenen Alignments A definiert durch*

$$M_\alpha(A, R) = \frac{P(A, R) \cdot R(A, R)}{(1 - \alpha) \cdot P(A, R) + \alpha \cdot R(A, R)}$$

Wenn α mit 1 belegt wird, ist der Wert für das F-Maß gleich dem Wert für Precision, wenn für α 0 gewählt wird, so ist der Wert für das F-Maß gleich dem Wert für Recall. Sehr oft wird für α der Wert 0.5 verwendet, wodurch sich die Formel in der Definition oberhalb wie folgt umschreiben lässt [Euzenat u. a., 2005a]:

$$M_{0.5}(A, R) = \frac{2 \cdot P(A, R) \cdot R(A, R)}{P(A, R) + R(A, R)}$$

Im Bereich von Ontology Alignment hat sich die Ontology Alignment Evaluation Initiative (OAEI) gebildet, die seit 2004 jährlich Evaluierungen von eingereichten Ontology Alignment Algorithmen durchführt. Die Evaluierungsszenarien werden dabei kontinuierlich verfeinert. [Euzenat u. a., 2005b; Caracciolo u. a., 2008] Für eine detaillierte Beschreibung der einzelnen Evaluierungsmethoden und -szenarien sei an dieser Stelle auf deren

	FOAM		
	Prec.	Rec.	F-Maß
2005	0.9	0.69	0.78
2004	0.74	0.59	0.66

Tabelle 5.2: Ergebnisse für FOAM bei der Ontology Alignment Evaluierung 2004 und 2005 durch OAEI [Quelle: Euzenat u. a., 2005b]

Homepage[4] verwiesen, wo auch die Ergebnisse der Evaluierung zu finden sind. Die Tabellen 5.3 und 5.2 zeigen die Werte für Precision, Recall und das F-Maß aus Evaluierungen durch die OAEI für *ASMOV*, *RiMOM*, *Lily* und *AROMA* aus den Jahren 2007 und 2008 sowie für *FOAM* aus den Jahren 2004 und 2005. Aus den Tabellen ist ersichtlich, dass bei allen Alignment Algorithmen die Precisionwerte immer höher sind als die Recallwerte. D.h. der Anteil an richtigen Alignments in A ist höher, als das Verhältnis zwischen der Anzahl der richtigen Alignments aus A und der Anzahl der Referenz-Alignments aus R. Außerdem zeigen die Werte in den Tabellen, dass alle Algorithmen bis auf *AROMA* die Qualität von einem Jahr auf das andere durch Weiterentwicklungen verbessern oder zumindest beibehalten konnten. Weiters kann beim Vergleich der Werte aus den Jahren 2004/05 mit den Werten aus den Jahren 2007/08 eine allgemeine Qualitätsverbesserung bei den Alignment Algorithmen abgeleitet werden. Betrachtet man für das Jahr 2008 die Werte für das F-Maß, so konnte der Alignment Algorithmus mit dem Namen *Lily* die besten Ergebnisse für die Evaluierungsszenarien der OAEI liefern.

Nachdem in diesem Kapitel die allgemeine Vorgehensweise beim Ontology Alignment erläutert und bereits einige Ontology Alignment Algorithmen genannt wurden, erfolgt im nächsten Kapitel die Beschreibung der Basiselemente und die konkrete Arbeitsweise des Alignment Algorithmus *FOAM*.

[4] `http://oaei.ontologymatching.org/`

	ASMOV			RiMOM			Lily			AROMA		
	Prec.	Rec.	F-Maß	Prec.	Rec.	F-Maß	Prec.	Rec.	F-Maß	Prec.	Rec.	F-Maß
2008	0.95	0.86	0.90	0.96	0.84	0.90	0.97	0.88	0.92	0.95	0.70	0.81
2007	0.95	0.85	0.90	0.95	0.83	0.89	0.96	0.87	0.91	0.96	0.72	0.82

Tabelle 5.3: Ergebnisse für ASMOV, RiMOM, Lily und AROMA bei der Ontology Alignment Evaluierung 2007 und 2008 durch OAEI [Quelle: Caracciolo u. a., 2008]

Kapitel 6

Framework for Ontology Alignment and Mapping: FOAM

FOAM ist ein Ontology Alignment Algorithmus, der an der Universität Karlsruhe am Institut für Angewandte Informatik und Formale Beschreibungssprachen[1] entwickelt wurde. Die Implementierung erfolgte in Java[2] und steht auf der FOAM-Homepage[3] als Download mit den ausführbaren Dateien, dem Programmcode, der Installationsanleitung, der Dokumentation und den Testontologien frei zur Verfügung. In diesem Kapitel wird basierend auf [Ehrig, 2006] bzw. [Ehrig u. a., 2006a] der grobe Aufbau und die Verwendung von FOAM genauer erläutert, da dieser Ontology Alignment Algorithmus für den praktischen Teil dieser Arbeit von wesentlicher Bedeutung ist.

6.1 Grundlagen

FOAM kann über die Kommandozeile, über eine Java-API als auch über ein Webservice verwendet werden. Für die Verarbeitung von Ontologien, maschinelles Lernen und Wörterbuchsuche setzt FOAM bereits vorhandene Software ein:

- *KAON2*[4] ist ein Framework, welches eine API für die Verarbeitung von OWL-DL, SWRL[5] und F-Logic[6] Ontologien bereitstellt. FOAM selbst unterstützt nur OWL-

[1] http://www.aifb.uni-karlsruhe.de/
[2] http://java.sun.com
[3] http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/
[4] http://kaon2.semanticweb.org/
[5] http://www.w3.org/Submission/SWRL/
[6] http://flora.sourceforge.net/aboutFlogic.php

DL Ontologien. KAON2 ist mit Java 1.5 implementiert, wodurch auch FOAM diese Java-Version voraussetzt.

- *WEKA*[7] ist ein Open Source Framework, das unter GNU General Public License steht. FOAM verwendet die in WEKA enthaltene Sammlung an Algorithmen und Werkzeugen für maschinelles Lernen (siehe Kapitel 6.5).
- *WordNet*[8] ist eine freie Software, die eine lexikalische Datenbank für die englische Sprache enthält. Für Ontology Alignment ist WordNet vor allem für das Finden von Synonymen beim Vergleich der Entitätsbezeichnungen interessant.

Basierend auf den Konfigurationseinstellungen (siehe Kapitel 6.6) können beim Ontology Alignment mit FOAM folgende Methoden zum Einstatz kommen:

- Naive Ontology alignMent (NOM)
- Quick Ontology Mapping (QOM)
- Active Ontology Alignment (AOA)
- Alignment Process Feature Estimation and Learning (APFEL)

In den nachfolgenden Kapiteln werden diese Methoden in Anlehnung am allgemeinen Alignment Prozess (siehe Kapitel 5.6) genauer erklärt.

6.2 Naive Ontology alignMent: NOM

NOM [Ehrig, 2006] stellt die Grundlage von FOAM dar. Die in den darauffolgenden Abschnitten beschriebenen Methoden von FOAM ersetzen bzw. erweitern Teilbereiche von NOM.

Für die Ermittlung der Ähnlichkeiten verwendet NOM URIs, RDF/RDFS- und OWL-Sprachelemente sowie auch davon abgeleitete Informationen. Es wird dabei jede Entität der ersten Ontologie mit jeder Entität der zweiten Ontologie, die auch den gleichen Typ (Konzept, Relation, Instanz) aufweist, verglichen. Beim Beispiel aus Kapitel 5.4.1 ergibt dies 55 Candidate Alignments: 42 (6x7) Konzept-, vier (2x2) Relations- und neun (3x3) Instanzpaare. Tabelle 6.1 listet alle Merkmale und die dazugehörigen Ähnlichkeitsmaße, die beim Schritt *Ähnlichkeitsberechnung* herangezogen werden, auf. Die erste Spalte

[7] `http://www.cs.waikato.ac.nz/~ml/weka/`
[8] `http://wornet.princeton.edu/`

dieser Tabelle gibt den Typ der Entität an. Für den Ähnlichkeitsvergleich werden die gleichen Merkmale (siehe Spalte *Merkmal*) aus beiden Ontologien extrahiert und verglichen. Ausnahmen bilden dabei die Einträge mit Nummer 8 und 9 bei den Konzepten. Dort werden die direkten Subkonzepte der ersten Ontologie mit den direkten Superkonzepten der zweiten Ontologie und umgekehrt gegenübergestellt. Die letzte Spalte definiert das zu den Merkmalen entsprechende Ähnlichkeitsmaß. Es handelt sich dabei um jene Ähnlichkeitsfunktionen aus Kapitel 5.4.4.

Neben einer linearen Summierung unterstützt NOM bei der Ähnlichkeitsaufsummierung auch die Summierung unter Verwendung der Sigmoid Funktion. Die Gewichte, die auch negativ sein können, müssen manuell festgelegt werden. Aus dem dem Autor dieser Arbeit vorliegendem Programmcode von FOAM ist ersichtlich, dass die Gewichte hardcodiert sind und nicht über die Konfiguration von außen gesteuert werden können. Somit muss bei gewünschten Gewichtsänderungen der Programmcode verändert sowie neu kompiliert werden.
Die Interpretation der aggregierten Ähnlichkeitswerte erfolgt danach in zwei Schritten. Zunächst werden Entitätspaare, deren zugehöriger Ähnlichkeitswert unter einem festgelegten Schwellwert liegt, aussortiert. Bei den restlichen Entitätspaaren geht NOM nach der One Alignment Link Strategie vor.
Innerhalb der ersten Iteration des Alignment Prozesses werden nur Zeichenkettenvergleiche für Labels und URIs verwendet. In den nachfolgenden Iterationen werden dann auch strukturbezogene Informationen extrahiert und die entsprechenden Ähnlichkeitswerte basierend auf Tabelle 6.1 berechnet. Die Iterationsanzahl ist bei NOM auf 10 fixiert, da sich durch Tests herausstellte, dass nach 10 Iterationen das Alignment-Ergebnis nahezu unverändert bleibt.

```
1 ...
2 http://www.example.com/ontologies/auto/autoA.owl#Vehicle;
      http://www.example.com/ontologies/auto/autoB.owl#Vehicle;1.0
3 http://www.example.com/ontologies/auto/autoA.owl#300km_h;
      http://www.example.com/ontologies/auto/autoB.owl#fast;0.9821428571428571
4 ...
```

Listing 6.1: Auszug einer Ausgabedatei von NOM

Als Ausgabe liefert NOM jene Entitätspaare, die über dem Schwellwert liegen, mit Strichpunkt vom dazugehörigen aggregierten Ähnlichkeitswert getrennt. Nach der Anwendung der One Alignment Link Strategie, steht des endgültige Ergebnis fest. Listing 6.1 zeigt einen kleinen Auszug aus einer Beispielausgabe. Hier wurden die Konzepte *Vehicle* aus

Vergleich	Nr.	Merkmal	Ähnlichkeitsmaß
Konzepte	1	(label, X_1)	syntactic(X_1, X_2)
	2	(identifier, X_1)	equality(X_1, X_2)
	3	(X_1,sameAs, X_2) relation	object(X_1, X_2)
	4	(direct relations, Y_1)	multi(Y_1, Y_2)
	5	all (inherited relations, Y_1)	multi(Y_1, Y_2)
	6	all (superconcepts, Y_1)	multi(Y_1, Y_2)
	7	all (subconcepts, Y_1)	multi(Y_1, Y_2)
	8	(subconc., Y_1) / (superconc., Y_2)	multi(Y_1, Y_2)
	9	(superconc., Y_1) / (subconc., Y_2)	multi(Y_1, Y_2)
	10	(concept siblings, Y_1)	multi(Y_1, Y_2)
	11	(instances, Y_1)	multi(Y_1, Y_2)
Relationen	1	(label, X_1)	syntactic(X_1, X_2)
	2	(identifier, X_1)	equality(X_1, X_2)
	3	(X_1, sameAs, X_2) relation	object(X_1, X_2)
	4	(domain, X_{d1}) and (range, X_{r1})	object(X_{d1}, X_{d2}), (X_{r1}, X_{r2})
	5	all (superrelations, Y_1)	multi(Y_1, Y_2)
	6	all (subrelations, Y_1)	multi(Y_1, Y_2)
	7	(relation siblings, Y_1)	multi(Y_1, Y_2)
	8	(relation instances, Y_1)	multi(Y_1, Y_2)
Instanzen	1	(label, X_1)	syntactic(X_1, X_2)
	2	(identifier, X_1)	equality(X_1, X_2)
	3	(X_1, sameAs, X_2) relation	object(X_1, X_2)
	4	all (parent concepts, Y_1)	multi(Y_1, Y_2)
	5	(relation instances, Y_1)	multi(Y_1, Y_2)
Relations-instanzen	1	(domain, X_{d1}) and (range, X_{r1})	object(X_{d1}, X_{d2}), (X_{r1}, X_{r2})
	2	(parent relation, Y_1)	multi(Y_1, Y_2)

Tabelle 6.1: Merkmale und Ähnlichkeitsmaße bei NOM [Quelle: Ehrig, 2006]

beiden Ontologien sowie die Instanzen *300 km/h* und *fast* als gleich erkannt, da der aggregierte Wert über dem definierten Schwellwert von 0.9 liegt.

6.3 Quick Ontology Mapping: QOM

Oft ist es neben dem Ermitteln des besten Alignment-Ergebnisses auch notwendig ein Ergebnismenge mit den Alignments in kurzer Zeit bereitzustellen. Der Ausgleich zwischen Effektivität und Effizienz muss hier gefunden werden, d.h. es muss ein Kompromiss zwischen der Anzahl der korrekten Alignments und dem Zeitverbrauch für die Ermittlung der Alignments gefunden werden.
QOM [Ehrig und Staab, 2004] versucht das Alignment-Ergebnis schneller und dabei mit einem nur unwesentlichen Qualitätsverlust gegenüber NOM zu erzielen. Dazu verändert QOM die zeitkritischen Schritte *Suchraumbestimmung* und *Ähnlichkeitsberechnung* von NOM. Die restlichen Schritte sowie die Ausgabe bleiben unverändert.
Bei der Suchraumbestimmung setzt QOM folgende Strategien ein, damit nicht ständig alle Entitäten aus den Ontologien miteinander verglichen werden:

- *Random:* Die einfachste Möglichkeit, die Anzahl der Canditate Alignments einzuschränken ist, dass per Zufall eine bestimmte Anzahl oder ein bestimmter Prozentsatz aller möglichen Kanditaten ausgewählt wird.
- *Label:* Zwei Entitäten aus den zugrundeliegenden Ontologien werden verglichen, wenn die Werte im dazugehörigen `rdfs:label`-Tag oder die Fragement-Teile der URIs ähnlich sind, d.h. wenn die ersten drei Zeichen übereinstimmen.
- *Change Propagation:* Es erfolgt ein Vergleich von Entitäten, wenn für deren benachbarten Entitäten ein Alignment in der letzten Iteration hinzugefügt wurde. Dieser Ansatz geht davon aus, dass zwischen Entitäten, die an Entitäten eines vorhin gefundenen Aligments angrenzen, auch eine Ähnlichkeit besteht. Damit hier auch der Vergleich von sehr vielen Entitäten z.B. bei Ontologien mit nur einer Hierarchieebene vermieden wird, verwendet man dabei eine festgelegte Obergrenze für die Anzahl der Canditate Alignments.

QOM verfolgt in der ersten Iteration die Label-Strategie, daraufhin die Change Propagation-Strategie. Bei den letzten Iterationen liegt der Fokus auf der Random-Strategie.
Bei der Ähnlichkeitsberechnung vermeidet QOM den paarweisen Vergleich von ganzen

Hierarchiebäumen und verwendet stattdessen den Vergleich von Merkmalen direkt angrenzender Entitäten. Die betroffenen Merkmale bzw. Ähnlichkeitsmaße sind in Tabelle 6.2 mit einem kleinen a neben der Nummer markiert bzw. ersichtlich.

6.4 Active Ontology Alignment: AOA

Viele Ontology Alignment Algorithmen ermitteln zunächst das Ergebnis-Alignment vollautomatisch und ermöglichen dem zw. der BenutzerIn danach zu entscheiden, welche Alignments korrekt bzw. nicht korrekt sind und für die Weiterverarbeitung in einer entsprechenden Applikation verwendet werden sollen. Oft resultiert aus dem vollautomatischen Ontology Alignment kein zufriedenstellendes Ergebnis. Aus diesem Grund versucht Active Ontology Alignment (AOA) [Ehrig und Sure, 2005] die Interaktion des Benutzers bzw. der Benutzerin maßgeblich beim Alignment Prozess einzubringen. Dazu bietet sich der Schritt *Interpretation* an. Alle anderen Schritte laufen wie bei NOM bzw. QOM ab.

Alle Entitätspaare, deren ermittelter aggregierter Ähnlichkeitswert über dem festgelegten Schwellwert liegt, werden zum Alignment hinzugefügt. Jene, deren Wert darunter liegt, werden nicht hinzugefügt bzw. verworfen.
Genau bei dieser Entscheidung, ob ein Entitätspaar zum Alignment hinzugefügt wird oder nicht, wird der bzw. die BenutzerIn beim AOA miteinbezogen. Natürlich ist es auf Grund der Anzahl nicht sinnvoll alle Alignments bzw. Non-Alignments dem bzw. der BenutzerIn zur Validierung vorzulegen. Der Schwellwert stellt einen kritischen Wert dar. Für die automatische Klassifizierung kommen bei diesem Wert die größten Zweifel auf, ob man sich nun für oder gegen das Hinzufügen zur Ergebnis-Alignmentmenge entschließt. Es werden daher nur jene Entitätspaare zur Validierung aufgelistet, deren aggregierter Ähnlichkeitswert am nähesten zum Schwellwert liegt. Falls zu viele Einträge dieses Kriterium erfüllen, wird nur eine Teilmenge zur Validierung angezeigt und dabei Alignment-Vorschläge mit Entitäten, die innerhalb der Quellontologie stark verknüpft sind, bevorzugt. Diese Bevorzugung wird dadurch begründet, dass Entitäten mit vielen Verbindungen zu anderen Entitäten mehr Einfluss auf das Finden weiterer Alignments in Folgeiterationen haben, als jene mit geringen Verbindungen. Wird vom bzw. von der BenutzerIn ein Alignment-Vorschlag als richtig klassifiziert, dann wird der Ähnlichkeitswert 1 zugewiesen, andernfalls 0. Die weiteren Iterationen verwenden diese Ähnlichkeitswerte wiederum, um eine neue sowie verbesserte Ergebnis-Alignmentmenge zu generieren.

Vergleich	Nr.	Merkmal	Ähnlichkeitsmaß
Konzepte	1	(label, X_1)	syntactic(X_1, X_2)
	2	(identifier, X_1)	equality(X_1, X_2)
	3	(X_1,sameAs, X_2) relation	object(X_1, X_2)
	4	(direct relations, Y_1)	mulit(Y_1, Y_2)
	5a	(relations of direct superconc., Y_1)	multi(Y_1, Y_2)
	6a	(direct superconcepts, Y_1)	multi(Y_1, Y_2)
	7a	(direct subconcepts, Y_1)	multi(Y_1, Y_2)
	8a	(subconc., Y_1) / (superconc., Y_2)	multi(Y_1, Y_2)
	9a	(superconc., Y_1) / (subconc., Y_2)	multi(Y_1, Y_2)
	10	(concept siblings, Y_1)	multi(Y_1, Y_2)
	11a	(direct instances, Y_1)	multi(Y_1, Y_2)
Relationen	1	(label, X_1)	syntactic(X_1, X_2)
	2	(identifier, X_1)	equality(X_1, X_2)
	3	(X_1, sameAs, X_2) relation	object(X_1, X_2)
	4	(domain, X_{d1}) and (range, X_{r1})	object(X_{d1}, X_{d2}), (X_{r1}, X_{r2})
	5a	(direct superrelations, Y_1)	multi(Y_1, Y_2)
	6a	(direct subrelations, Y_1)	multi(Y_1, Y_2)
	7	(relation siblings, Y_1)	multi(Y_1, Y_2)
	8a	(direct relation instances, Y_1)	multi(Y_1, Y_2)
Instanzen	1	(label, X_1)	syntactic(X_1, X_2)
	2	(identifier, X_1)	equality(X_1, X_2)
	3	(X_1, sameAs, X_2) relation	object(X_1, X_2)
	4a	(direct parent concepts, Y_1)	multi(Y_1, Y_2)
	5	(relation instances, Y_1)	multi(Y_1, Y_2)
Relations-	1	(domain, X_{d1}) and (range, X_{r1})	object(X_{d1}, X_{d2}), (X_{r1}, X_{r2})
instanzen	2	(parent relation, Y_1)	multi(Y_1, Y_2)

Tabelle 6.2: Merkmale und Ähnlichkeitsmaße bei QOM [Quelle: Ehrig, 2006]

Entsprechend der Klassifzierungen des Benutzers bzw. der Benutzerin wird der Schwellwert für die nächste Iteration angepasst. Wenn die Mehrzahl der Alignment-Vorschläge mit richtig bzw. positiv klassifiziert wurde, liegt der Schwellwert zu hoch und wird daher verringert. Im anderen Fall, d.h. wenn es mehr negativ als positiv klassifizierte Alignment-Vorschläge gibt, ist der Schwellwert zu niedrig und wird erhöht. Üblicherweise wird das Verhältnis zwischen den positiven und negativen Validierungen mit einem festgelegten maximalen Änderungswert multipliziert und das Ergebnis zum bzw. vom aktuellen Schwellwert addiert oder substrahiert.

6.5 Alignment Process Feature Estimation and Learning: APFEL

Wie in den vorangegangenen Abschnitten erläutert, verwenden die Methoden NOM, QOM und AOA festgelegte Merkmale und dazugehörige Ähnlichkeitsfunktionen. Allerdings ist es auch für Ontologie- bzw. Wissensrepräsentationsexperten nicht möglich vorauszusagen, mit welcher Strategie bzw. Kombination von Merkmalsvergleichen das beste Alignment-Ergebnis für bestimmte Ontologien effizient ermittelt werden kann. Dies liegt vor allem an der steigenden Komplexität der Ontologiesprachen sowie am Anteil domänenspezifischer Aspekte, die für das Ermitteln einer optimalen Alignmentmenge miteinbezogen werden sollten.

APFEL [Ehrig u. a., 2005b] verwendet Methoden aus dem Bereich des maschinellen Lernens, um den optimalen Einsatz von Strategien und Kombination von Merkmalsvergleichen bei vorgegebenen Ontologien zu finden. Je nach Konfiguration (siehe Kapitel 6.6) kommen entweder Support Vector Machines oder Entscheidungsbäume (*Decision Trees*) für das maschinelle Lernen zum Einsatz. Abbildung 6.1 zeigt den Ablaufprozess bei APFEL.

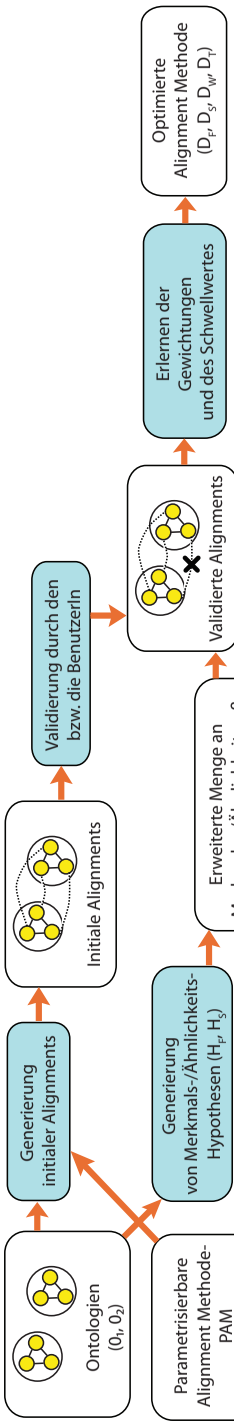

Abbildung 6.1: Ablaufprozess bei APFEL [Quelle: Ehrig, 2006]

Hauptaufgabe von APFEL ist das Erlernen von Parametern, die zur optimalen Alignmentmenge führen. Zunächst sind dafür die Paramter:

- Q_F: Merkmale, die aus den Ontologien für den Ähnlichkeitsvergleich extrahiert werden sollen (Labels, spezielle Strukturinformationen, usw.),
- Q_S: Ähnlichkeitsfunktionen, die zum Vergleich der extrahierten Merkmale herangezogen werden sollen,
- Q_W: Gewichte für die Aggregation der berechneten Ähnlichkeitswerte und
- Q_T: Strategie zur Interpretation der aggregierten Werte z.B. Werte müssen über einem bestimmten Schwellwert liegen,

die von außen gesetzt werden, relevant. Diese Parameter werden im ersten Schritt einer parameterisierbaren Alignment Methode (PAM) übergeben, um initiale Alignments A_I abzuleiten. PAM stellt dabei eine allgemeine Alignment Funktion dar. Zu Beginn kann z.B. $align_{init}$ durch $align_{init} := PAM(QOM)$ definiert sein. Da zunächst nur initiale bzw. vorübergehende Alignments gefunden werden sollen, reichen in den meisten Fällen Ähnlichkeitsvergleiche von Basismerkmalen wie z.B. Zeichenkettenvergleiche der Labels sowie eine Durchschnittsberechnung für die Aggregation als auch ein fixierter Schwellwert aus. Außerdem sollte der Schwellwert eher niedriger angesetzt werden, damit auch falsche Alignments im ersten Ergebnis enthalten sind. Die initialen Alignments werden daraufhin absteigend nach dem aggregierten Ähnlichkeitswert sortiert und dem bzw. der BenutzerIn präsentiert. Er bzw. sie überprüft diese Alignments und bewertet sie mit richtig (1) oder falsch (0). Zusätzlich können auch weitere Alignments, falls diese noch nicht gefunden wurden, vom bzw. von der BenutzerIn hinzugefügt werden. Die Anzahl und Qualität der validierten Alignments hat Auswirkungen auf die Ergebnisqualität der Methoden des maschinellen Lernens.

In weiterer Folge extrahiert APFEL zusätzliche Merkmale H_F, die in beiden Ontologien auftreten. Dies können z.B. Merkmale mit gleichen URIs, bestimmten OWL Sprachelementen oder speziellen XML Datentypen sein. An dieser Stelle kann auch domänenspezifisches Wissen in den Alignment Prozess einfließen. Die gefundenen Merkmale werden jeweils zunächst mit allen vordefinierten Elementen aus der Menge an Ähnlichkeitsfunktionen in Verbindung gesetzt. Bei Ontologien, die aus einer ähnlichen Domäne wie jene Ontologien aus Kapitel 5.4.1 von Seite 48 stammen, könnten dabei z.B. die Merkmale *extras* und *licensenumber* extrahiert werden. Wenn *equality* und *subset* bzw. *substring*[9] in der Menge der vordefinierten Ähnlichkeitsfunktionen enthalten sind, so könnten die

[9]je nach Datentyp der Merkmale wird entweder *subset* oder *substring* verwendet

Vergleich	Nr.	Merkmal H_F	Ähnlichkeitsmaß aus H_S
cars	H1	(extras, X_1)	equality(X_1, X_2)
cars	H2	(extras, X_1)	subset(X_1, X_2)
cars	H3	(licensenumber, X_1)	equality(X_1, X_2)
cars	H4	(licensenumber, X_1)	substring(X_1, X_2)

Tabelle 6.3: Generierte Hypothesen von APFEL [Quelle: Ehrig, 2006]

aufgestellten Hypothesen wie in Tabelle 6.3 aussehen. Durch das maschinelle Lernen werden nicht nützliche Hypothesen erkannt. So kann z.B. festgestellt werden, dass die Hypothese H4 aus Tabelle 6.3 mit einem Vergleich der Führerscheinnummer basierend auf Teilzeichenketten für das Auffinden der Alignments nicht brauchbar ist.

Die vorgegebenen Merkmale und Ähnlichkeitsfunktionen (Q_F, Q_S) mit den extrahierten Hypothesen (H_F, H_S) bilden eine erweiterte Sammlung (D_F, D_S). Diese stellt mit den dazu berechneten Ähnlichkeitswerten sowie der Bewertung der initialen Alignments durch den bzw. der BenutzerIn das Trainingsset A_V für das maschinelle Lernen dar.
Die gewählte Methode für das maschinelle Lernen eruiert nun anhand dieser Daten eine optimale Gewichtung D_W für die einzelnen Merkmale und eine optimalen Schwellwert(strategie) D_T. Für eine Beschreibung, wie die Werte ermittelt werden, sei an dieser Stelle auf [Hariri u. a., 2006] verwiesen. Abhängig von der Komplexität des Alignment Problems, kann es sinnvoll sein weitere Durchläufe für die Generierung von Testdaten und das Ermitteln der optimalen Gewichtungen bzw. des Schwellwertes durchzuführen. Es ergibt sich also ein Tupel (D_F, D_S, D_W, D_T), das die optimale Alignmentfunktion darstellt bzw. schließlich auf die vorliegenden Ontologien angewendet wird.

6.6 Konfiguration

Wie bereits in den vorhergehenden Abschnitten angedeutet, wird über Parameter der Alignment Prozess von FOAM konfiguriert. Listing 6.2 zeigt eine FOAM-Konfigurationsdatei mit den Parametern, die auf jeden Fall angegeben werden sollten. Die ersten beiden Parameter geben dabei den Pfad zu den beiden Eingabeontologien an. Beim Parameter `cutoffFile` wird definiert, wohin das Alignment-Ergebnis mit nur jenen Entitätspaaren, deren Ähnlichkeitswert über dem Schwellwert liegt, gespeichert werden soll. Ein Auszug einer Datei, die ein Ergebnis-Alignment enthält, wurde bereits in Listing 6.1 gezeigt.

```
ontology1 = C:/ontology1.owl;
ontology2 = C:/ontology2.owl;
cutoffFile = C:/result.txt;
```

Listing 6.2: FOAM-Konfigurationsdatei mit den Pflichtparametern [Quelle: Ehrig, 2006]

Bei FOAM werden weiters optionale Parameter verwendet. Für diese Parameter werden Standardwerte festgelegt, falls die entsprechenden Parameterwerte nicht gesetzt wurden, jedoch für den Alignment-Prozess erforderlich sind. Die Details, wann welche Standardwerte gesetzt werden, können im Programmcode (siehe Datei `Parameter.java` aus dem Package `edu.unika.aifb.foam.main`) eingesehen werden. Nachfolgend findet sich eine Auflistung der optionalen Parameter mit jeweils einer Kurzbeschreibung.

- `strategy = DECISIONTREE;`
  Mit diesem Parameter kann bestimmt werden, dass
  - nur die Labels der Entitäten (und keine weiteren Merkmale) verglichen werden und dabei genau übereinstimmen müssen, damit ein entsprechendes Alignment registriert wird (Parameterwert `EQUALLABELS`).
  - nur die Labels der Entitäten (und keine weiteren Merkmale) auf Ähnlichkeit überprüft werden (Parameterwert `ONLYLABELS`).
  - alle definierten Merkmalsvergleiche (siehe Tabelle 6.1 bzw. 6.2) durchgeführt und die Ähnlichkeitswerte mit festgelegten Werten gewichtet werden sollen (Parameterwert `MANUALWEIGHTED`). Wie bereits erwähnt, sind die Gewichte im Programmcode direkt definiert und können somit von außen nicht gesteuert werden. D.h. bei Gewichtsänderungen muss der Programmcode geändert und neu übersetzt werden.
  - alle definierten Merkmalsvergleiche durchgeführt und bei der Ähnlichkeitsaufsummierung die Sigmoid Funktion zum Einsatz kommt (Parameterwert `MANUALSIGMOID`). Auch hier sind die Parameter für die Sigmoid Funktion pro Merkmal im Programmcode festgelegt, wodurch dies auch nicht von außen beeinflusst werden kann.
  - beim Ontology Alignment-Prozess eine Support Vector Machine zum Einsatz kommen soll (Parameterwert `MACHINE`. Dazu sind allerdings auch die Parameterwerte für `classifierFile` sowie `rulesFile` erforderlich.

  - beim Ontology Alignment-Prozess ein Eintscheidungsbaum verwendet werden soll (Parameterwert `DECISIONTREE`). Auch diese Einstellung erfordert die Werte zu den Parametern `classifierFile` und `rulesFile`.

- `classifierFile = C:/tree.obj;`
  Die Datei, die den mit dem Framework WEKA erlernten Classifier bzw. Entscheidungsbaum enthält, ist erforderlich bzw. muss hier angegeben werden, wenn beim Parameter `strategy` `MACHINE` oder `DECISIONTREE` gewählt wurde.

- `rulesFile = C:/generatedRules.obj;`
  Dieser Parameter mit der Datei, welche mit WEKA generierte bzw. erlernte Regeln beinhaltet, wird ebenfalls bei den Strategien `MACHINE` und `DECISIONTREE` benötigt.

- `internaltoo = EXTERNAL;`
  Wenn Alignments auch innerhalb eines Namespaces erlaubt sind, so kann dies hier mit dem Wert `INTERNAL` spezifiziert werden. Beim Parameterwert `EXTERNAL` müssen die Entitäten eines Alignments aus unterschiedlichen Namespaces stammen.

- `efficientAgenda = EFFICIENT;`
  Ist der Parameterwert `EFFICIENT` gesetzt, so werden nur Entitäten, die nach bestimmten Algorithmen ausgewählt werden, verglichen (siehe Kapitel 6.3). Hingegen werden bei `COMPLETE` alle Entitäten mit gleichem Typ verglichen.

- `semi = FULLAUTOMATIC;`
  Der Wert zu diesem Parameter wird auf `SEMIAUTOMATIC` gestellt, wenn es erwünscht ist, dass die gefundenen Alignments aus einer Iteration von einem bzw. einer BenutzerIn validiert werden. Die validierten Alignments werden in der darauffolgenden Iteration für die Ermittlung eines verbesserten Alignment-Ergebnisses verwendet. Andernfalls wird der Parameter auf `FULLAUTOMATIC` gesetzt, wodurch ausschließlich das Alignment-Ergebis nach dem Durchlauf der festgelegten Iterationen (siehe Parameter `maxiterations`) geliefert wird.

- `cutoffvalue = 0.9;`
  Der Schwellwert, der basierend auf dem aggregierten Ähnlichkeitswert bestimmt, ob zwei Entitäten gleich sind oder nicht, wird hier definiert.

- `maxError = 0.9;`
  Für die Auswahl der zu validierenden Alignments durch den Benutzer wird der Wert beim Parameter `maxError` als Schwellwert verwendet.

- `maxiterations = 10;`
  Dieser Parameter legt die Anzahl der Iterationen des Alignment-Prozesses fest.

- `removeDoubles = REMOVEDOUBLES;`
  Mit `REMOVEDOUBLES` darf eine Entität nur in einem Alignment vorkommen bzw. wird nur jenes Alignment mit dem größten aggregierten Ähnlichkeitswert verwendet (*One Alignment Link* Strategie). `ALLOWDOUBLES` erlaubt das Vorkommen einer Entität in beliebig vielen Alignments (*Multiple Alignment Link* Strategie).

- `resultFile = C:/wholeResult.txt;`
  Die hier angegebene Datei, enthält nach dem Ontology Alignment Prozess alle gefundenen Alignments (auch jene, deren Ähnlichkeitswert über dem Schwellwert liegt) sowohl mit dem aggregierten Ähnlichkeitswert als auch mit den Ähnlichkeitswerten zu allen angewandten Ähnlichkeitsfunktionen.

- `explicitFile = C:/preknown.txt;`
  Sind bereits Alignments oder Entitäten, die gleich oder sicher nicht ähnlich sind, bekannt, so kann bei diesem Parameter der Dateiname mit den Alignments angegeben werden. Für bekannte gültige Alignments enthält die Datei Einträge in der Form `URI1;URI2;1`. Zwei Entitäten, die sicher nicht ähnlich sind, werden in der Form `URI1;URI2;0` vermerkt. `URI1` und `URI2` repräsentieren dabei den URI der entsprechenden Entitäten.

- `questionFile = C:/question.txt;`
  Alignments, deren aggregierter Ähnlichkeitswert gleich bzw. nahe beim Schwellwert ist bzw. liegt, werden in dieser Datei gespeichert, damit sie vom bzw. von der BenutzerIn überprüft werden können. Als Schwellwert wird hier der Wert beim Parameter `maxError` und nicht jener beim Parameter `cutoffvalue` verwendet. Die validierten Alignments sollten beim Anstoß der nächsten Iteration des Alignment-Prozesses übergeben werden.

- `numberQuestions = 5;` Hier wird die Anzahl der Alignments bestimmt, die dem bzw. der BenuzterIn zur Validierung in der Datei (siehe Parameter `questionFile`) gespeichert werden.

- `manualmappingFile = C:/goldstandard.txt;`
  Für eine Evaluierung des erzielten Alignment-Ergebnisses, welches im `cuttoffFile` gespeichert ist, kann hiermit die Datei mit den korrekten Alignments (Referenz-Alignments) übergeben werden.

Man kann sich leicht vorstellen, dass es beim Arbeiten mit Ontologien, die viele Entitäten enthalten, äußerst schwierig ist, den Überblick zu behalten, falls nur Programme

ohne visuelle Darstellungen zum Einsatz kommen. Dies gilt natürlich auch beim Betrachten der Ergebnis-Alignmentmenge nach dem Ontology Alignment zwischen solch großen Ontologien. Im nächsten Kapitel werden daher vorhandene Visualisierungsmethoden und Programme mit visuellen Komponenten vorgestellt, die es für Ontologien bzw. Ontology Alignment gibt.

Kapitel 7

AlViz - Ein Tool für visuelles Ontology Alignment

Im vorangegangenen Kapitel wurde in Listing 6.1 auszugsweise ein Alignment-Ergebnis von FOAM basierend auf dem Ontologie-Beispiel aus Kapitel 5.4.1 gezeigt. Bereits ab einer geringen Anzahl an gefundenen Alignments ist solch eine textuelle Liste für den Menschen schwierig zu verstehen und zu interpretieren. Eine wesentliche Erleichterung können dabei Methoden aus der Informationsvisualisierung (*InfoVis*) bereitstellen. InfoVis bietet Techniken, die die Repräsentation von hierarchischen und semistrukturierten Daten mit Hilfe von visuellen Metaphern übernehmen. Graphische Elemente wie Punkte, Linien, Flächen oder Volumina werden zur Kodierung von Informationen eingesetzt. Diese Elemente sind durch Größe, Form, Ausrichtung, Raumposition, Farbe, Textur usw. charakterisiert. [Stuart u. a., 1999] Das Ziel von InfoVis ist, dass durch die Informationsdarstellung auf einer intuitiven Ebene ein besseres Verständnis sowie neue Einsichten in die Daten ermöglicht wird. [Tufte, 2001] Dabei spielt Abstraktion eine wesentliche Rolle. Mit den Methoden der InfoVis können bestimmte Informationen ausgeblendet bzw. vermindert oder ausgewählte Aspekte der Daten verstärkt werden, damit die Interpretation dieser Daten im Hinblick auf entsprechende Ziele leichter fällt. Weiters wird dies durch die Kombination verschiedener Visualisierungsmethoden gefördert. [Lanzenberger und Sampson, 2006a]

Nach [Lanzenberger, 2008] sollen Visualisierungtechniken die Arbeit mit Ontologien bei folgenden Aspekten unterstützen:

- direkte Manipulation von Entitäten,
- Behalten des Überblicks,
- Fokusieren auf die Ontologiestruktur oder auf die Instanzdaten,

- effizientes Vergleichen von Ontologien und
- Erkunden bzw. Abfragen der Ontologiedaten.

Allerdings ist die Visualisierung von Ontologien keine leichte Aufgabe, da dies sowohl eine effektive Darstellung aller Informationen (Konzepte, -hierarchien, Relationen zwischen Konzepten, Instanzdaten und beim Ontology Alignment auch die gefundenen Alignments) beinhaltet als auch dem bzw. der BenutzerIn die Durchführung bestimmter Operationen bei den Ontologien bzw. Alignments ermöglichen soll. [Katifori u. a., 2007]

Derzeit gibt es viele Tools für die Visualisierung von Ontologien mit unterschiedlichen Visualisierungsmethoden wie z.B. OntoViz [Sintek, 2007] (2D Graph Visualisierungsmethode), OntoSphere [Bosca u. a., 2005] (3D Graph Visualisierungsmethode), Jambalaya [Storey u. a., 2002] (Simple Hierachical Multi-Perspective - eine Visualisierungsmethode nach [Wu und Storey, 2000]) und TGVizTab [Alani, 2003] (TouchGraph[1] Visualisierungsmethode). Weitere Tools sowie Kurzbeschreibungen dazu lassen sich in [Katifori u. a., 2007] finden. Leider gibt es nur sehr wenige Tools wie z.B. AlViz [Lanzenberger und Sampson, 2006a], OLA [Euzenat u. a., 2006] und OPTIMA [Kolli und Doshi, 2008], die Ontology Alignment visuell unterstützen.

Da AlViz - neben FOAM (siehe Kapitel 6) - für den praktischen Teil dieser Arbeit von großer Bedeutung ist, wird nun der Fokus auf dieses Tool gelegt.

Der erste Prototyp von AlViz sowie die Grundlagen dazu wurden in [Lanzenberger und Sampson, 2006a] präsentiert. Im Rahmen der Masterarbeit von Martin Huber ([Huber, 2009]) wurde das Tool um weitere visuelle Aspekte für die Unterstützung des Ontology Alignments ergänzt bzw. verbessert. Diese beiden Arbeiten bilden somit die Literaturgrundlage für die folgenden Unterabschnitte. Dabei werden zunächst die Grundlagen von AlViz mit den verwendeten Basis-Visualisierungstechniken und Basiskomponenten der graphischen Benutzeroberfläche erläutert. Danach wird näher auf die Ähnlichkeitsklassen/-regeln, die bei der Festlegung bzw. Änderung von Alignments zum Einsatz kommen, eingegangen. Weiters wird die Benutzerinteraktion beim Bestimmen von Alignments mit AlViz beschrieben. Im vorletzten Unterkapitel werden die erforderlichen Eingabedateien für AlViz angeführt und das XML-Format mit Alignments erklärt. Abschließend werden offene Aspekte bzw. neue Ideen für die zukünftige Weiterentwicklung und Verbesserung von AlViz aufgezählt.
Es sei an dieser Stelle angemerkt, dass die nachstehenden Unterkapitel vor allem die

[1] http://www.touchgraph.com

relevanten Informationen für das Verständnis des praktischen Teiles der vorliegenden Arbeit wiedergeben. Für detailliertere Beschreibungen bezüglich weiterer intergrierter Visualisierungskonzepte sowie weiterer graphischer Benutzerdialoge sei auf [Huber, 2009] verwiesen.

7.1 Grundlagen

> *„AlViz soll dazu dienen, den durch den Menschen durchgeführten Nachbearbeitungs- und Korrekturprozess eines Alignment-Ergebnisses eines bestehenden automatisierten Systems mittels geeigneter Visualisierungs- und darauf aufbauenden Bearbeitungsmechanismen zu unterstützen.“* [Huber, 2009]

Das Tool ist als *Multiple View* Plug-In für Protégé [Gennari u. a., 2002] (Version 3.3.1) in Java (Version 1.5) implementiert und setzt daher auch dieses als Ausführungsumgebung voraus. Multiple View bedeutet hier, dass die selben Daten mit mehreren unterschiedlichen Visualisierungsmethoden dargestellt werden. Bei AlViz wird eine Ontologie zum einen mit einem J-Tree und zum anderen mit einem Small World Graph [van Ham und van Wijk, 2004] visualisiert. Derzeit werden nur die Konzepte bzw. Konzepthierarchien sowie auch nur Alignments zwischen Konzepten visualisiert.

7.1.1 Visualisierungstechniken

Ein J-Tree visualisiert die Ontologie durch eine klassische Baumstruktur mit einem Wurzelknoten, inneren Knoten und Blattknoten. Die Struktur der Knoten, wie sie z.B. auch von Dateimanagern bekannt ist, repräsentiert die hierarchische Struktur von Konzepten bzw. Klassen. Durch einen Mausklick auf einen Elternknoten können die Kindknoten auf- und zugeklappt werden. Weiters erlaubt der J-Tree das Hinzufügen, Bearbeiten und Löschen von Konzepten. Diese Veränderung der Ontologiedaten innerhalb des J-Trees wird derzeit von AlViz allerdings nicht auf den dazugehörigen Small World Graph übernommen.
Der große Nachteil von J-Trees bzw. der baumstrukturierten Darstellung ergibt sich bei der Anzeige von Ontologien mit vielen Konzepten und Hierarchieebenen, da man die Übersicht über die ganze Ontologie sehr leicht verliert.

Das Problem des fehlenden Überblicks über große Ontologien wird bei AlViz mit einem Graphen, im speziellen mit dem Small World Graph, bewältigt.

Der Small World Graph basiert auf dem *Kleine-Welt-Phänomen* aus dem sozialpsychologischen Bereich. Dabei ist gemeint, dass sich die Personen einer Gesellschaft nicht alle direkt kennen, aber dennoch zwischen allen Personen kurze Wege über persönliche Beziehungen innerhalb der sozialen Vernetzung existieren. [Milgram, 1967]
Dieses Phänonmen wurde auf Graphen übertragen, daher auch der Name Small World Graph. Dieser Graph ist durch eine kleine durchschnittliche Pfadlänge (durchschnittliche Länge der kürzesten Pfade zwischen zwei Knoten) und einen hohen Vernetzungsgrad (Verhältnis zwischen Zahl der tatsächlichen Verbindungen und der Zahl der theoretisch möglichen Verbindungen) gekennzeichnet. Weiters weist die Struktur des Graphen weder eine vollstängie Ordnung noch eine vollständige Unordnung auf. [Watts und Strogatz, 1998; Huber, 2009]
Neben der intuitiven Erfassung und Analyse der Ontologiestruktur ergibt sich durch die Graphendarstellung die Möglichkeit der Clusterung von Knoten. D.h. es können - abhängig von einem wählbaren Detailgrad - Knoten bzw. Konzepte entsprechend der Konzepthierarchie gruppiert bzw. zusammengefasst werden. Somit repräsentieren die Knoten des Graphen entweder genau ein Konzept oder eine Menge von Konzepten. Je größer der Detailgrad gewählt wurde, desto mehr Knoten sind sichtbar. Die Kanten zwischen den Knoten bilden die hierarchische Struktur der Konzepte ab.

Die Visualisierung einer Ontologie durch den J-Tree und den Small World Graph sind miteinander gekoppelt. Werden im J-Tree ein oder mehrere Elemente selektiert, so wird diese Selektion auch auf den Graph übernommen bzw. werden die entsprechenden Knoten hervorgehoben. Genauso wird/werden bei der Selektion eines Graph-Knotens der/die entsprechende(n) Eintrag/Einträge markiert.

7.1.2 Basiskomponenten der graphischen Benutzeroberfläche

Abbildung 7.1 zeigt die schematische Darstellung von AlViz, welche aus drei Komponenten besteht:

- Bedienleiste: Enthält Steuerelemente für die Änderung der Darstellungsart und für das Laden sowie Speichern der aktuellen Alignments.
- Ontologie 1: In diesem Bereich werden links der J-Tree und rechts der Small World Graph der ersten Ontologie eingebettet.
- Ontologie 2: Der J-Tree und der Small World Graph der zweiten Ontologie werden innerhalb dieses Bereichs dargestellt.

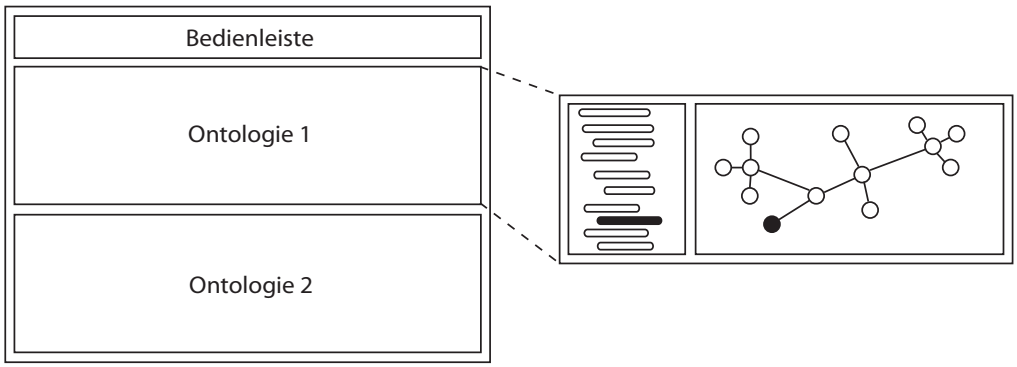

Abbildung 7.1: Schematische Darstellung von AlViz [Quelle: Huber, 2009]

Damit die Alignments zwischen den beiden Ontologien visualisiert werden können, ist eine Kopplung zwischen den Komponenten *Ontologie 1* und *Ontologie 2* vorhanden. Jene Ontologie des Konzeptes, von der aus die dazugehörigen Alignments ermittelt bzw. dargestellt werden sollen, wird im weiteren Verlauf der Arbeit als *aktive* Ontologie bezeichnet. Die *passive* Ontologie ist jene, in der die korrespondierenden Entitäten gesucht bzw. dargestellt werden. Abbildung 7.2 zeigt die Unterscheidung zwischen aktiver und passiver Ontologie. Die obere Ontologie ist im dargestellten Fall auch die aktive. Erkennbar ist dies an den senkrechten Balken links und rechts der Graphendarstellung sowie auch durch die farbliche Hervorhebung (schwarz) des selektierten Knotens im Graph und in der Baumdarstellung. In der passiven Ontologie werden die Knoten bzw. Klassen, zu welchen bereits eine Ähnlichkeit von der selektieren Klasse aus der aktiven Ontologie besteht, auch farblich hervorgehoben (grau).

Durch Selektion (Klick mit der linken Maustaste) eines anderen Knotens im Graph bzw. Eintrags im J-Tree aus der aktiven Ontologie wird der neue Knoten farblich markiert und die Visualisierung der passiven Ontologie entsprechend aktualisiert. Durch Selektion eines Elementes aus der passiven Ontologie wird diese automatisch zur aktiven Ontologie und die vorhin aktive Ontologie wird zur passiven Ontologie. Somit ist die passive Ontologie nicht mit der Komponente *Ontologie 1* aus Abbildung 7.1 gleichzusetzen.

Abbildung 7.3 zeigt einen Screenshot von AlViz mit den beiden Ontologien aus Kapitel 5.4.1. Dabei ist die obere Ontologie die aktive und die untere die passive. Weiters visualisiert AlViz in der Abbildung eine von FOAM ermittelte Ähnlichkeit zwischen dem Konzept `Speed` der oberen und dem Konzept `Speed` der unteren Ontologie. Damit allerdings der Graph für die beiden Ontologien visualisiert werden kann, muss ein zusätzliches Wurzel-Konzept erstellt werden, von dem aus sich jene Konzepte ableiten,

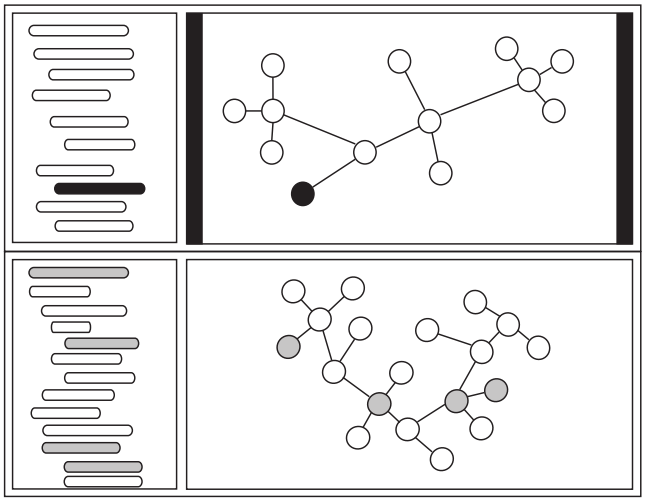

Abbildung 7.2: Unterscheidung zwischen der aktiven und passiven Ontologie [Quelle: Huber, 2009]

die vorher noch kein Superkonzept hatten. Somit ist es für AlViz notwendig, dass aus den zugrundeliegenden Ontologien ein zusammenhängender Graph aufgebaut werden kann.

7.2 Alignments

AlViz setzt voraus, dass basierend auf den beiden Ontologien bereits ein Ontology Alignment mit FOAM (siehe Kapitel 6) durchgeführt wurde. Daher muss, bevor die Visualisierungen dargestellt werden, eine Alignmentdatei mit den von FOAM ermittelten Alignments in einem festgelegten XML-Format (siehe Kapitel 7.3) vom Benutzer bzw. von der Benutzerin ausgewählt werden. Auf Grund der Voraussetzung, dass die generierten Alignments nur von FOAM kommen, wurden bei AlViz die von FOAM verwendeten Regeln zur Bestimmung eines Alignments bzw. einer Ähnlichkeit integriert und zu Ähnlichkeitsklassen gruppiert.
Die Tabelle 7.1 zeigt alle vorhandenen Ähnlichkeitsklassen, die zugeordneten Regeln und die farblichen Hervorhebungen, die in der passiven Ontologie bei vorhanden sein der entsprechenden Ähnlichkeitsklasse bzw. -regel verwendet werden. Zu beachten sind die letzten zwei Ähnlichkeitsklassen/-regeln. Mit der Ähnlichkeitsregel *unknown relation to...* kann der bzw. die BenutzerIn eine Ähnlichkeitsbeziehung definieren, wobei der

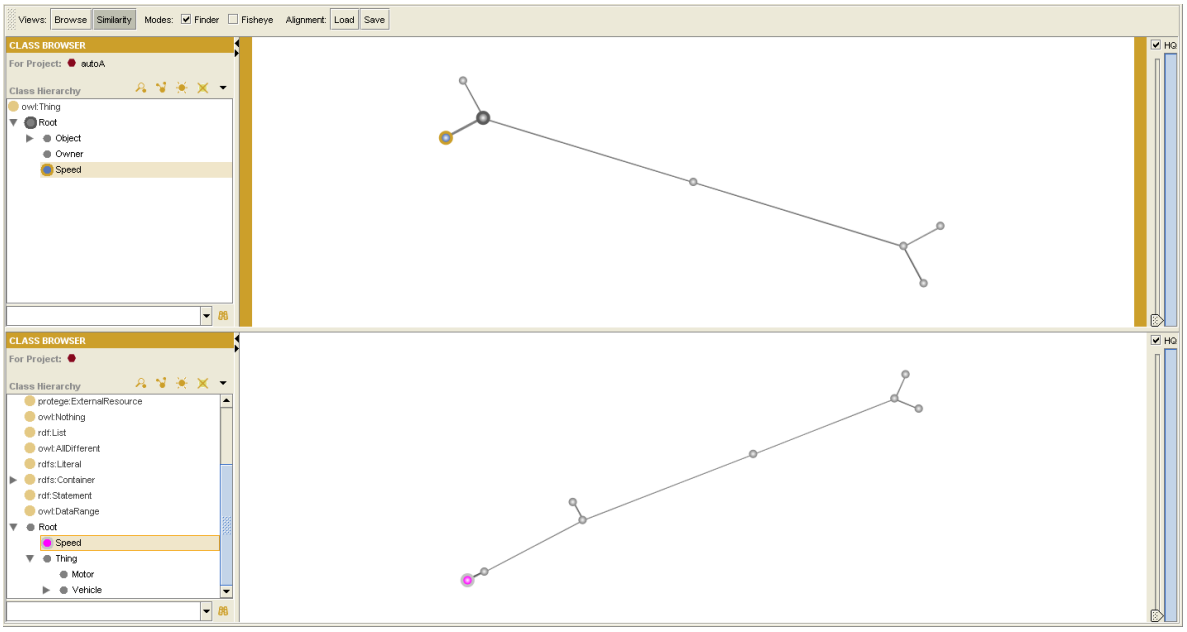

Abbildung 7.3: AlViz mit den Ontologien aus Kapitel 5.4.1

konkrete Ähnlichkeitstyp nicht bekannt ist. Es wird dabei automatisch eine 100%-ige Wahrscheinlichkeit für die Ähnlichkeit zwischen den Konzepten angenommen. Steht für den bzw. die BenutzerIn fest, dass keine Ähnlichkeit zwischen zwei Konzepten besteht, so kann dies mit der Ähnlichkeitsregel *unrelated to...* bestimmt werden. Hier wird die Wahrscheinlichkeit für die Richtigkeit dieser Regel ebenfalls auf 100 % gesetzt. Dies heißt somit auch, dass die Wahrscheinlichkeit einer Ähnlichkeit sicher bei 0 % liegt. Bei den restlichen Regeln kann der bzw. die BenutzerIn einen Wert zwischen 0.01 und 1 wählen, der die Wahrscheinlichkeit der Gültigkeit der entsprechenden Regel repräsentiert - man spricht auch von Konfidenzwert.
Die Elemente in der Baumdarstellung sowie die Knoten im Graph, die bezüglich des aktuell selektierten Knotens keine Zuweisung zu einer Ähnlichkeitsregel aus 7.1 haben, werden grau eingefärbt.
AlViz unterstützt derzeit nur die Visualisierung genau einer Ähnlichkeitsregel. D.h. werden z.B. durch FOAM zwischen zwei Konzepten bezüglich drei Regeln die Konfidenzwerte 0.3, 0.5 und 1.0 ermittelt, so visualisiert AlViz nur die Regel mit dem größten Konfidenzwert, welcher in diesem Fall 1.0 ist. Die ermittelten Regeln und Konfidenzwerte, die nicht visualisiert werden, bleiben allerdings erhalten bzw. werden beim Speichern des aktuellen Alignments von AlViz in einer XML-Datei mitabgelegt.

Ähnlichkeitklasse	**Ähnlichkeitsregel (interne Id)**	**Farbe**
gleich (*equal*)	*Equal URI (1)* *Equal class member individuals (7)*	rot
syntaktisch gleich (*syntactically equal*)	*Syntactic similarity (0)*	magenta
ähnlich (*similar*)	*Similar superclasses (2)* *Similar subclasses (3)* *Similar class data properties (4)* *Similar class object properties from... (5)* *Similar class object properties to... (6)* *Similar data property domain (10)* *Similar super data properties 11)* *Similar sub data properties (12)* *Similar data property members (13* *Similar object property domain (14)* *Similar object property range (15)* *Similar object property super (16)* *Similar object property sub (17)* *Similar object property members (18)* *Similar individual member of class (19)* *Similar individual data properties (20)* *Similar individual object property members from... (21)* *Similar individual object property members to... (22)*	grün
allgemeiner-als (*broader-than*)	*Broader than (9)*	blau
spezieller-als (*narrower-than*)	*Narrower than (10)*	türkis
unbekannt (*unknown*)	*Unknown relation to... (-2)*	schwarz
keine Ähnlichkeit (*unrelated*)	*Unrelated to... (-1)*	gelb

Tabelle 7.1: Ähnlichkeitsklassen, -regeln und farbliche Hervorhebung in der passiven Ontologie [Quelle: Huber, 2009]

7.2.1 Definition bzw. Änderung eines Alignments mit AlViz

Bisher wurden nur die Visualisierungsaspekte rund um Alignments bzw. Ähnlichkeiten zwischen Konzepten erläutert, allerdings nicht wie der bzw. die BenutzerIn mit Hilfe von AlViz neue Alignments definiert sowie von FOAM geladene Alignments ändert. Dies wird nun Schritt für Schritt beschrieben.

1. Sobald ein Element in der Baumdarstellung oder ein Knoten im Graph in einer der beiden Ontologien selektiert (Klick mit der linken Maustaste) wurde, wird die entsprechende Ontologie zur aktiven Ontologie.
2. In der passiven Ontologie sind nun Elemente bzw. Knoten entweder grau, sofern vom Konzept der aktiven Ontologien zum entsprechenden Konzept der passiven Ontologie keine Ähnlichkeit definiert oder gefunden wurde, oder in der Farbe korrespondierend zur Tabelle 7.1. Durch einen Klick mit der rechten Maustaste auf einen bestimmten Knoten im Graph der passiven Ontologie (Achtung: in der Baumdarstellung ist dies nicht möglich) öffnet sich ein Kontextmenü mit allen möglichen Ähnlichkeitsklassen und falls eine Ähnlichkeitsbeziehung zwischen den Konzepten bereits definiert wurde, auch die aktuelle Ähnlichkeitsregel.
3. Durch Klick mit der linken Maustaste auf eine Ähnlichkeitsklasse kann der bzw. die BenutzerIn den Konfidenzwert für eine bestimmte Regel der gewählten Ähnlichkeitsklasse festlegen. Bei den Ähnlichkeitsklassen *unknown* und *unrelated* muss der bzw. die BenutzerIn nur einen sich öffnenden Dialog bestätigen, wodurch automatisch 0 bzw. 1 als Konfidenzwert herangezogen wird.
   Beim Klick auf die aktuelle Ähnlichkeitsregel gelangt der bzw. die BenutzerIn automatisch zum Dialog mit allen Ähnlichkeitsregeln der zugehörigen Klassen, wobei die aktuelle Ähnlichkeitsregel vorausgewählt ist.
4. Nach Bestätigung der Eingabe werden die Visualisierung der passiven Ontologie aktualisiert sowie die neuen Daten für die interne Verarbeitung übernommen.

Für die Definition von Alignments ist zu beachten, dass die festgelegte Ähnlichkeitsbeziehung nur von der Klasse der aktiven Ontologie auf die Klasse der passiven Ontologie übernommen wird. D.h. es wird nicht automatisch die Beziehung vom Konzept der passiven auf das Konzept der aktiven Ontologie mit übernommen, somit ist standardmäßig eine mit AlViz definierte Ähnlichkeitsbeziehung im mathematischen Sinne nicht symmetrisch.
Weiters ist anzumerken, dass die Festlegung eines Alignments mit AlViz zwischen einem

Knoten im Graph der aktiven Ontologie und einem Knoten im Graph der passiven Ontologie nur dann möglich ist, wenn der Knoten des Graphen der aktiven Ontologie genau ein Konzept repräsentiert. Würde der Knoten im Graph der aktiven Ontologie mehrere Konzepte beinhalten, so wäre die Darstellung verschiedener Ähnlichkeitsregeln/-klassen, die von unterschiedlichen Konzepten des einen Knotens ausgehen, nicht mehr nachvollziehbar, was den Designprinzipien für InfoVis widerspricht. Stellt der Knoten im Graph der passiven Ontologie mehr als ein Konzept dar, so übernehmen alle Konzepte die festgelegte Ähnlichkeitsregel mit dem Konfidenzwert.

7.3 Eingabedateien und -formate

AlViz benötigt als Eingabedateien zunächst die beiden Ontologien zwischen den die Alignments visualisiert bzw. verändert werden sollen. Die Ontologiedateien müssen dabei im OWL-Format vor liegen (siehe Kapitel 4). Da eine Ontologie durch das Protégé-Projekt bereits geladen wird, muss nur noch die zweite Ontologiedatei explizit vom Benutzer bzw. von der Benutzerin ausgewählt werden.
Außerdem ist die XML-Datei, welche die von FOAM generierten oder mit AlViz gespeicherten Alignments mit den Ähnlichkeitsregeln sowie Konfidenzwerten enthält, als Eingabe notwendig. Listing 7.1 zeigt die Grundstruktur einer solchen XML-Datei.

```
<?xml version="1.0" encoding="UTF-8"?>
<AlViz>
  <Nodes>
    <Node uri="http://www.example.com/ontologies/auto/autoA.owl#Speed">
        <Entity label="Speed" type="C" />
        <CNode uri="http://www.example.com/ontologies/auto/autoB.owl#Speed">
           <Entity label="Speed" type="C" />
           <Similarity rule="0" value="1.0"/>
            <!-- Syntactical Similarity -->
           <SimilarityInfo Correctvalue="0">
        <Confidence value="1.0" />
           </SimilarityInfo>
        </CNode>
    </Node>
</AlViz>
```

Listing 7.1: Grundstruktur der XML-Datei mit Alignments

Das `AlViz`-Tag repräsentiert dabei das Wurzelelement und umschließt das `Nodes`-Element, welches alle ermittelten Alignments enthält. Im konkreten Fall ist nur ein Alignment zwischen den URIs `http://www.example.com/ontologies/auto/autoA.owl#Speed` (siehe Attribut `uri` des `Node`-Tags) und `http://www.example.com/ontologies/auto/autoB.owl#Speed` (siehe Attribut `uri` des `CNode`-Tags) vorhanden. Innerhalb des `Node`- sowie `CNode`-Tags gibt es ein `Entity`-Tag, welches den Wert des `rdfs:label`-Tags aus der Ontologie (siehe Attribut `label`) und den Typ der entsprechenden Entität (siehe Attribut `type`) speichert.
Als Entitätstyp können C für Konzepte bzw. Klassen, D für Dateneigenschaften, O für Objekteigenschaften und I für Instanzen auftreten.
Innerhalb des `CNode`-Elements befindet sich weiters für jede zutreffende Ähnlichkeitsregel ein `Similarity`-Tag mit Informationen zur Id der Regel (siehe Attribut `rule` - dies entspricht der internen Id aus Tabelle 7.1) und dem Konfidenzwert zu dieser Regel (siehe Attribut `value`). Am Ende eines `CNode`-Tags befindet sich ein `SimilarityInfo`-Element mit dem Attribut `Correctvalue`, welches aussagt, ob das Alignment von einem Benutzer bzw. einer Benutzerin als richtig (Wert 1) oder falsch (Wert 0) validiert wurde. Das darin enthaltene `Confidence`-Tag zeigt den aggregierten Ähnlichkeitswert zwischen beiden Entitäten an.
Sowohl das Attribut `Correctvalue` des `SimilarityInfo`-Tags als auch das Attribut `value` des `Confidence`-Tags werden derzeit von AlViz dem bzw. der BenutzerIn nicht veranschaulicht.

Die Beziehung zwischen dem `Node`- und `CNode`-Elements eines Alignments kann man sich in der AlViz-Visualisierung so vorstellen, dass das `Node`-Element dem Graphknoten der aktiven Ontologie entspricht und das `CNode`-Element dem Graphknoten der passiven Ontologie. Da laut Tabelle 7.1 die Ähnlichkeitsregel 0 der Ähnlichkeitsklasse *syntactically equal* angehört, wird der Knoten in der passiven Ontologie mit der Farbe magenta hervorgehoben (siehe Abbildung 7.3 auf Seite 95).

7.4 Offene Punkte

Die Implementierungen an AlViz sind bei Weitem noch nicht abgeschlossen und es gibt ständig Verbesserungspotenzial bzw. neue Ideen, die umgesetzt und eingebaut werden können.
In [Lanzenberger, 2008] werden unter anderem folgende offenen Punkte in Verbindung mit AlViz aufgezählt:

- Integration von Graphanalyseverfahren, um die Analyse der Alignments zu unterstützen,
- weitere Focus + Context Visualisierungstechniken integrieren,
- Informationen des `SimilarityInfo`- bzw. `Correctvalue`-Tags in die Verarbeitung bzw. Visualisierung mit einbeziehen,
- Beschriftung und Einfärbung von Kanten im Small World Graph und
- stärkere Integration des Ontology Alignment Algorithmus in AlViz.

Der zuletzt angeführte Punkt, nämlich die stärkere Integration des Ontology Alignment Algorithmus in AlViz, war Anlass für die vorliegende Arbeit. Die praktische Umsetzung dieses Punktes wird in den folgenden zwei Kapiteln struktiert nach Analyse, Design, Implementierung und Evaluierung skizziert.

Kapitel 8

Praktischer Teil: Analyse & Design

Nachdem die wesentlichen Aspekte der aktuellen Funktionalität von AlViz und somit die Ausgangssituation für den praktischen Teil in Kapitel 7 dargestellt wurde(n), erfolgt in diesem Kapitel die Erläuterung der Ziele und Anforderungen der praktischen Umsetzung sowie des entsprechenden software-technischen Designs.

8.1 Analyse

Wie bereits in Kapitel 7 angemerkt wurde, ist für die Verwendung von AlViz eine XML-Datei im definierten Format (siehe Kapitel 7.3), welche die von FOAM gefundenen Alignments enthält, Voraussetzung. Weiters ist die Zuordnung der Regel-Id (siehe `rule`-Tag im Eingabe-XML) zur konkreten FOAM-Regel und daher auch zur Ähnlichkeitsklasse im Programmcode von AlViz verankert. Dies zeigt, dass die aktuelle AlViz-Version sehr eng an das Alignment-Ergebnis von FOAM gekoppelt ist. Abgesehen davon bietet AlViz momentan keine Möglichkeit FOAM oder einen anderen Ontology Alignment Algorithmus für die Ermittlung von Alignments direkt zu starten und die daraus resultierenden Ähnlichkeiten unmittelbar zu visualisieren.

AlViz soll dahingehend erweitert werden, sodass Ontology Alignment Algorithmen stärker in AlViz integriert werden können, damit z.B. der bzw. die BenutzerIn nach Änderungen von Ähnlichkeitsbeziehungen eine Aktualisierung der Alignments anstoßen kann. Für die Umsetzung dieser Erweiterung soll als globales Ziel die Einhaltung einer losen Kopplung zwischen AlViz und einem beliebigen Alignment Algorithmus verfolgt werden.

Im Zusammenhang mit dieser Erweiterung wurden folgenden Anforderungen definiert:

- Noch bevor die Visualisierung der beiden zugrunde liegenden Ontologien angezeigt wird, soll sich der bzw. die BenutzerIn entscheiden können, ob entweder
  - Alignments von einer XML-Datei, welches einem defierten Format entspricht, geladen werden sollen,
  - ein bestimmter Ontology Alignment Algorithmus für die Generierung der Alignments aufgerufen werden soll oder
  - vorerst gleich die Ontology-Visualisierung ohne Alignments gezeigt werden soll und der bzw. die BenutzerIn dann bereits bekannte Alignments selbst definiert.

  Nachdem der bzw. die BenutzerIn gewählt und bestätigt hat, wird in den ersten beiden Optionen die entsprechende Aktion durchgeführt und die Visualisierung aufgebaut. Bei der dritten Option werden sofort die Ontologien visualisert.
- Das Laden von Alignments aus einer XML-Datei sowie auch das Generieren bzw. Aktualisieren von Alignments mit einem Alignment Algorithmus, muss auch möglich sein, nachdem die Visualisierung der Ontologien und möglicher Alignments geladen und dargestellt wurde.
  Beim Laden sollen alle bestehenden Alignments gelöscht und nur jene aus der Datei dem bzw. der BenutzerIn ersichtlich sein.
  Beim Generieren bzw. Aktualisieren sollen Alignments, falls welche vorhanden sind, dem gewünschtem Ontology Alignment Algorithmus übergeben werden.
- Sobald AlViz die Visualisierungen geladen und angezeigt hat, soll es möglich sein vorhandene Alignments in einer Datei zu speichern. Dabei entspricht das Speicherformat dem bereits definierten XML-Format, damit diese Datei von AlViz klarerweise auch wieder geladen werden kann.
- Damit der bzw. die BenutzerIn die Alignments besser analysieren kann, soll eine Filterfunktion basierend auf Ähnlichkeitswertintervallen zur Verfügung stehen und nach Intervalländerungen eine sofortige Aktualisierung der Visualisierung bewirken.
- Mittels Konfigurationsdatei sollen Ontology Alignment Algorithmen in AlViz eingebunden werden. Zusätzlich soll zu jedem Alignment Algorithmus optional ein Parameter-Konfigurationsdialog angegeben werden können, der vor der Alignmentgenerierung des entsprechenden Algorithmus vom Benutzer bzw. von der Benutzerin zur Festlegung spezifischer Parameter aufgerufen werden kann.

- Der Alignment Algorithmus soll AlViz den aggregierten Ähnlichkeitswert und optional auch regelbasierte Ähnlichkeitswerte - sofern welche vorliegen - zu einem Alignment übergeben können. Unter regelbasierten Ähnlichkeitswerten werden jene Ähnlichkeitswerte zu einem Alignment verstanden, aus denen sich der aggregierte Ähnlichkeitswert zusammensetzt bzw. die zu einer bestimmten Ähnlichkeitsregel erfasst wurden. Beispiele für Ähnlichkeitsregeln sind in Tabelle 7.1 auf Seite 96 angeführt.
  Übergibt der Alignment Algorithmus regelbasierte Ähnlichkeitswerte, so muss der Algorithmus auch eine kurze Beschreibung zur entsprechenden Ähnlichkeitsregel mitgeben und diese einer von AlViz vordefinierten Ähnlichkeitsklasse zuweisen. Somit sollen die konkreten Ähnlichkeitsregeln nur vom Algorithmus festgelegt werden.

- Jeder konfigurierte Ontology Alignment Algorithmus soll beliebig oft gestartet werden können bzw. soll nachdem z.B. Algorithmus A die Alignments generiert hat, auch Algorithmus B, der die von A gefundenen Alignments als Eingabe erhält, angestoßen werden können. Für den Folge-Algorithmus - im erwähnten Beispiel ist dies Algorithmus B - muss es möglich sein zumindest die aggregierten Ähnlichkeitswerte zu verarbeiten.

- Der bzw. die BenutzerIn soll festlegen können, ob entweder die aggregierten Ähnlichkeitswerte (a) oder die regelbasierten Ähnlichkeitswerte (b) in der Visualisierung dargestellt werden sollen. Bei Änderung von einer Option auf die andere soll sich die Visualisierung sofort aktualisieren.
  Wird (a) gewählt, so soll nur bestimmt werden können, mit welcher Wahrscheinlichkeit (Ähnlichkeitswertebereich zwischen 0.1 und 1.0) zwei Entitäten gleich sind oder dass zwei Entitäten nicht in Beziehung stehen (Ähnlichkeitswert 0.0).
  Wird (b) gewählt, so sollen für den bzw. die BenutzerIn in AlViz nur die Ähnlichkeitsregeln ersichtlich sein und Alignments zugewiesen werden können, die vom letzten aufgerufenen Ontology Alignment Algorithmus festgelegt oder von der XML-Datei geladen wurden.
  Liegen noch keine Ähnlichkeitsregeln vor, da z.B. noch kein Algorithmus-Durchlauf gestartet wurde oder keine XML-Datei geladen wurde, welche Ähnlichkeitsregeln enthält, so soll nur (a) gewählt werden können.

8.2 Design

Dieser Unterabschnitt beschreibt das software-technische Design, wobei auf die wesentlichen Komponenten der Architektur von AlViz eingegangen wird, die für die Erweiterung notwendig ist und in Abbildung 8.1 dargestellt wird.

In der Abbildung trennen die blau strichlierten Linien AlViz von dem bzw. der in Interaktion stehende(n) BenutzerIn und den Ontology Alignment Algorithmen.
Beim Start von AlViz werden über die Datei `alvizconfig.xml` alle konfigurierten Alignment Algorithmen eingebunden (siehe Kapitel 8.2.2). Der bzw. die BenutzerIn gibt die beiden OWL-Ontologien an und kann entweder Alignments - sofern ihm bzw. ihr welche bekannt sind - definieren, bevor ein Algorithmus gestartet wird oder Alignments aus einer Datei laden, die mit AlViz davor erstellt wurden. Bevor einer der eingebundenen Algorithmen zur Generierung bzw. Aktualisierung von Alignments angestoßen wird, kann der bzw. die BenutzerIn die Parameter für den ausgewählten Algorithmus über einen Konfigurationsdialog spezifizieren. Da der Konfigurationsdialog vom jeweiligen Alignment Algorithmus zur Verfügung gestellt wird und nur optional in der Datei `alvizconfig.xml` angegeben werden muss, und somit nicht immer Parameter vom Benutzer bzw. von der Benutzerin eingestellt werden können, wurden in Abbildung 8.1 die entsprechenden Verbindungen strichliert gezeichnet.
Beim Starten des Alignment Algorithmus werden die beiden Ontologien, Alignments - sofern schon welche vorhanden sind - und optional für den Algorithmus spezifische Parameter in einem einheitlichen Format von AlViz an den Alignment Algorithmus übergeben. Nachdem die Menge der Alignments innerhalb des Algorithmus aktualisiert wurde, wird diese an AlViz zurückgegeben. Für den Austausch der Ontologien, Alignments und Parameter kommt die Ontology Alignment API und Erweiterungen davon zum Einsatz.

In den folgenden Unterkapiteln werden zuerst auf Grund der Relevanz in der Architektur die Ontology Alignment API und danach die Einbindung von Alignment Algorithmen sowie das XML-Format, welches beim Laden und Speichern von Alignments bei AlViz verwendet wird, erläutert.

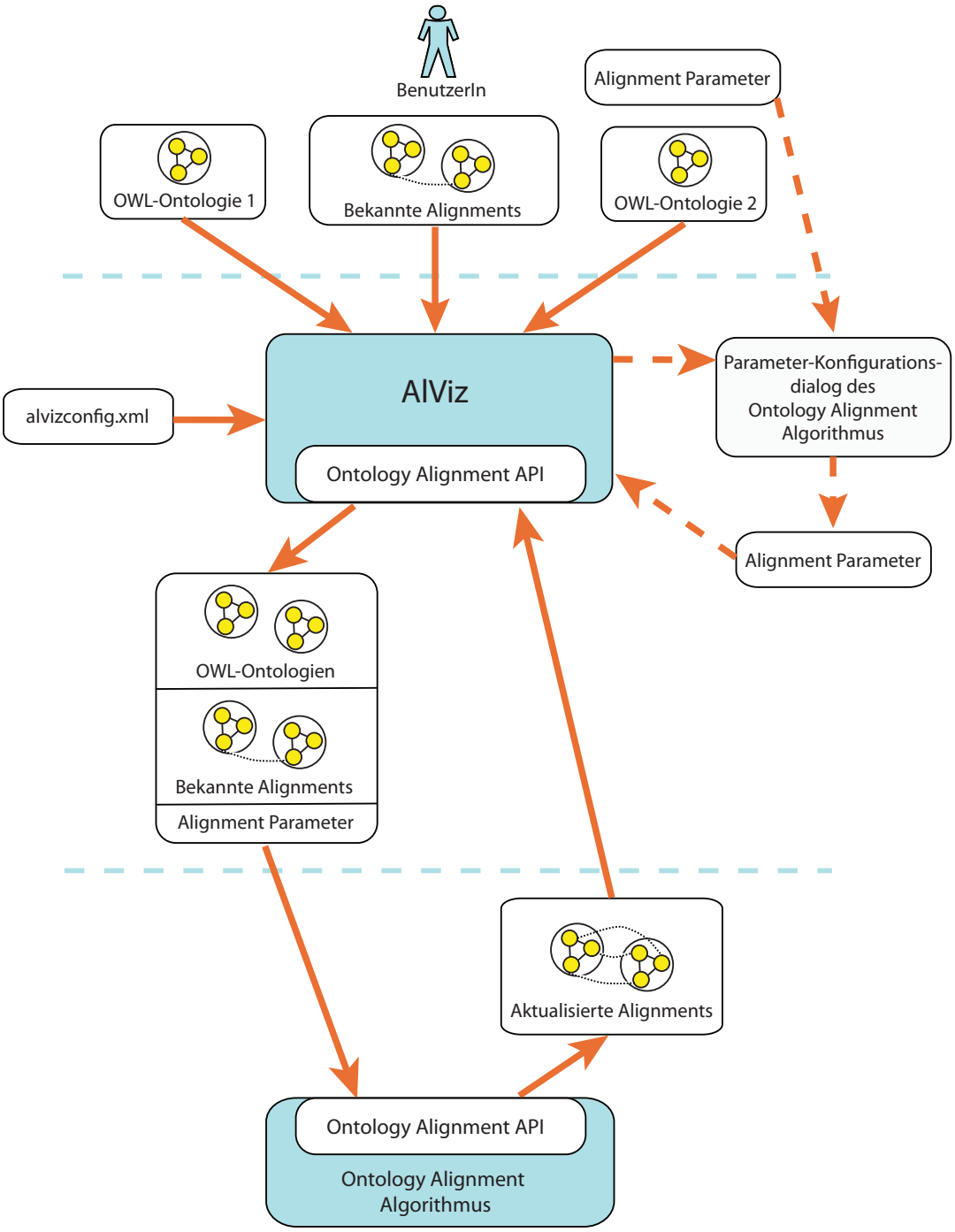

Abbildung 8.1: Architektur von AlViz

8.2.1 Ontology Alignment API

Die Ontology Alignment API definiert eine Schnittstelle zum Austausch bzw. zur Serialisierung von Alignments, ist in Java implementiert und ist zum freien Download[1] bzw. zur freien Verwendung verfügbar.
Zusätzlich zur Java-Interfacedefinition für Alignments, beinhaltet das Download-Paket eine Java-Standardimplementierung dieser Interfaces sowie auch Interfaces bzw. Klassen für die Verarbeitung von OWL-Ontologien. Letzteres wird nicht direkt der Ontology Alignment API zugeordnet, wodurch die API selbst völlig unabhängig von OWL ist.

Nachfolgend werden die definierten Interfaces der Ontology Alignment API mit ihren Eigenschaften basierend auf [Euzenat, 2006] aufgeschlüsselt.

8.2.1.1 Interfaces

Alignment

Das `Alignment`-Interface beschreibt die Menge an Alignments zwischen zwei Ontologien.

- **`xml`**: Gibt an, ob die Menge der Alignments und Eigenschaften gegenüber dem in [Euzenat, 2006] definierten DTD valide ist. Wenn dies zutrifft wird *true* angegeben, andernfalls *false*.
- **`level`**: Definiert das bei den Alignments verwendete Level. Gültige Werte sind *0*, *1*, *2**. Level 0 (Wert *0*) wird gesetzt, wenn jedes Alignment nur zwischen einzelnen Entitäten existiert. Bei Level 1 (Wert *1*) bestehen Alignments zwischen zwei Mengen von Entitäten. Level 2 (Wert *2**) unterstützt die Verwendung von Ausdrücken einer bestimmten Sprache (z.B. mit Regel aus logikorientierten Sprachen) für die Festlegung der Menge aus Ontologie 1 und der korrespondierenden Menge aus Ontologie 2.
- **`type`**: Beschreibt den Typ der Alignments, d.h. es sagt aus, ob es sich bei den Alignments um *one to one* (*11*), *one to many* (*1**), *many to many* (****) usw. Mappings handelt. Erlaubte Werte sind *11*, *1?*, *1+*, *1**, *?1*, *??*, *?+*, *?**, *+1*, *+?*, *++*, *+**, **1*, **?*, *?+*, ****.
- **`onto1`**: URI der ersten Ontologie.

[1] http://alignapi.gforge.inria.fr/

- **`onto2`**: URI der zweiten Ontologie.
- **`map`**: Menge an Alignments (`Cell`-Objekte) zwischen den Entitäten der Ontologien.

Cell

Dieses Interface repräsentiert ein Alignment zwischen Entitäten mit dem aggregierten Ähnlichkeitswert.

- **`rdf:resource`**: Id des Aligments.
- **`entity1`**: URI der Entitäten aus der ersten Ontologie.
- **`entity2`**: URI der Entitäten aus der zweiten Ontologie.
- **`measure`**: Die Wahrscheinlichkeit, dass die Relation zwischen den Entitäten vollkommen richtig ist. Dies entspricht dem aggregierten Ähnlichkeits- bzw. Konfidenzwert. Der Wertebereich geht von 0 bis 1.
- **`relation`**: Relation, die zwischen den Entitäten gegeben ist (`Relation`-Objekt).

Relation

Das `Relation`-Interface stellt die Relation zwischen Entitäten z.B. Gleichheit (=) dar.

Parameters

Das `Parameters`-Interface enthält alle Parameter als Key/Value-Paare für den entsprechenden Alignment Algorithmus.

AlignmentProcess

Das `AlignmentProcess`-Interface leitet vom Interface `Alignment` ab und muss dadurch die `align`-Methode zur Verfügung stellen, welche als Parameter eine Instanz einer Implementierung des `Alignment`-Interfaces und eine Instanz einer Implementierung des `Parameters`-Interfaces hat. Die `align`-Methode wird - wie der Name vermuten lässt - zur Generierung von Alignments aufgerufen. Dieses Interface muss von jedem Alignment Algorithmus implementiert werden.

Evaluator

Dieses Interface beschreibt den Vergleich von zwei Alignment-Mengen bzw. `Alignment`-Objekten.

- **align1**: Erste Alignment-Menge, welche oft die Referenz-Alignment-Menge repräsentiert.
- **align2**: Zweite Alignment-Menge.

8.2.1.2 Verwendung in AlViz

Ein ausschlaggebender Grund für die Verwendung der Ontology Alignment API in AlViz war, dass sie auch von der OAEI zur Evaluierung von Alignment Ergebnissen der eingereichten Algorithmen herangezogen wird. Somit sollten solche Algorithmen nach nur geringfügiger Änderung in AlViz integriert werden können. Allerdings unterstützt AlViz derzeit nur Alignments mit Level 0 und *one to one* Mappings.

Für die Integration eines Alignment Algorithmus in AlViz kommen die Interfaces `Alignment`, `Cell`, `Relation`, `Parameters`, `AlignmentProcess` zum Einsatz. Die Standardimplementierung dazu stellen die Klassen `BasicAlignment`, `BasicCell`, `BasicRelation` und `BasicParameters` dar, die auch im Download-Paket der Alignment API enthalten sind.
Mit dem `Cell`-Interface bzw. der dazugehörigen Klasse `BasicCell` lässt sich nur der aggregierte Ähnlichkeitswert zwischen Entitäten spezifizieren. Wie bei den Anforderungen in Kapitel 8.1 angeführt, soll es aber beim Austausch der Alignments zwischen AlViz und den Alignment Algorithmen möglich sein, regelbasierte Ähnlichkeitswerte, die zur Berechnung des aggregierten Ähnlichkeitwertes herangezogen werden, anzugeben. Aus diesem Grund wurden neue Klassen hinzugefügt:

- **`AlVizAlignment`**
  Subklasse von `BasicAlignment` mit folgenden zusätzlichen Eigenschaften:
  - **`alignmentAlgorithmName`**: Name des Ontology Alignment Algorithmus, der die Alignments ermittelt hat.
  - **`alignmentRules`**: Liste von `AlVizSimilarityRule`-Objekten, die beim entsprechenden Alignment Algorithmus insgesamt verfügbar sind.
- **`AlVizCell`**
  Subklasse von `BasicCell` mit nachstehender hinzugefügter Eigenschaft:
  - **`ruleSimilarities`**: Map, welche die regelbasierten Ähnlichkeitswerte enthält (Key: `AlVizSimilarityRule`-Objekt, Value: Gleitkommazahl mit dem Wertebereich von 0 bis 1.

AlVizSimilarityClass-Element	ordernr	name	identifier	color
EQUAL	0	equal	$=$	rot
SYNTACTICALLY_EQUAL	1	syntactically equal	$\simeq$	magenta
SIMILAR	2	similar	$\approx$	grün
BROADER_THAN	3	broader than	$>$	blau
NARROWER_THAN	4	narrower than	$<$	türkis
UNRELATED	5	unrelated	$\neq$	schwarz

Tabelle 8.1: Elemente der AlVizSimilarityClass-Enumeration

- **`AlVizSimilarityClass`**
  Enumeration, die alle verfügbaren Ähnlichkeitsklassen enthält. Zu jeder Ähnlichkeitsklasse kann eine Menge von Ähnlichkeitsregeln bzw. `AlVizSimilarityRule`-Objekten zugeordnet werden. Folgende Eigenschaften charakterisieren ein Element:
  - **`ordernr`**: Beeinflusst die Reihenfolge bzw. Sortierung der entsprechenden Ähnlichkeitsklasse in Bezug auf alle verfügbaren Ähnlichkeitsklassen.
  - **`name`**: Name der Ähnlichkeitsklasse.
  - **`identifier`**: Repräsentatives Zeichen für die Ähnlichkeitsklasse.
  - **`color`**: Farbe, durch die die Klasse in AlViz repräsentiert werden soll.

  Tabelle 8.1 zeigt die definierten Elemente mit den dazugehörigen Eigenschaftswerten. Beim Vergleich mit Tabelle 7.1 auf Seite 96 ist ersichtlich, dass die Ähnlichkeitsklasse *unknown* nicht in der `AlVizSimilarityClass`-Enumeration enthalten ist. Der Grund dafür ist, dass ein Alignment Algorithmus eine regelbasierte Ähnlichkeit, wo der genaue Ähnlichkeitstyp nicht bekannt ist, nicht verwenden kann, um damit einen aggregierten Ähnlichkeitswert zu berechnen. Weiters wird dadurch nun die Farbe schwarz für die Ähnlichkeitsklasse *unrelated* verwendet, da sich diese Farbe besser von den anderen Klassen unterscheidet.

- **`AlVizSimilarityRule`**
  Repräsentiert eine Ähnlichkeitsregel und weist nachstehende Eigenschaften auf:
  - **`id`**: Id der Ähnlichkeitsregel.
  - **`shortDescription`**: Kurzbeschreibung der Regel.

– **`similarityClass`**: Ähnlichkeitsklasse (`AlVizSimilarityClass`), zu dem die entsprechende Ähnlichkeitsregel zugeordnet ist.

Abbildung 8.2 zeigt zusammenfassend die Beziehungen aller verwendeten Interfaces und Klassen der Ontology Alignment API mit den beschriebenen Erweiterungen als UML-Klassendiagramm. Die Eigenschaften wurden dabei zu Gunsten der Übersichtlichkeit nicht angegeben. Außerdem wurden aus Platzgründen die Multiplizitäten nur an jenen Enden von Verbindungen angegeben, wo sie ungleich 1 sind.

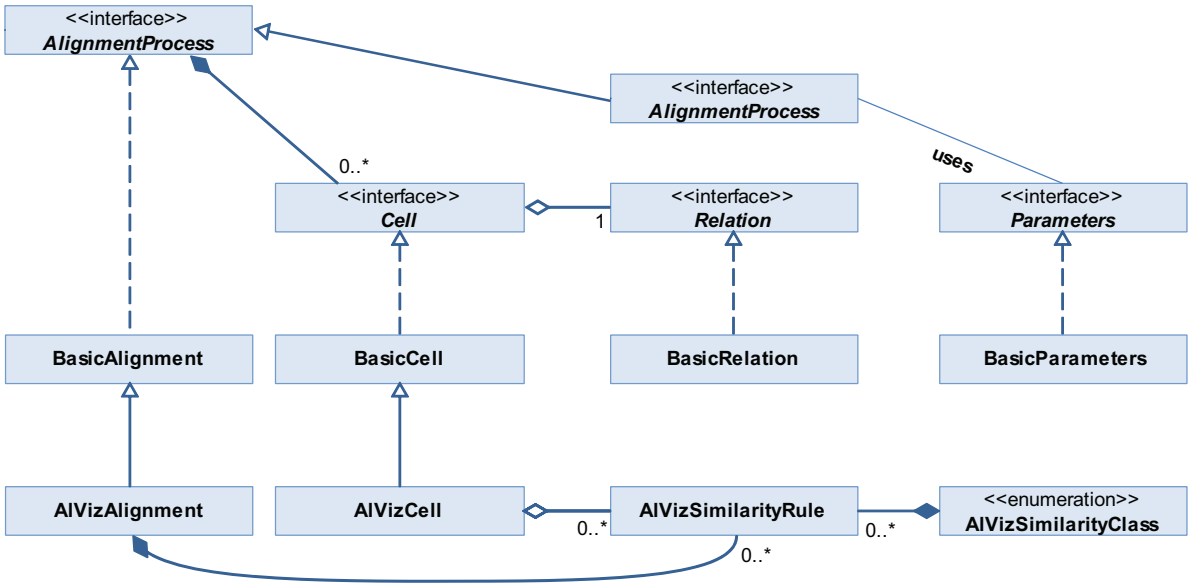

Abbildung 8.2: UML-Klassendiagramm der verwendeten Interfaces/Klassen aus der Ontology Alignment API und den Erweiterungen

Durch die neuen Klassen sowie die Enumeration kann der jeweilige Alignment Algorithmus neben dem aggregierten Ähnlichkeitswert auch seine internen Ähnlichkeiten mit den dazugehörigen Ähnlichkeitsregeln über die Ontology Alignment API an AlViz weitergeben. Beim Aufruf des nächsten Algorithmus übergibt AlViz einfach die erhaltenen Informationen vom vorigen Algorithmus mit einem `AlVizAlignment`-Objekt weiter. Kennt der neue Algorithmus auch die Erweiterung zur Ontology Alignment API, so kann er basierend auf dem `alignmentAlgorithmName` des `AlVizAlignment`-Objektes überprüfen, ob diese Alignments von ihm selbst stammen oder ob er den Algorithmus kennt bzw. die Ähnlichkeitsregeln des vorigen Algorithmus auf seine eigene abbilden kann.
Falls eines der Kriterien zutrifft, so kann er neben den aggregierten Ähnlichkeitswerten

auch die regelbasierten Ähnlichkeitswerte für eine Aktualisierung der bereits erhaltenen Alignments und Ermittlung neuer Alignments weiterverwenden. Falls keines der Kriterien zutrifft bzw. die API Erweiterungen dem Algorithmus nicht bekannt sind, so kann nur auf die aggregierten Ähnlichkeitswerte, welche mit dem vordefinierten Interface bzw. den Standardimplementierungen der Alignment API zur Verfügung stehen, zurückgegriffen werden.
Wenn noch kein Algorithmus zur Generierung von Alignments angestoßen wurde, dann ist es für den bzw. die BenutzerIn in AlViz nur möglich die aggregierten Ähnlichkeitswerte festzulegen. Nachdem ein Algorithmus-Durchlauf erfolgt ist, sind die Ähnlichkeitsregeln, sofern der jeweilige Algorithmus welche an AlViz weitergegeben hat, für den bzw. die BenutzerIn sichtbar. In AlViz sind immer nur die Regeln des zuletzt ausgeführten Alignment Algorithmus verwendbar.

Die hier beschriebenen Erweiterungen der Ontology Alignment API sind im Hauptverzeichnis von AlViz (z.B. /Applications/Protege_3.3.1/plugins/at.ac.tuwien.ifs.alviz) in der Java-Archivdatei `alvizAlignAPI.jar` abgelegt, welche standardmäßig von Protegé in den Classpath geladen wird. Die Java Package Bezeichnung für diese Klassen lautet `at.ac.tuwien.ifs.alviz.align.api`.

Wie bereits erwähnt, werden zum Download-Paket der Ontology Alignment API auch Interfaces für die Verarbeitung von OWL-Ontologien sowie eine Standardimplementierung dazu mitgeliefert. Aus diesem Grund wurden diese Interfaces in die Schnittstellenbeschreibung zwischen AlViz und den Ontology Alignment Algorithmen für den Austausch der OWL-Ontologiedaten aufgenommen. Für den Austausch bzw. die Verarbeitung der Ontologiedaten kommen die Interfaces `OWLOntology` (repräsentieren Ontologien), `OWLEntity` (repräsentieren Entitäten), `OWLClass` (repräsentieren Klassen bzw. Konzepte), `OWLDataProperty` (repräsentieren Daten-Eigenschaften), `OWLObjectProperty` (repräsentieren Objekt-Eigenschaften) und `OWLIndividual` (repräsentieren Instanzen) aus dem Package `org.semanticweb.owl.model` und die Klasse `OWLManager` aus dem Package `org.semanticweb.owl.util`, die das Laden und Konvertieren der Ontologiedaten aus einer OWL-Datei in die interne Objektstruktur übernimmt, zum Einsatz.

8.2.2 Einbindung von Ontology Alignment Algorithmen

Die Einbindung eines Ontology Alignment Algorithmus erfolgt über die XML-Konfigurationsdatei mit dem Namen `alvizconfig.xml`, welches sich im Hauptverzeichnis von AlViz befindet.

Den Grundaufbau der Konfigurationsdatei zeigt Listing 8.1. Für jeden Algorithmus, der dem bzw. der BenutzerIn zum Generieren von Alignments zur Auswahl stehen soll, muss ein `AlignmentAlgorithm`-Element definiert werden. Dabei wird mit dem Parameter `name` die Bezeichnung des Algorithmus festgelegt, die in AlViz repräsentativ für den Algorithmus angezeigt sowie auch der Eigenschaft `alignmentAlgorithmName` des `AlVizAlignment`-Objektes zugewiesen wird. Alle notwendigen Java-Archivdateien für einen Algorithmus müssen in einem Ordner im Verzeichnis `plugin`, das sich im AlViz-Hauptverzeichnis befindet, abgelegt werden. D.h. alle Java .class-Dateien müssen in Archivdateien zusammengefasst werden. Mit dem Parameter `plugindir` wird der Name des Verzeichnisses bestimmt, in dem die entsprechenden Algorithmus-Dateien gespeichert sind. Alle Java-Archivdateien, die sich in diesem Verzeichnis befinden, werden von AlViz zum Classpath hinzugefügt. Der dritte Parameter (`classname`) des `AlignmentAlgorithm`-Elements spezifiziert jene Java-Klasse, die das Interface `AlignmentProcess` der Ontology Alignment API implementiert und somit den Aufruf der Methode `align` mit den Interface-Parametern `Alignment` und `Parameters` ermöglicht.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <AlVizConfig>
3    <AlignmentAlgorithm name="<alignment algorithm name>" plugindir="<plugin directory>"
         classname="<class which implements org.semanticweb.owl.align.AlignmentProcess>">
4      <MainJars>
5        <Jar name="<jar-file of which AlViz must load every class>"/>
6      </MainJars>
7
8      <!-- optional -->
9      <ConfigDialog classname="<class which extends
           at.ac.tuwien.ifs.alviz.align.config.AlignAlgorithmConfigDlg and will be shown
           when the user wants to edit parameters of the specific algorithm>" />
10   </AlignmentAlgorithm>
11 </AlVizConfig>
```

Listing 8.1: Aufbau der Konfigurationsdatei `alvizconfig.xml` zur Einbindung eines Ontology Alignment Algorithmus in AlViz

Innerhalb des `MainJars`-Elements können jene Java-Archivdateien mit den `Jar`-Element definiert werden, aus denen jede Java-Klasse nacheinander geladen werden muss. Somit muss mindestens ein Eintrag für jene Java-Archivdatei angegeben werden, in der sich auch die Klasse befindet, die beim vorhin erwähnten Parameter `classname` bestimmt wurde.

```java
package at.ac.tuwien.ifs.alviz.align.config;

import java.awt.Dialog;

import javax.swing.JDialog;

import org.semanticweb.owl.align.Parameters;

public abstract class AlignAlgorithmConfigDlg extends JDialog{

  private static final long serialVersionUID = 1L;

  /* in a subclass of this class there must be a constructor with Dialog-parameter! */
  public AlignAlgorithmConfigDlg( Dialog dlg, String title ){
    /* this dialog is always modal */
    super( dlg, title, true);
  }

  /* will be called before the dialog is shown to the user */
  public abstract void setParameters( Parameters parameters );
  /* will be called after the dialog was closed by the user */
  public abstract Parameters getParameters();
}
```

Listing 8.2: AlignAlgorithmConfigDlg.java - Schnittstellendefinition zwischen AlViz und dem Parameter-Konfigurationsdialog eines Ontology Alignment Algorithmus

Optional kann schließlich noch mit dem `ConfigDialog`-Element bzw. mit dem dazugehörigen Parameter `classname` die Konfigurationsdialog-Klasse zum entsprechenden Algorithmus festgelegt werden. Diese Dialog-Klasse muss von der Klasse `AlignAlgorithmConfigDlg` (siehe Listing 8.2) aus dem Package `at.ac.tuwien.ifs.alviz.align.config`, die von Protegé standardmäßig in den Classpath geladen wird, ableiten. Diese Klasse muss durch die Ableitung eine Set- bzw. Get-Methode zum Setzen bzw. Abfragen von Parametern zur Verfügung stellen. Die Parameter werden beim Aufruf der `align`-Methode von AlViz an den Alignment Algorithmus übermittelt. Außerdem setzt AlViz voraus, dass ein Konstruktor mit einem `java.awt.Dialog`-Parameter existiert. Die Klasse `AlignAlgorithmConfigDlg` ist in die Java-Archivdatei `alvizAlignAlgoConfigDlg.jar` eingebettet, die genauso wie die Konfigurationsdatei selbst im Hauptverzeichnis von AlViz zu finden ist.
Sofern die angegebene Konfigurationsdialog-Klasse nicht in der gleichen Java-Archivdatei ist, wie jene, die das Interface `AlignmentProcess` implementiert, muss ein weiteres

`Jar`-Element innerhalb von `MainJars` definiert werden.

Die Instanziierung der konfigurierten Klassen bzw. der Aufruf der notwendigen Methoden erfolgt in AlViz mit Hilfe der Java Reflection API[2].

8.2.3 Ein-/Ausgabeformat

Listing 8.3 zeigt den Aufbau einer von AlViz gespeicherten XML-Datei exemplarisch nur mit einem Alignment, welches vom der Ontology Alignment Algorithmus FOAM ermittelt wurde.

Im Vergleich zu Listing 7.1 auf Seite 7.1 ist ersichtlich, dass nur das Element `AlignAlgorithmInfo` hinzugefügt wurde. Mit dem Parameter `algorithmName` dieses Elements wird angegeben, welcher Algorithmus die Alignments generiert bzw. aktualisiert hat. Der Parameterwert wird entsprechend der Konfiguration in `alvizconfig.xml` gesetzt bzw. beim nächsten Aufruf eines Alignment Algorithmus als Eigenschaftswert für `alignmentAlgorithmName` des `AlVizAlignment`-Objektes verwendet.

Die Subelemente von `AlignAlgorithmInfo` bestimmen, welche Ähnlichkeitsregeln vom entsprechenden Algorithmus für die Ermittlung des aggregierten Ähnlichkeitswertes verwendet werden und welcher Ähnlichkeitsklasse der `AlVizSimilarityClass`-Enumeration aus der Ontology Alignment API Erweiterung sie zugewiesen sind. Mit dem Parameterwert von `name` des `SimilarityClass`-Tags erfolgt die Zuordnung zur richtigen Klasse. Dabei muss der Enumeration-Elementname verwendet werden und nicht der Eigenschaftswert von `name` des jeweiligen Elements.
Die Parameterwerte `id` und `shortDescription` des `SimilarityRule`-Elements werden bei den gleichnamigen Eigenschaften der `AlVizSimilarityRule`-Objektes gesetzt.
Wenn der gestartete Alignment Algorithmus die zugrundeliegenden Ähnlichkeitsregeln an AlViz nicht übermittelt hat, dann enthält das `AlignAlgorithmInfo`-Element keine Subelemente. Weiters beinhalten die `CNode`-Tags keine `Similarity`-Elemente.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <AlViz>
3   <AlignAlgorithmInfo algorithmName="FOAM">
4     <SimilarityClass name="EQUAL">
5       <SimilarityRule id="1" shortDescription="Equal URI" />
6       <SimilarityRule id="7" shortDescription="Equal class member individuals" />
```

[2] http://java.sun.com/docs/books/tutorial/reflect/

```
7     </SimilarityClass>
8     <SimilarityClass name="SYNTACTICALLY_EQUAL">
9       <SimilarityRule id="0" shortDescription="Syntactic similarity" />
10    </SimilarityClass>
11    <SimilarityClass name="SIMILAR">
12      <SimilarityRule id="2" shortDescription="Similar superclasses" />
13      <SimilarityRule id="3" shortDescription="Similar subclasses" />
14      <SimilarityRule id="4" shortDescription="Similar class data properties" />
15      <SimilarityRule id="5" shortDescription="Similar class object properties
            from..." />
16      <SimilarityRule id="6" shortDescription="Similar class object properties to..."
            />
17      <SimilarityRule id="10" shortDescription="Similar data property domain" />
18      <SimilarityRule id="11" shortDescription="Similar super data properties" />
19      <SimilarityRule id="12" shortDescription="Similar sub data properties" />
20      <SimilarityRule id="13" shortDescription="Similar data property members" />
21      <SimilarityRule id="14" shortDescription="Similar object property domain" />
22      <SimilarityRule id="15" shortDescription="Similar object property range" />
23      <SimilarityRule id="17" shortDescription="Similar object property sub" />
24      <SimilarityRule id="16" shortDescription="Similar object property super" />
25      <SimilarityRule id="19" shortDescription="Similar individual member of class" />
26      <SimilarityRule id="18" shortDescription="Similar object property members" />
27      <SimilarityRule id="21" shortDescription="Similar individual object property
            members from..." />
28      <SimilarityRule id="20" shortDescription="Similar individual data properties" />
29      <SimilarityRule id="22" shortDescription="Similar individual object property
            members to..." />
30    </SimilarityClass>
31    <SimilarityClass name="BROADER_THAN">
32      <SimilarityRule id="9" shortDescription="Broader than" />
33    </SimilarityClass>
34    <SimilarityClass name="NARROWER_THAN">
35      <SimilarityRule id="8" shortDescription="Narrower than" />
36    </SimilarityClass>
37  </AlignAlgorithmInfo>
38  <Nodes>
39    <Node uri="http://www.example.com/ontologies/auto/autoA.owl#Speed">
40        <Entity label="Speed" type="C"/>
41        <CNode uri="http://www.example.com/ontologies/auto/autoB.owl#Speed">
42          <Entity label="Speed" type="C"/>
43          <Similarity rule="0" value="1.0" />
44          <Similarity rule="6" value="1.0" />
45          <SimilarityInfo Correctvalue="?">
46            <Confidence value="1.0" />
47          </SimilarityInfo>
```

```
48          </CNode>
49        </Node>
50    </Nodes>
51 </AlViz>
```

Listing 8.3: AlViz-XML mit einem von FOAM ermittelten Alignment

Wurde noch kein Ontology Alignment Algorithmus gestartet und wurden z.B. nur dem bzw. der BenutzerIn bekannte Alignments definiert, so wird der Wert des Parameters `algorithmName` auf `AlViz` gesetzt. Außerdem gibt es dadurch auch keine `AlVizSimilarityClass`-, `AlVizSimilarityRule`- und `Similarity`-Elemente (siehe Listing 8.4).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <AlViz>
3    <AlignAlgorithmInfo algorithmName="AlViz">
4    </AlignAlgorithmInfo>
5    <Nodes>
6        <Node uri="http://www.example.com/ontologies/auto/autoA.owl#Speed">
7          <Entity label="Speed" type="C"/>
8          <CNode uri="http://www.example.com/ontologies/auto/autoB.owl#Speed">
9            <Entity label="Speed" type="C"/>
10           <SimilarityInfo Correctvalue="?">
11             <Confidence value="1.0" />
12           </SimilarityInfo>
13         </CNode>
14       </Node>
15   </Nodes>
16 </AlViz>
```

Listing 8.4: AlViz-XML mit einem nur vom Benutzer bzw. von der Benutzerin definierten Alignment

Kapitel 9

Praktischer Teil: Implementierung & Evaluierung

Basierend auf der Analyse und dem Design aus dem vorigen Kapitel wird in diesem Abschnitt die konkrete Implementierung sowie eine informale Evaluierung von AlViz vor allem mit Fokus auf die umgesetzten Erweiterungen erläutert.

9.1 Implementierung

Die Erweiterungen zu AlViz wurden, genauso wie die Ausgangsversion von AlViz, für Protegé 3.3.1 implementiert. Zu den externen Bibliotheken (*Prefuse* für Graphendarstellung und -manipluation, *JOX* für das Lesen von XML-Daten, *Colt* für mathematische Algorithmen, *Simple Log* für Logging), die bereits vor den hier beschriebenen Erweiterungen verwendet wurden, kam nur die bereits erwähnte Ontology Alignment API mit deren Abhängigkeiten hinzu. Als Entwicklungsumgebung wurde Eclipse[1] in Version 3.4.2 eingesetzt.

Die folgenden Unterkapitel erläutern nun den Aufbau der Implementierung durch die Struktur- und Klassenbeschreibung und die hinzugekommenen bzw. angepassten graphischen Benutzeroberflächenelementen. Schließlich wird anhand des Ontology Alignment Algorithmus FOAM gezeigt, wie ein Algorithmus konkret in AlViz integriert werden kann.

[1] http://www.eclipse.org/

9.1.1 Aufbau der Implementierung

Abbildung 9.1 zeigt das Klassendiagramm jener Klassen, die durch die Erweiterungen von AlViz verändert wurden oder hinzukamen. Die Klassen wurden dabei nach Java-Packages gruppiert, strichliert umrahmt und mit Notizen des bzw. der zugehörigen Java-Packages/Java-Packagegruppe versehen. Besteht eine Beziehung zwischen allen Klassen einer Gruppierung mit allen Klassen einer anderen Gruppierung, so wurden zur Erhaltung der Übersichtlichkeit nur Verbindungen zwischen den Rahmen eingezeichnet (z.B. Assoziation zwischen dem Package `at.ac.tuwien.ifs.alviz.align.data` und dem Package `at.ac.tuwien.ifs.alviz.align.io`). Stehen zwei Klassen unterschiedlicher Gruppierungen in Relation, so wurde die Verbindung zwischen den beiden Klassen dargestellt (z.B. Assoziation zwischen der Klasse `AlVizGraph` und der Klasse `AlViz`). Die Verbindung zwischen einer Menge von Klassen einer Gruppierung `A` zu einer speziellen Klasse einer anderen Gruppierung `B` wurde so visualisiert, dass von den Klassen in `A` die Linien an den Rahmen der eigenen Gruppierung geführt und schließlich eine Linie von dem Rahmen zur Klasse aus `B` eingezeichnet wurde (z.B. Assoziationen zwischen fünf Klassen aus dem Package `at.ac.tuwien.ifs.alviz.gui` bzw. `at.ac.tuwien.ifs.alviz.align.config` und der Klasse `AlViz` aus dem Package `at.ac.tuwien.ifs.alviz`).
Außerdem wurden aus Platzgründen die Multiplizitäten nur an jenen Enden der Verbindungen angegeben, wo sie ungleich 1 sind.

In den nachfolgenden Unterabschnitten werden die einzelnen Klassen des Klassendiagramms, nach Java-Package strukturiert, kurz beschrieben.

9.1.1.1 at.ac.tuwien.ifs.alviz

Dieses Package beinhalted die Hauptklasse für die Einbettung von AlViz in Protegé sowie die Klasse, welche das Panel für eine Ontologie in der GUI repräsentiert.

- `AlViz`: Stellt den Dreh- und Angelpunkt von AlViz dar und ist somit die Hauptklasse für die Einbettung von AlViz in Protegé. Die Klasse verteilt alle über das GUI getätigen Benutzerinteraktionen an die entsprechenden Subkomponenten.
- `AlVizClsesPanel`: Eine Instanz dieser Klasse repräsentiert das Panel für eine Ontologie in der GUI, welche den JTree und den Small World Graph enthält.

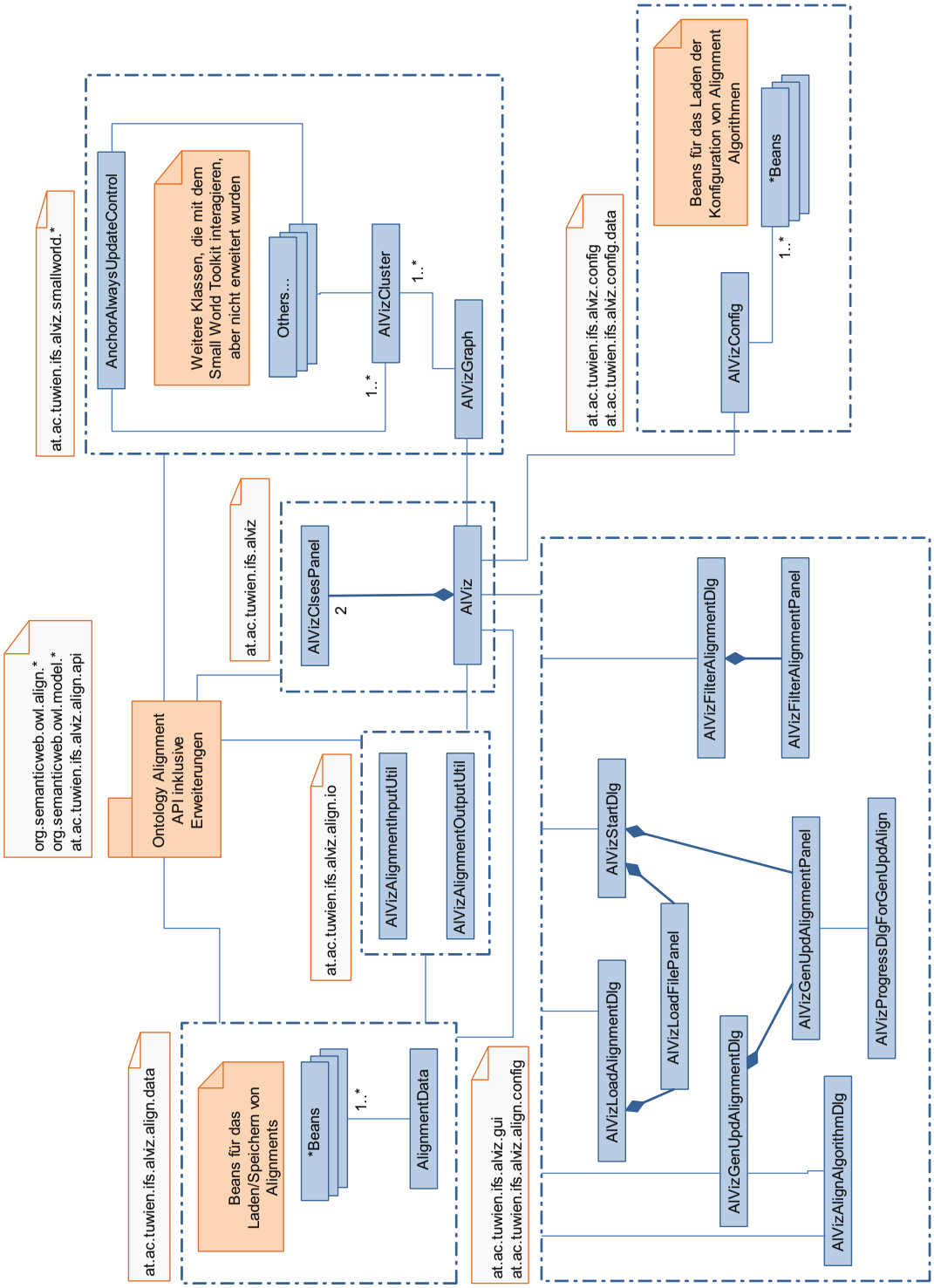

Abbildung 9.1: Klassendiagramm von AlViz

9.1.1.2 at.ac.tuwien.ifs.alviz.align.api

Jene Klassen bzw. Enumeration, die die Erweiterung der Ontology Alignment API (`AlVizAlignment`, `AlVizCell`, `AlVizSimilarityClass` und `AlVizSimilarityRule` - siehe Kapitel 8.2.1.2) umsetzen, sind in diesem Package angesiedelt. Nach der Installation von AlViz sind die dazugehörigen .class-Dateien im Hauptverzeichnis von AlViz als Java-Archivdatei (`alvizAlignAPI.jar`) abgelegt. Somit kann diese Java-Archivdatei für die Einbindung in Ontology Alignment Algorithmen verwendet werden.

9.1.1.3 at.ac.tuwien.ifs.alviz.align.config

In diesem Package ist nur die Klasse `AlignAlgorithmConfigDlg` zu finden. Jene Klassen, die als Dialog für die Parameter-Konfiguration eines Alignment Algorithmus in AlViz fungieren sollen, müssen von der Klasse `AlignAlgorithmConfigDlg` ableiten. Die Implementierung dazu wurde bereits in Listing 8.2 in Kapitel 8.2.2 gezeigt. Die Java-Archivdatei `alvizAlignAlgoConfigDlg.jar`, welches sich nach der AlViz-Installation im Hauptverzeichnis befindet, enthält die dazughörige .class-Datei.

9.1.1.4 at.ac.tuwien.ifs.alviz.align.data

- `AlignmentData`: Verwendet die Dateninhalte der Beans (siehe nächster Aufzählungspunkt), um interne Datenstrukturen zu befüllen bzw. schnelleres Auffinden von Alignments, Ähnlichkeitsregeln/-klassen zu ermöglichen. Die Klasse repräsentiert die in AlViz verfügbaren Alignments, wodurch dieses Package in Relation zur Klasse `AlViz`, den Klassen des Packages `at.ac.tuwien.ifs.alviz.align.io` und der Ontology Alignment API steht. In der ursprünglichen Version von AlViz war der Name dieser Klasse `FoamData` und war im Package `at.ac.tuwien.ifs.alviz.smallworld.types` zu finden. Da die Verwendung von AlViz nun nicht mehr auf Daten von FOAM eingeschränkt ist, wurde diese Klasse umbenannt und verschoben.
- `*Beans`: Java-Beans, die für das Laden von Alignments nach dem definierten Format (siehe Kapitel 8.2.3) für die Verwendung von JOX notwendig sind. In der Ausgangsversion von AlViz waren diese Klassen im Package `at.ac.tuwien.ifs.alviz.smallworld.data` angesiedelt, welches im Laufe der Implementierung entfernt wurde. Der Grund für die Verschiebung war die Erzielung einer besseren Strukturierung bzw. eines aussagekräftigeren Packagenamens.

9.1.1.5 at.ac.tuwien.ifs.alviz.align.io

In diesem Package befinden sich Hilfsprozeduren, die das Laden und Speichern von Alignments von bzw. in eine(r) XML-Datei anstoßen. Weiters übernehmen sie die Konvertierung von Alignmentdaten, die von einem eingebundenen Alignment Algorithmus stammen oder an einen ausgewählten Algorithmus weitergegeben werden.

- `AlVizAlignmentInputUtil`: Die Klasse enthält Hilfsprozeduren, die das Laden von Alignments aus einer XML-Datei sowie die Konvertierung von Alignmentdaten, die von einem eingebundenen Alignment Algorithmus entgegengenommen wurden, entsprechend den internen Objekten anstoßen bzw. übernehmen.
- `AlVizAlignmentOutputUtil`: Hilfsprozeduren für das Speichern von Alignments in einer XML-Datei sowie für notwendige Konvertierungen vor der Weitergabe von Alignmentdaten an einen ausgewählten Algorithmus sind in dieser Klasse zu finden.

9.1.1.6 at.ac.tuwien.ifs.alviz.config

Die darin enthaltende Klasse `AlVizConfig` validiert die Datei `alvizconfig.xml`, d.h. ob die angegebenen Java-Archivdateien und Klassen existieren bzw. ob diese Java-Klassen die notwendigen Interfaces/Klassenimplementierung erweitern.

9.1.1.7 at.ac.tuwien.ifs.alviz.config.data

Dieses Package enthält die Java-Beans für das Laden der Datei `alvizconfig.xml`, die für die Verwendung von JOX gebraucht werden. Die Daten der Beans werden von der Klasse `AlVizConfig` verwendet.

9.1.1.8 at.ac.tuwien.ifs.alviz.gui

Die Implementierung der Java-Swing-Dialoge sowie -Panels sind in diesem Package zu finden.

- `AlVizFilterAlignmentDlg`: Will der bzw. die BenutzerIn nach bestimmten Ähnlichkeitswerten filtern, so wird dieser Dialog geöffnet. Er enthält das Panel `AlViz-FilterAlignmentPanel`.

- `AlVizFilterAlignmentPanel`: Dieses Panel enthält die Steuerelemente für das Setzen eines Filters bezüglich Ähnlichkeitswerten.
- `AlVizGenUpdAlignmentDlg`: Nachdem die Visualisierungen der beiden Ontologien im AlViz-Tab von Protegé angezeigt werden, kann der bzw. die BenutzerIn die Aktualisierung von Alignments anstoßen. Daraufin erscheint dieser Dialog, der das Panel `AlVizGenUpdAlignmentDlg` beinhaltet.
- `AlVizGenUpdAlignmentPanel`: Die konfigurierten Alignment Algorithmen werden in einer Auswahlbox in diesem Dialog angezeigt. Weiters kann über die Steuerelemente dieses Panels der angegebene Dialog zum Konfigurieren von Parameter eines Algorithmus geöffnet werden.
- `AlVizLoadAlignmentDlg`: Dieser Dialog enthält das Panel `AlVizLoadFilePanel` und wird vom Benutzer bzw. von der Benutzerin geöffnet, wenn dieser bzw. diese die Alignments nachträglich von einer XML-Datei laden will.
- `AlVizLoadFilePanel`: Dieses Panel enthält Steuerelemente für die Auswahl einer Datei.
- `AlVizProgressDlgForGenUpdAlign`: Dieser Dialog enthält eine Progressbar und soll dem bzw. der BenutzerIn zeigen, dass gerade der ausgewählte Ontology Alignment Algorithmus Alignments aktualisiert bzw. generiert.
- `AlVizStartDlg`: Nach der Auswahl des Protegé-Projektes öffnet sich dieser Dialog, der zweimal das Panel `AlVizLoadFilePanel` für das Laden der zweiten OWL-Datei und möglicherweise einer Alignment XML-Datei sowie das Panel `AlVizGenUpdAlignmentPanel` enthält, damit der bzw. die BenutzerIn gleich zu Beginn Alignments mit einem gewünschten Algorithmus durchführen kann.

9.1.1.9 at.ac.tuwien.ifs.alviz.smallworld.*

Jene Klassen, die mit der Darstellung sowie Benutzerinteraktion des Small World Graphen in Verbindung stehen, sind auf diese Packages verteilt. Für die in dieser Arbeit beschriebenen Erweiterungen von AlViz, waren nur Änderungen an folgenden Klassen notwendig:

- `at.ac.tuwien.ifs.alviz.smallworld.types`

  - `AlVizCluster`: Diese Klasse repräsentiert abhängig vom festgelegten Zoomfaktor ein Konzept bzw. eine Menge von Konzepten einer Ontologie im Small World Graph. Auf Grund der Unterscheidung zwischen aggregierten und regelbasierten Ähnlichkeitswerten in AlViz und der dazugehörigen Farbvisualisierungen musste die Klasse angepasst werden.
  - `AlVizGraph`: Die Instanziierungen der Klasse `AlVizCluster` für den Aufbau des Small World Graphen erfolgt in dieser Klasse. Weiters werden in dieser Klasse, die in Relation stehenden `AlVizCluster`-Objekte zusammengesucht. Auch diese Klasse musste auf Grund der Unterscheidung zwischen aggregierten und regelbasierten Ähnlichkeitswerten erweitert werden. In der Ausgangsversion von AlViz war der Name dieser Klasse `AlVizReader`. Dieser Name wurde verändert, da die usprüngliche Version keine Funktionalität enthielt, die man üblicherweise mit einer `Reader`-Klasse in Verbindung bringt.
  - `AlVizWriter`: Diese Klasse wurde aus der usprünglichen AlViz-Version entfernt, da zumindest ein Teil der enthaltenen Funktionalität nicht einer `Writer`-Klasse zugeordnet wird. Der Codeteil für das Laden der Alignments aus einer XML-Datei wurde in die Klasse `AlVizAlignmentOutputUtil` eingefügt und jener Codeteil für das Synchronisieren der Ähnlichkeitsinformationen zwischen `AlVizCluster`-Objekten und dem `AlignmentData`-Objekt, wurde in die Klasse `AlViz` verschoben.

- `at.ac.tuwien.ifs.alviz.smallworld.control`
  - `AnchorAlwaysUpdateControl`: Beim Setzen der Ähnlichkeitswerte durch den bzw. die BenutzerIn wird diese Klasse aufgerufen, die sich um das Öffnen und Befüllen des Dialogs mit den richtigen Ähnlichkeitswerten/-regeln kümmert. Nach der Bestätigung des Dialogs werden die Eigenschaften des bzw. der jeweiligen `AlVizCluster`-Objekte(s) aktualisiert. Durch die hinzugekommene Unterscheidung an Ähnlichkeitswerten wurde auch hier eine Codeanpassung vorgenommen.

9.1.2 Erweiterungen der graphischen Benutzeroberfläche

Im Zuge der Erweiterungen von AlViz wurde auch die graphische Benutzeroberfläche erweitert und teilweise verändert. Dieses Kapitel beschreibt die entsprechenden Dialoge und Komponenten.

9.1.2.1 Dialoge und Bedienleiste

Nachdem der bzw. die BenutzerIn mit dem Protegé-Menü bzw. dem Startdialog von Protegé das Projekt ausgewählt hat und daraufhin eine OWL-Datei geladen wurde, erscheint der Startdialog von AlViz, der in Abbildung 9.2 ersichtlich ist.
Mit Hilfe der *Browse*-Schaltfläche im Bereich *Second OWL file* kann der bzw. die BenutzerIn die zweite OWL-Datei mit einem Dateiauswahldialog bestimmen. Außerdem besteht auch die Möglichkeit die Datei inklusive Pfad in die Textzeile links von der Schaltfläche einzugeben. Damit AlViz bzw. die Visualisierungen vollständig initialisiert werden können, ist die Festlegung der zweiten OWL-Datei klarerweise verpflichtend.
Im *Alignment*-Bereich kann gewählt werden, ob die Alignments

- von einer XML-Datei, die dem bereits in Kapitel 8.2.3 besprochenen Format entsprechen muss, geladen werden sollen oder
- vor der Visualisierung der Ontologien mit einem der konfigurierten Ontology Alignment Algorithmen generiert werden sollen oder
- zunächst AlViz und die Visualisierung der Ontologien initialisiert werden soll und danach erst vom Benutzer bzw. von der Benutzerin Alignments manuell festgelegt, geladen oder generiert werden sollen.

Beim Hin- und Herschalten zwischen den Optionen, werden die dazugehörigen Steuerelemente entweder aktiviert oder deaktiviert. Damit wird noch besser sichtbar welche Option aktuell ausgewählt bzw. welche Eingabe noch erforderlich ist.

Wenn die zweite Option markiert wird, so kann der bzw. die BenutzerIn einen der eingebundenen Alignment Algorithmen auswählen und mit der Schaltfläche *Edit parameter configuration* den dazu entsprechenden Parameter-Konfigurationsdialog - sofern dieser in der Konfigurationsdatei `alvizconfig.xml` angegeben wurde - öffnen und die Parameter einstellen.

Nach der Bestätigung dieses Dialogs werden bei der Wahl der ersten Option die Alignments aus der angegebenen Datei geladen und AlViz bzw. die Visualisierung initialisiert. Bei der Wahl der zweiten Option wird dem bzw. der BenutzerIn der Dialog in Abbildung 9.3 gezeigt, der mit der enthaltenen Progressbar auf die momentane Generierung der Alignments durch den Algorithmus hinweist. Im Titel des Dialoges wird der Name des ausgewählten Alignment Algorithmus angezeigt, was im konkreten Fall FOAM ist. Bei der Wahl der dritten Option werden direkt AlViz sowie die Ontologievisualisierungen geladen.

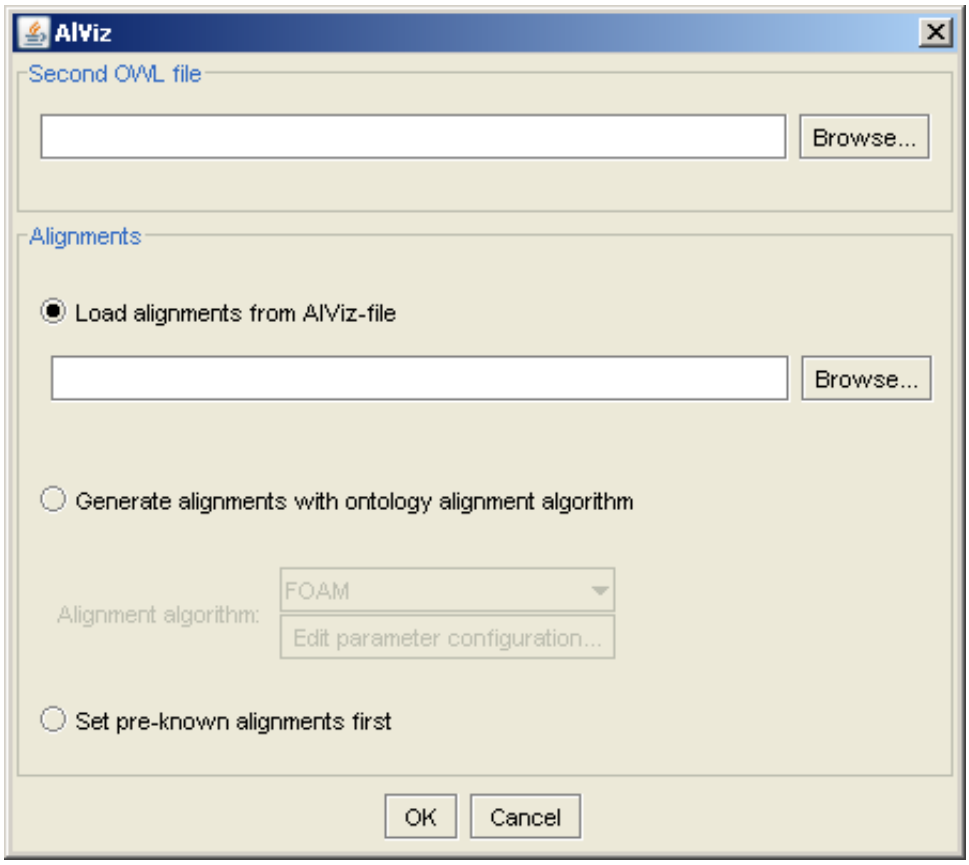

Abbildung 9.2: Startdialog von AlViz

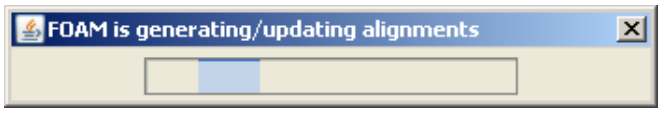

Abbildung 9.3: Progress-Dialog während der Generierung von Alignments

Abbildung 9.6 zeigt AlViz nach der Initialisierung der Ontologie-Visualisierungen. Im Vergleich zur usprünglichen AlViz-Version wurden in der Bedienleiste Schaltflächen für das Arbeiten mit Alignments hinzugefügt. Alle diese Schaltflächen sind in Abbildung 9.4 dargestellt.

Abbildung 9.4: Schaltflächen der Bedienleiste für das Arbeiten mit Alignments

Mit der Schaltfläche *Load...* können Alignments von einer XML-Datei geladen werden, wodurch die derzeitigen Alignments in AlViz überschrieben werden. Der dazugehörige Dialog ist in Abbildung 9.5 ersichtlich. Der Dialog enthält das gleiche Panel, welches bereits beim Startdialog von AlViz verwendet wurde.

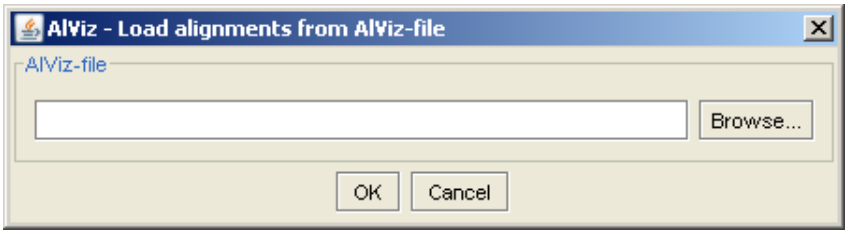

Abbildung 9.5: Dialog zum Laden von Alignments über die Bedienleiste

Abbildung 9.7 zeigt den Dialog zum Aktualisieren bzw. Generieren von Alignments, der über die Schaltfläche *Generate/Update...* geöffnet werden kann. Die Alignments die von einem ausgewählten Alignment Algorithmus an AlViz nach der Aktualisierung zurückgegeben werden, überschreiben die vorhandenen Alignments. Auch hier wurde das Panel aus dem Startdialog wiederverwendet bzw. wird während der Generierung der Alignments durch den Algorithmus wieder der Progress-Dialog aus Abbildung 9.3 angezeigt.

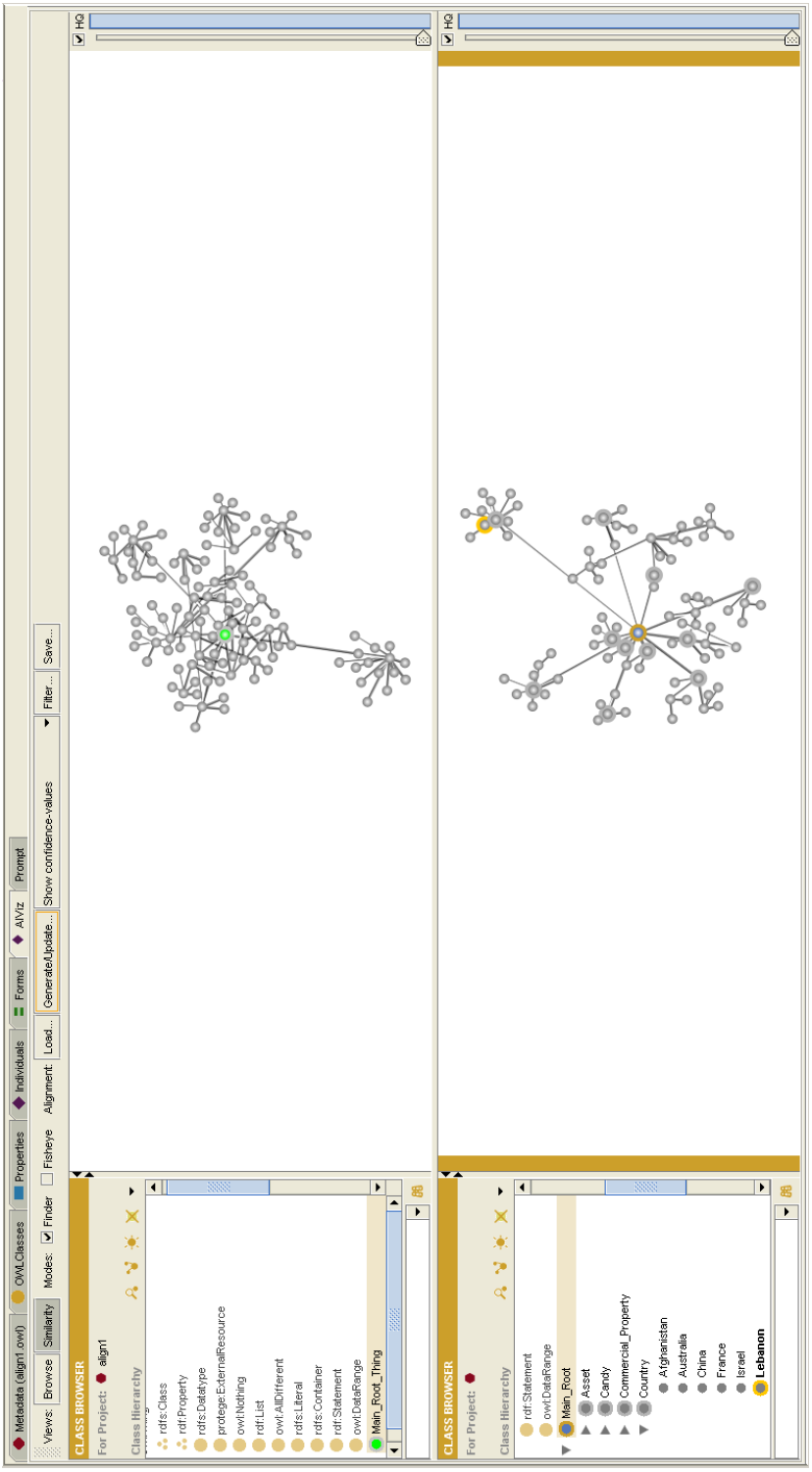

Abbildung 9.6: AIViz nach der Initialisierung der Ontologie-Visualisierungen

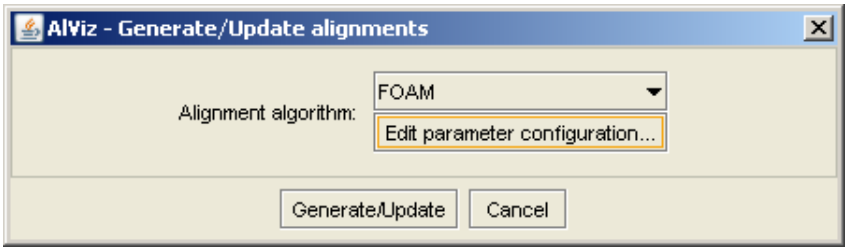

Abbildung 9.7: Dialog zum Laden von Alignments über die Bedienleiste

Mit der Auswahlbox in der Bedienleiste kann der bzw. die BenutzerIn festlegen, welche Ähnlichkeitswerte in der passiven Ontologie-Visualisierung farblich hervorgehoben werden sollen. Es kann entweder *Show confidence-values*, wonach nur die aggregierten Ähnlichkeitswerte visualisiert werden, oder *Show rule-based-similarity-values*, wodurch die höchste Ähnlichkeit aus der Menge der regelbasierten Ähnlichkeiten zwischen dem markierten Knoten aus der aktiven Ontologie und den entsprechenden Konzepten der passiven Ontologie farblich angezeigt werden, gewählt werden (siehe Abbildung 9.8). Sind keine regelbasierten Ähnlichkeiten vorhanden, da z.B. noch kein Alignment Algorithmus gestartet wurde oder der Algorithmus keine Ähnlichkeitsregel an AlViz übergeben hat, so kann nur der Eintrag *Show confidence-values* gewählt werden. Beim Versuch auch den anderen Eintrag zu wählen wird eine entsprechende Hinweismeldung ausgegeben.

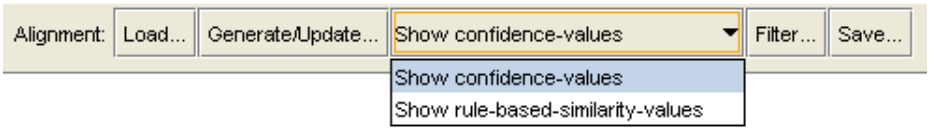

Abbildung 9.8: Auswahlmöglichkeit für unterschiedliche Ähnlichkeitsansichten

Über die Schaltfläche *Filter...* kann der bzw. die BenutzerIn den Dialog aus Abbildung 9.9 öffnen. Damit kann festgelegt werden, welche Ähnlichkeitswerte angezeigt werden sollen. Der mögliche Wertebereich liegt dabei zwischen 0.0 und 1.0 und wird standardmäßig angenommen. Im Dialog wird mit der Schaltfläche *Apply filter* der eingestellte Wertebereich bestätigt und die Visualisierungen aktualisiert. Durch einen Klick auf die Schaltfläche *Clear filter* kann man auf den Standardwertebereich von 0.0 bis 1.0 zurücksetzen. Wurde ein Wertebereich festgelegt, der nicht von 0.0 bis 1.0 reicht, so wird die Schaltflächenbeschriftung rot eingefärbt, damit für den bzw. die BenutzerIn ersichtlich ist, dass ein Filter definiert wurde.

Relationen zwischen Konzepten, die explizit auf *unrelated* gesetzt wurden, werden unabhängig vom definierten Wertebereich immer angezeigt. Der eingestellte Wertebereich wird immer auf die aktuelle Alignment-Sicht, d.h. ob nur aggregierte oder regelbasierte Ähnlichkeiten angezeigt werden sollen, angewendet.
Wenn z.B. der Filter von 0.5 bis 1.0 eingestellt wurde und aktuell nur aggregierte Ähnlichkeiten angezeigt werden, jedoch dabei nur Werte unter 0.5 vorhanden sind, so werden in der passiven Ontologie-Visualisierung keine Konzepte farblich hervorgehoben. Existieren regelbasierte Ähnlichkeitswerte über 0.5 und erfolgt ein Wechsel zur Ansicht der regelbasierten Ähnlichkeitswerte, so werden diese farblich markiert.

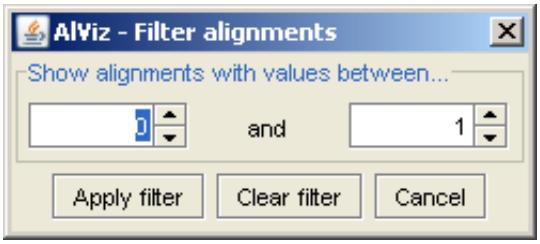

Abbildung 9.9: Dialog zum Filtern von bestimmten Ähnlichkeitswerten

Schließlich können mit der Schaltfläche *Save...* aus der Bedienleiste die aktuellen Alignments in einer XML-Datei gespeichert werden. Der bzw. die BenutzerIn wird dabei durch einen Standard-Speicherdialog unterstützt.

9.1.2.2 Setzen von Alignments

Das Setzen von Alignments hat sich vom Grundprinzip her gegenüber der ursprünglichen AlViz-Version nicht verändert (siehe Kapitel 7.2.1). Änderungen waren allerdings beim Befüllen der Dialoge zum Setzen der Alignments notwendig.

Da nun die Ähnlichkeitsregeln vom zuletzt gestarteten Ontology Alignment Algorithmus bzw. von der zuletzt geladenen XML-Datei mit Alignments gesteuert werden, musste das Laden dieser Regeln dementsprechend angepasst werden. Außerdem musste beachtet werden, für welche Ähnlichkeitsklassen der entsprechende Algorithmus Ähnlichkeitsregeln zur Verfügung gestellt hat.
Eine weitere Anpassung war auf Grund der Unterscheidung zwischen der Ansicht von entweder aggregierten Ähnlichkeitswerten oder regelbasierten Ähnlichkeitswerten notwendig. Wenn die Ansicht auf aggregierte Ähnlichkeitswerte gestellt wird, so kann nur zwischen *similar* und *unrelated* gewählt werden. Die Abbildungen 9.10 und 9.11 zeigen

die Dialoge für das Setzen von aggregierten und regelbasierten Ähnlichkeitswerten.
In der Abbildung 9.10 ist ersichtlich, dass beim Festlegen einer vorhandenen Ähnlichkeitsbeziehung bei Ansicht der aggregierten Werte logischerweise keine Ähnlichkeitsregel als Auswahlbox angezeigt wird, da ja nur ein Wert, der repräsentativ für die Aggregation der regelbasierten Ähnlichkeitswerte angesehen werden soll, verändert werden kann.

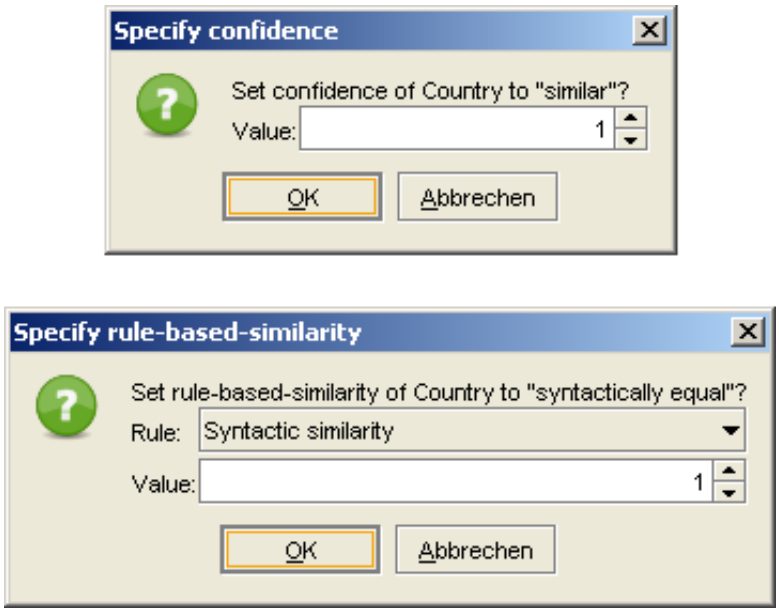

Abbildung 9.10: Dialoge zum Setzen von aggregierten Ähnlichkeitswerten (oben) und regelbasierten Ähnlichkeitswerten für die Ähnlichkeitsklasse *SYNTACTICALLY_EQUAL* (unten)

Wird vom Benutzer bzw. von der Benutzerin zwischen Konzepten eine regelbasierte Ähnlichkeitsregel definiert wo noch kein aggregierter Ähnlichkeitswert vorhanden ist, so wird abhängig von der Ähnlichkeitsklasse der gewählten Regel ein vordefinierter aggregierter Ähnlichkeitswert zugewiesen (siehe Tabelle 9.1). Die Werte dieser Tabelle wurden nach dem Testen der Auswirkungen von Ähnlichkeitsregeln auf die aggregierten Werte mit dem Algorithmus FOAM bestimmt.

9.1.3 Integration von FOAM in AlViz

Da der Ontology Alignment Algorithmus FOAM (siehe Kapitel 6) bereits in der Ausgangsversion von AlViz zum Einsatz kam, wurde auch dieser Algorithmus herangezogen,

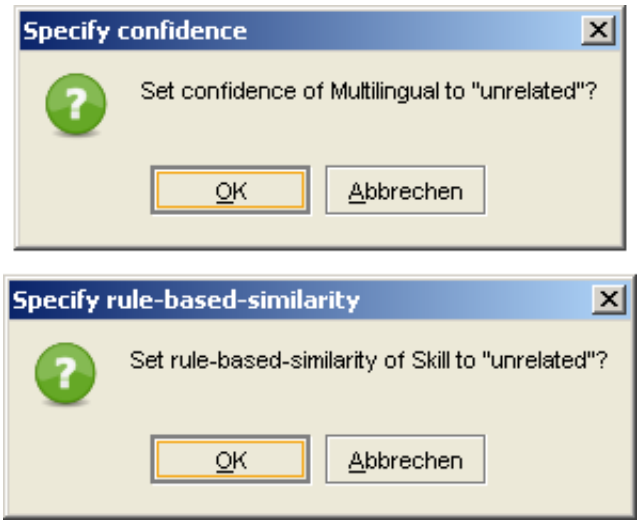

Abbildung 9.11: Dialoge zum Setzen der Beziehungen zwischen Konzepten auf *UNRELATED* bei Ansicht der aggregierten (oben) und regelbasierten (unten) Ähnlichkeitswerte

AlVizSimilarityClass-Element Element	**aggregierter Ähnlichkeitswert**
EQUAL	1.0
SYNTACTICALLY_EQUAL	0.7
SIMILAR	0.5
BROADER_THAN	0.1
NARROWER_THAN	0.1

Tabelle 9.1: Vordefinierte aggregierte Ähnlichkeitswerte für die erstmalige Definition einer Ähnlichkeitsregel durch den bzw. die BenutzerIn

um die Integration in AlViz exemplarisch vorzuzeigen.

Da FOAM keine Schnittstellen entsprechend der Ontology Alignment API zur Verfügung stellt, wurde eine Wrapper-Klasse erstellt, die die Ontologien, Alignments sowie Parameter laut der Definition der API entgegennimmt, daraufhin konvertiert und die Eigenschaften der FOAM-Implementierung befüllt bzw. die Methoden der FOAM-Implementierung aufruft.
Ziel bei der Integration war es daher, die eigentliche FOAM-Implementierung so unverändert, wie nur möglich zu lassen.

Folgende Unterabschnitte erläutern die erstellte Wrapper-Klasse, den Parameter-Konfigurationsdialog, der von AlViz aus geöffnet wird und schließlich die Konfigurationsdatei von AlViz, mit der die Einbindung von FOAM in AlViz erfolgt.

9.1.3.1 FOAM Wrapper-Klasse

Neben der Implementierung des Ontology Alignment API Interfaces `AlignmnetProcess` aus dem Package `org.semanticweb.owl.align` leitet sie auch von der Klasse `AlVizAlignment` aus dem Package `at.ac.tuwien.ifs.alviz.align.api` ab, damit die festgelegte Interface-Methoden aus `AlignmnetProcess` bzw. `Alignment` nicht noch einmal implementiert werden müssen bzw. damit auch die bereits besprochenen Erweiterungen der API genützt werden können.
Es wird ein Konstruktor ohne Parameter - dieser ist nach der Ontology Alignment API erforderlich - zur Verfügung gestellt. Darin wird *FOAM* als Alignment Algorithmus Name des Alignments festgelegt, der die bei FOAM verfügbaren Ähnlichkeitsregeln definiert, sowie Level bzw. der Typ des Alignments auf *0* bzw. *11* gesetzt.
Sobald beide OWL-Ontologien aus `OWLOntology`-Objekte mit den Set-Methoden festgelegt wurden, werden die enthaltenen Entitäten extrahiert und für die internen Datenstrukturen von FOAM konvertiert.

Durch den Aufruf der `align`-Methode, die als Parameter jeweils ein Objekt einer Implementierung des `Alignment`-Interfaces und des `Parameters`-Interfaces erwartet, wird die Generierung der Alignments mit FOAM angestoßen.
Doch bevor die Wrapper-Klasse den FOAM-Algorithmus aufruft, werden erforderliche Objekte in der Basisklasse der FOAM-Implementierung erstellt bzw. Eigenschaften entsprechend dem `Parameters`-Objekt, welches die Informationen des Parameter-Konfigurationsdialogs enthält, gesetzt. Außerdem werden die bereits von AlViz erhaltenen Alignments für eine spätere schnellere Verarbeitung zwischengespeichert und entsprechend

der durch den bzw. die AlViz-BenutzerIn festgelegten Parameter (siehe nächstes Kapitel 9.1.3.2) auch der Hauptklasse des FOAM-Algorithmus weitergegeben.

Nachdem der FOAM-Algorithmus erfolgreich durchgelaufen ist, extrahiert die Wrapper-Klasse die Ergebnisliste mit den gefundenen Alignments. Da die Wrapper-Klasse das Interface `AlignmentProcess` implementiert bzw. von `AlVizAlignment` ableitet, können die Alignments als Eigenschaft der Wrapper-Klasse mit `AlVizCell`-Objekten gespeichert werden. AlViz kann über die Methode `getElements` der Wrapper-Klasse, welche auf Grund der Ontology Alignment API Implementierung vorhanden sein muss, eine Enumeration von `AlVizCell`-Objekten verwenden.
Bevor allerdings die Prozesskontrolle mit dem Ende der `align`-Methode an AlViz zurückgegeben wird, wird überprüft, ob zumindest alle Alignments mit aggregiertem Ähnlichkeitswert von 0.0 in der Alignmentmenge enthalten sind. Dazu wird die zu Beginn der Methode zwischengespeicherte Alignmentmenge herangezogen. Falls Alignments fehlen, so werden diese hinzugefügt. Abhängig von den festgelegten Parametern, wird auch das Vorhandensein von Alignments mit einem Ähnlichkeitswert größer als 0.0 überprüft.

9.1.3.2 Parameter-Konfigurationsdialog

Wie in dieser Arbeit bereits erwähnt muss ein Parameter-Konfigurationsdialog von der Klasse `AlignAlgorithmConfigDlg` aus dem Package `at.ac.tuwien.ifs.alviz.align.config` ableiten. Dadurch muss eine Set- bzw. Get-Methode für das Interface `Parameters` implementiert werden. Weiters initialisiert AlViz den Dialog über einen Konstruktor mit einem Parameter vom Type `java.awt.Dialog`, wodurch ein solcher auch zur Verfügung gestellt werden muss.

Im Konstruktor des FOAM Konfigurationsdialogs werden das übergebene `Dialog`-Objekt sowie ein Dialogtitel für den Aufruf des Konstruktors der Superklasse verwendet. Weiters erfolgt darin die Erstellung aller Steuerelemente sowie die Festlegung der Größe und Position des Dialogs.

Bevor der Dialog geöffnet wird, erfolgt von AlViz der Aufruf der Methode `setParameters`. Es werden in dieser Methode daher die Elemente des `Parameters`-Objekts extrahiert und die Werte auf die Eigenschaften der jeweiligen Steuerelemente übertragen.
Abbildung 9.12 zeigt den Konfigurationsdialog für FOAM, der dem bzw. der BenutzerIn nach der `setParameters`-Methode gezeigt wird.

Der Großteil der Parameter wurde bereits in Kapitel 6.6 erläutert. *Alignment strategy* entspricht dem Parameter `strategy`, wobei hier nur die Strategien *Manual Weighted* und

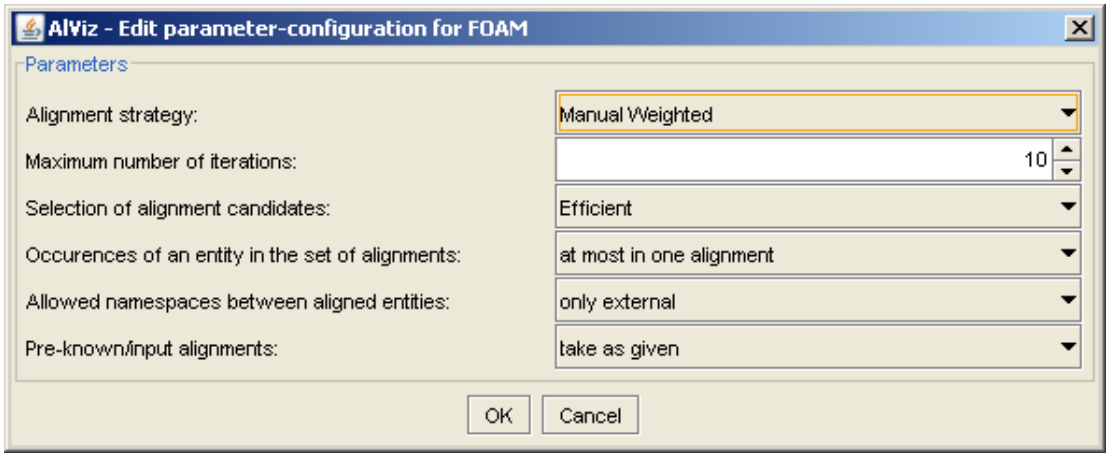

Abbildung 9.12: Parameter-Konfigurationsdialog von FOAM in AlViz

Manual Sigmoid auswählbar sind, da die anderen Strategien in der FOAM-Algorithmus-Implementierung teilweise nicht vollständig ausprogrammiert wurden. *Maximum number of iterations* entspricht dem Parameter `maxiterations`, *Selection of alignment candidates* dem Parameter `efficientAgenda`, *Occurences in the set of alignments* dem Parameter `removeDoubles` und *Allowed namespaces between aligned entities* dem Parameter `internaltoo`. Der FOAM Parameter `semi` wird immer auf `FULLAUTOMATIC` gesetzt, da das Ergebnis erst nach dem Durchlauf aller Iterationen des Algorithmus an AlViz zurückgegeben werden sollen. Alle restlichen FOAM-Parameter werden nicht gesetzt bzw. verwendet. Somit werden immer alle Alignments zurückgegeben, d.h. der Parameter `cutoffvalue` findet keine Beachtung.

Einzig und allein die Einstellung zu *Pre-known/input-alignments* ist kein Standard-FOAM-Parameter. Damit kann festgelegt werden, dass entweder

- alle aggregierten Alignments aus AlViz als gegeben betrachtet werden sollen, d.h. auch nach dem FOAM-Durchlauf müssen diese Ähnlichkeiten vorhanden sein (Einstellungswert *take as given*) oder
- alle Ähnlichkeitswerte (sowohl aggregierte als auch regelbasierte) werden nur zur Initialisierung des FOAM-Algorithmus verwendet, wodurch die Ähnlichkeitswerte während dem Durchlauf wieder überschrieben werden können. Ausnahme bilden hierbei Alignments mit einem aggregierten Ähnlichkeitswert von 0. Diese werden auch hier als gegeben betrachtet (Einstellungswert *use only for initialization (except unrelated similarities)*).

9.1.3.3 Konfigurationsdatei

Listing 9.1 zeigt die Datei `alvizconfig.xml` für die Integration von FOAM in AlViz. Der Wert `FOAM` wurde sowohl für die Bezeichnung des Algorithmus als auch für das Plugin-Verzeichnis angegeben, da alle FOAM relevanten Dateien inklusive externer Bibliotheken unter `/Applications/Protege_3.3.1/plugins/at.ac.tuwien.ifs.alviz/FOAM` abgelegt wurden. Die Klasse beim Parameter `classname` entspricht der Wrapper-Klasse von FOAM. Diese Klasse wird dort angegeben, da sie die Schnittstelle der Ontology Alignment API umsetzt.

Als zentrale Java-Archivdatei wird nur `foam.jar` definiert. Diese Datei enthält sowohl die Wrapper-Klasse und die Klasse für den Parameter-Konfigurationsdialog als auch die Implementierung des FOAM-Algorithmus.

Die Klasse des Konfigurationsdialogs wird schlussendlich beim Parameter `classname` des Elements `ConfigDialog` angegeben.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <AlVizConfig>
3   <AlignmentAlgorithm name="FOAM" plugindir="FOAM"
        classname="at.ac.tuwien.ifs.alviz.foam.FOAM">
4     <MainJars>
5       <Jar name="foam.jar"/>
6     </MainJars>
7     <ConfigDialog classname="at.ac.tuwien.ifs.alviz.foam.FOAMConfigDlg" />
8   </AlignmentAlgorithm>
9 </AlVizConfig>
```

Listing 9.1: `alvizconfig.xml` für die Integration von FOAM

9.2 Evaluierung

Nach der Implementierung der in dieser Arbeit beschriebenen Architektur zur Integration von Ontology Alignment Algorithmen in AlViz, wurde eine informale Evaluierung von AlViz mit Fokus auf der Verwendung des über diese Architektur eingebundenen Alignment Algorithmus FOAM durchgeführt.

Ziel war es die Bedienbarkeit vor allem der hinzugekommenen Benutzeroberflächen und die Verwendbarkeit der Funktionen in AlViz zum Nachvollziehen der generierten Alignments zu untersuchen. Weiterführendes Ziel der Evaluierung war, dass Erkenntnisse für weitere nützliche Features für AlViz gesammelt werden.

9.2.1 Testpersonen und -aufbau

Die Evaluierung wurde mit zwei Testpersonen durchgeführt, wobei eine Person, die im weiteren Verlauf Testperson *A* genannt wird, über die Grundprinzipien von Semantic Web bescheid wusste, aber sich noch nicht mit Ontology Alignment Algorithmen beschäftigt hatte und weder mit Protegé noch mit AlViz arbeitete.
Die andere Testperson, die in weiterer Folge als Testperson *B* bezeichnet wird, verwendete bereits die Alignment Algorithmen FOAM und HMatch[2] und hatte daher auch detailliertes Wissen über die Arbeitsweise solcher Algorithmen. Testperson *B* kam zwar schon mit Protegé in Berührung allerdings nocht nicht mit AlViz.

Die Evaluierung wurde auf einem Lenovo T400 Notebook unter Windows XP mit jeder Testperson einzeln durchgeführt und die Dauer mit jeweils einer Stunde angesetzt.
Bevor die Testpersonen mit AlViz arbeiteten, wurde eine kurze Einführung zum Thema Ontology Alignment gegeben, und es wurden der Grundaufbau der Benutzeroberfläche sowie Einsatzbereich und Basisfunktionalitäten von AlViz näher erläutert. Weiters wurde auf die in dieser Arbeit hinzugefügten Erweiterungen aufmerksam gemacht.

Jede Testperson erhielt eine Liste mit durchzuführenden Evaluierungstätigkeiten bzw. dazugehörigen Evaluierungsfragen, die schriftlich zu beantworten waren und wie folgt aussahen:

1. Was ist der Unterschied zwischen den drei Optionen bezüglich Alignments beim Startdialog von AlViz?
   - *Load alignments from AlViz-file*
   - *Generate alignments with ontology alignment algorithm*
   - *Set pre-known alignments first*
2. Beantworten Sie folgende Frage erst nachdem Sie mit FOAM Alignments generiert haben: Welche Informationen werden bei der Ansicht *show confidence-values* im Gegensatz zu *show rule-based-similarity-values* visualisiert?

[2] http://islab.dico.unimi.it/hmatch/news.php

3. Wofür kann die Filter-Funktionalität verwendet werden?
4. Welche Information wird ihrer Vermutung nach durch die unterschiedliche farbliche Markierung von Knoten in der passiven Ontologie-Visualisierung bei Alignments repräsentiert und welcher Zusammenhang besteht dabei mit den beiden unterschiedlichen Ansichten der 2. Frage?
5. Welche Auswirkungen vermuten Sie bzw. ergeben sich bei der Einstellung/Veränderung der einzelnen Parameter des FOAM-Algorithmus?
    - *Alignment strategy* (mögliche Werte: Manual Weighted, Manual Sigmoid):
    - *Maximum number of iterations*:
    - *Selection of alignment candidates* (mögliche Werte: Efficient, Complete):
    - *Occurences in the set of alignments* (mögliche Werte: at most in one alignment, in serveral alignments):
    - *Allowed namespaces between aligned entities* (mögliche Werte: only external, external and internal):
    - *Pre-known/input-alignments* (mögliche Werte: take as given, use only for initialization (except unrelated similarities)):

Falls Fragen zur Tätigkeitsliste oder zum Arbeiten mit AlViz auftraten, so konnte die Testaufsichtsperson zu Rate gezogen werden. Während der Durchführung der Tätigkeiten durch die Testpersonen beobachtete die Testaufsichtsperson die Interaktionen mit AlViz bzw. das Verhalten der Testpersonen und machte sich entsprechende Notizen, welche ebenfalls in die Ausarbeitung der Ergebnisse einflossen.

Am Ende der Evaluierung wurde jede Testperson gebeten die positiven Aspekte von AlViz in Bezug auf den vorgegebenen Fokus sowie die vermissten Hilfestellungen und Funktionalitäten während dem Arbeiten mit AlViz anzuführen.

Zu Beginn wurden als Testontologien jene aus Kapitel 5.4.1 (siehe Abbildung 7.3) verwendet. Nachdem die Testpersonen einen Teil der Funktionalitäten von AlViz ausprobiert hatten, wurde ein wesentlich umfangreicheres Ontologiepaar, d.h. mit mehr Konzepten und Hierarchieebenen und somit mehreren Kanten im Small World Graph, mit AlViz geöffnet (siehe Abbildung 9.6). Die Ontologien wurden immer von der Testaufsichtsperson geöffnet.

9.2.2 Ergebnis

Die erste Frage wurde sowohl von Testperson *A* als auch *B* fast vollständig richtig beantwortet. Vor allem Testperson *A* war sich bei der Option *Set pre-known alignments first* ziemlich unsicher worin der genaue Unterschied zu den anderen Optionen liegt, da natürlich ohne den Klick auf einen Knoten, von dem aus ein Alignment existiert, dies nicht sofort sichtbar wird.

Auf die zweite Frage wusste keine der beiden Testpersonen ohne die Hilfe der Testaufsichtsperson eine Antwort. Nachdem die Testaufsichtsperson den Hintergrund dieser beiden Ansichten erläuterte, wurde zumindest für Testperson *B* die Verwendung einigermaßen klar. Testperson *B* war bei der Ansicht *show confidence-values* vor allem durch das Wort *confidence* verunsichert und konnte es nicht mit dem aggregierten Ähnlichkeitswert eines Alignments in Verbindung bringen.
Testperson *A* musste der Sinn dieser Ansichten nochmals erklärt werden, wonach dann auch dieser Person klar wurde, dass mit Hilfe der Ansicht *show rule-based-similarity-values* der bzw. die BenutzerIn bei der Beantwortung der Frage *Warum wurde vom Alignment Algorithmus dieses Alignment gefunden?* unterstützt wird.
Allerdings wurden dann dabei gleich Verbesserungsvorschläge vorgebracht. Da bei der Ansicht *show rule-based-similarity-values* nur immer die größte Ähnlichkeitsregel in der passiven Ontologie-Visualisierung bzw. beim Kontextmenü zum entsprechenden Knoten angezeigt wird, obwohl vielleicht mehrere Regeln zwischen den Konzepten zutreffen, würden sich die beiden Testpersonen eine Auflistung aller zutreffenden Ähnlichkeitsregeln mit -werten wünschen. Denn dadurch kann gezielt analysiert bzw. angepasst werden, welche Regeln gefunden wurden und welche nach Meinung des bzw. der BenutzerIn nicht korrekt sind oder noch fehlen. Derzeit kann eben nur eine gesetzt werden, wodurch die aktuell größte Ähnlichkeitsregel überschrieben wird.
Nach Meinung von Testperson *A* wäre auch eine auffällige Änderungen z.B. an der Visualisierung oder am Mauszeiger hilfreich, um leicht zu erkennen in welcher Alignment-Ansicht man sich befindet.

Die dritte Frage konnte von beiden Testpersonen nach kurzem Durchtesten bzw. kurzer Analyse der verfügbaren Steuerelemente im dazugehörigen Dialog rasch beantwortet werden. Auch die rote Markierung der Schaltflächenbeschriftung bei einem festgelegten Wertebereich ungleich 0.0 bis 1.0 wurde von beiden positiv aufgenommen.
Testperson *A* äußerte hinsichtlich des Filters den Wunsch auf erweiterte Funktionalität. Derzeit wird der definierte Wertebereich des Filters einfach auf die Ähnlichkeitswerte (egal ob aggregiert oder regelbasiert) angewendet. D.h. wenn bei der Ansicht der aggre-

gierten Ähnlichkeitswerte ein Knoten farblich markiert ist, kann diese Markierung nach dem Umschalten zur Ansicht der regelbasierten Ähnlichkeitswerte wieder verschwinden, obwohl Werte vorhanden sind, die jedoch nicht im Wertebereich des Filters liegen. An dieser Stelle würde sich Testperson *A* eine Möglichkeit wünschen, den Filter für eine bestimmte Ansicht zu fixieren, damit die Werte des gleichen Alignments in der anderen Ansicht sichtbar sind, obwohl sie außerhalb des Wertebereichs liegen. Außerdem wäre es nach Testperson *B* hilfreich auch nach bestimmten Ähnlichkeitsklassen zu filtern.

Bei der vierten Frage wurden beide Testpersonen durch das Vorkommen der Farbe grün in beiden Ansichten verwirrt. Die Farbe schwarz wurde dabei von beiden sofort als Repräsentation für keine Ähnlichkeit erkannt. Die Testaufsichtsperson musste daher den Hintergrund der restlichen Farben aufklären. Testperson *A* als auch *B* würden sich in AlViz an dieser Stelle eine Hilfestellung bzw. Information zu den Farben wünschen.

Bei der letzten Frage wurde ersichtlich, dass für die gezielte Beantwortung einiges an Hintergrundwissen vorhanden sein muss. Testperson *A* konnte bei wenigen Einstellungen zwar Vermutungen anstellen, die teilweise auch richtig waren, mehr allerdings nicht. Testperson *B* konnte auf Grund des vorhandenen Wissens so gut wie alle Einstellungen beschreiben. Lediglich bei der Einstellung *Pre-known/input-alignments* wusste Person *B* nicht genau was hinter *use only for initialization (except unrelated similarities)* steckt.

Grundsätzlich ist zu sagen, dass beide Testpersonen eine Übersicht über alle vorhandenen Alignments vermissten. Der Wunsch einer solchen Funktionalität trat vor allem beim Arbeiten mit der größeren Ontologie auf, da das Durchklicken aller Knoten des Graphen in kurzer Zeit nicht möglich ist.
Weiters ist es nach Meinung von Testperson *B* sehr schwierig die Veränderungen zwischen nacheinander erfolgten Durchläufen von eingebundenen Alignment Algorithmen zu erkennen. Hier würde eine übersichtliche Gegenüberstellung der gefundenen Alignments von davor und danach äußerst hilfreich sein. Dabei wäre vielleicht auch eine Versionierungsfunktion nützlich, damit auch nicht hintereinander generierte Ergebnisse, verglichen werden können.

Bezüglich Bedienbarkeit wurde von den Testpersonen der Aufbau der Dialoge als logisch, gut strukturiert und verständlich bewertet. Weiters wurde die Stabilität und Performance von AlViz, obwohl die evaluierte Version noch immer als Prototyp angesehen wird, von beiden Personen positiv hervorgehoben.
Ein Kritikpunkt kam bei der Gruppierung der Schaltflächen der Bedienleiste auf. Für beide war es nicht sofort erkennbar, dass die Schaltflächen von *Load...* bis *Save...* thematisch zusammengehören und sich auf die Alignments beziehen.

Trotz einiger Verbesserungsvorschläge wurde von den Testpersonen die Integration des Ontology Alignment Algorithmus FOAM als besonders gelungen empfunden. Um gezielte Verbesserungen der Alignments durch die Veränderungen der Parameter zu erreichen ist zwar oft ein detailliertes Hintergrundwissen des entsprechenden Algorithmus notwendig, allerdings kann durch die Integration von Alignment Algorithmen in AlViz schnell die Menge der generierten bzw. aktualisierten Alignments visualisiert werden.

Kapitel 10

Abschluss

Im letzten Kapitel wird nochmals eine Zusammenfassung der Arbeit geliefert, worauf eine Beschreibung der aus dieser Arbeit gezogenen Schlussfolgerungen folgt. Im dritten Abschnitt wird schließlich ein Ausblick auf notwendige bzw. mögliche Erweiterungen von AlViz gegeben.

10.1 Zusammenfassung

Das Semantic Web stellt derzeit ein äußerst aktives Forschungsgebiet dar. Die Visionen dabei sind, dass Maschinen die Bedeutung aller Informationen des Webs interpretieren und daraus weiteres Wissen ableiten können. Um dies zu ermöglichen muss das heutige Web mit maschinenverarbeitbaren Metadaten, die die Semantik abbilden, angereichert werden. Im Semantic Web Bereich werden dazu Ontologien verwendet. Vom World Wide Web Consortium wurden dazu die Standards RDF, RDF-Schema sowie die gegenüber von RDF-Schema ausdrucksstärkere Ontologie-Beschreibungssprache OWL definiert.

Die größte Herausforderung, die sich im Zusammenhang mit dem Kombinieren von Informationen aus unterschiedlichen Quellen, was auch in den Visionen des Semantic Webs verankert ist, ergibt, ist die Bewältigung der Heterogenität des Webs. D.h. verschiedene Personen bzw. Institutionen beschreiben die Bedeutung von Informationen aus gleichen bzw. ähnlichen Domänen auf unterschiedliche Art und Weise.
Für die Verarbeitung bzw. das Verknüpfen von Wissen, dessen Semantik durch Ontologien beschrieben wird, ist es somit erforderlich, dass die Ontologien aufeinander abgestimmt werden, damit sie konsistent sowie kohärent zueinander sind. Dieser Vorgang wird als Ontology Alignment bezeichnet, wobei die Ermittlung der Ähnlichkeiten

bzw. Übereinstimmungen - so genannte Alignments - zwischen den zugrunde liegenden Ontologien die Hauptaufgabe darstellt.

Es gibt bereits eine Menge von Ontology Alignment Algorithmen wie z.B. FOAM, ASMOV, RiMOM und Lily, die unter anderem basierend auf syntaktischen und strukturellen Vergleichen der Ontologie-Elemente die Alignments bestimmen und dabei eine bestimmte Wahrscheinlichkeit für die Richtigkeit der Ähnlichkeit (auch Ähnlichkeitswert genannt) dem Alignment zuweisen.
Seit 2004 werden jährlich durch die Ontology Alignment Evaluation Initiative (OAEI) eingereichte Alignment Algorithmen evaluiert. Unterschiedliche Evaluierungsszenarien, die von Jahr zu Jahr kontinuierlich verfeinert und erweitert werden, zeigen die Stärken und Schwächen der Algorithmen auf.

Die gefundenen Alignments sind beispielsweise bei FOAM nach dem Durchlauf des Algorithmus nur in Form einer textuellen Liste verfügbar. Bereits ab einer geringen Anzahl an gefundener Alignments ist solch eine Liste schwierig zu interpretieren.
Einige Tools verwenden daher Techniken aus dem Bereich der Visualisierung, um die Alignments übersichtlicher und verständlicher aufzubereiten. Eines dieser Tools ist AlViz, welches genau zwei Ontologien gegenüberstellen kann und diese mit einer baumstrukturierten Darstellung sowie Small World Graphen visualisiert. Die Alignments werden durch farbliche Markierungen nach dem Mausklick auf die Ontologie-Elemente in beiden Darstellungen hervorgehoben. Falls bestimmte Alignments nicht gefunden wurden bzw. der Ähnlichkeitswert eines Alignments nicht korrekt ist, so kann der bzw. die BenutzerIn entsprechende Änderungen vornehmen.

Das Problem bei AlViz bestand vor der Umsetzung des praktischen Teils dieser Arbeit allerdings dabei, dass keine Anbindung an einen Alignment Algorithmus vorzufinden war und die Alignments in einer XML-Datei in einem definierten Format vorliegen mussten. Weiters war die Implementierung eng an FOAM gekoppelt.

Basierend auf der Ontology Alignment API von Jérôme Euzenat, die auch bei den Evaluierungen der OAEI zum Einsatz kommt, wurde eine Architektur bzw. Schnittstelle entworfen und umgesetzt, mit Hilfe derer Ontology Alignment Algorithmen an AlViz angebunden bzw. integriert und Alignments zwischen AlViz und dem Algorithmus ausgetauscht werden können.
In weiterer Folge wurden zwei Ansichten in Bezug auf Alignments bei AlViz eingeführt. Durch die eine Ansicht kann sich der bzw. die BenutzerIn die Alignments mit dem zugeordneten Ähnlichkeitswert - auch Konfidenzwert bzw. aggregierter Ähnlichkeitswert genannt - anzeigen lassen. Durch die andere Ansicht sind jene Ähnlichkeitswerte ersichtlich,

die internen Regeln des Algorithmus zugeordnet sind - auch regelbasierte Ähnlichkeitswerte genannt. Mit Hilfe dieser regelbasierten Ähnlichkeitswerte wird der aggregierte Ähnlichkeitswert zum Alignment bestimmt. Die Ansicht der regelbasierten Ähnlichkeiten ist nur dann möglich, wenn die internen Regeln vom Alignment Algorithmus mit dem Alignment über die entworfene Schnittstelle an AlViz mitgeliefert werden.
Die internen regelbasierten Ähnlichkeitswerte geben Aufschluss über den Grund des Auffindens eines bestimmten Alignments, was natürlich für die Interpretation sehr hilfreich ist.

Anhand des Ontology Alignment Algorithmus FOAM wurde in der vorliegenden Arbeit schließlich die Integration eines konkreten Algorithmus über die umgesetzte Architektur in AlViz vorgezeigt. Dabei wurde auch ein Parameter-Konfigurationsdialog für das gezielte Setzen von FOAM-Parametern in AlViz eingebunden.

Die Ergebnisse der durchgeführten Evaluierung zu den vorgestellten Erweiterungen von AlViz werden am Ende der Arbeit präsentiert.

10.2 Schlussfolgerung

Durch die Implementierung einer Architektur mit einheitlicher Schnittstelle zwischen AlViz und den Ontology Alignment Algorithmen ist es gelungen, dass unterschiedliche Alignment Algorithmen einfach in AlViz integriert werden können. Daher können die von den Algorithmen gefundenen Alignments durch AlViz schnell, übersichtlich und verständlich dem bzw. der BenutzerIn dargestellt werden.
Da die Schnittstellendefinition auf der Ontology Alignment API basiert, die auch von der OAEI zur Evaluierung der Alignment-Ergebnisse eingereichter Algorithmen verwendet wird, müssen solche Algorithmen nur in der Konfigurationsdatei von AlViz eingetragen werden, um mit AlViz zu interagieren. Voraussetzung dabei ist allerdings auch, dass diese Algorithmen die Ontologiedaten mit den bei der Alignment API mitgelieferten Interfaces bzw. Klassen zur Repräsentation und Verarbeitung von OWL-Ontologien an AlViz übermitteln.

Ein weiterer Vorteil, der sich durch die Architektur ergibt, ist, dass ein Alignment Algorithmus mehrmals hintereinander gestartet werden kann, wobei die Alignment-Ergebnisse der vorangegangenen Iteration als Eingabe für die nächste Iteration verwendet werden können, um die Alignment-Ergebnisse zu erweitern bzw. zu verbessern. Auch die Kombination verschiedener Algorithmen ist nun möglich, d.h. entsprechend

den zugrundeliegenden Ontologien können die Stärken bestimmter Algorithmen gezielt eingesetzt werden.

Durch die Verwendung einheitlicher Interfaces und der Java Reflection API zum Aufruf der eingebundenen Ontology Alignment Algorithmen und der Parameter Konfigurationsdialoge, die von den Alignment Algorithmen optional zur Verfügung gestellt werden können, kann das Ziel der losen Kopplung zwischen AlViz und den Algorithmen, als auch der anderen gesetzten Ziele als erfüllt betrachtet werden.

Auch die informalen Evaluierungsergebnisse haben gezeigt, dass durch die Erweiterung von AlViz die Anwendung von Alignment Algorithmen ohne detaillierte Auseinandersetzung mit den Algorithmen möglich ist.

Trotz der Erweiterungen von AlViz, die im Rahmen der Arbeit durchgeführt wurden, ist die aktuelle Version noch immer als Prototyp anzusehen. Außerdem sind durch die Erweiterungen bzw. durch die Evaluierung wieder Ideen für neue interessante AlViz-Funktionalitäten sowie Aspekte, die in der bestehenden Version auf jeden Fall verbessert werden sollten, aufgekommen.

10.3 Ausblick

Betrachtet man die Ergebnisse der Evaluierung, so sollte dem bzw. der BenutzerIn von AlViz in manchen Aspekten mehr Hilfestellung gegeben werden. Z.B. wäre dies bei der Auswahlbox der unterschiedlichen Alignment Ansichten (*show confidence-values*, *show rule-based-similarity-values*) notwendig. Weiters wäre eine Legende mit Zusammengehörigkeit von Farbe und verfügbaren Ähnlichkeitsklassen bei der Ansicht der regelbasierten Ähnlichkeitswerte äußerst hilfreich.

Ein großes Manko ist sicherlich, dass der Überblick über alle vorhandenen Alignments fehlt. Vor allem bei großen Ontologien ist es damit sehr schwierig, die Alignments zu finden. In weiterer Folge ist es auch nicht wirklich nachvollziehbar wie sich die Alignments, bei mehrmaligem Durchlauf eines Alignment Algorithmus gegenüber einer vorigen Iteration, verändert haben. D.h. dies wäre auf jeden Fall eine wichtige Erweiterung für AlViz. In diesem Zusammenhang wurde auch von einer Testperson eine interessante Wunschfunktionalität geäußert, nämlich dass nicht nur die Alignment-Ergebnisse von hintereinander erfolgten Iterationen verglichen werden sollten, sondern auch die Ergebnisse zwischen beliebigen Durchläufen. Damit wäre eine Art Versionierungsverwaltung notwendig.

Das Bearbeiten von regelbasierten Ähnlichkeitswerten stellt in der aktuellen Version ebenfalls ein Problem dar. Dadurch, dass immer nur die größte regelbasierte Ähnlichkeit gezeigt wird, werden möglicherweise restliche vorhandene regelbasierte Ähnlichkeiten dem bzw. der BenutzerIn vorenthalten. Weiters können nicht mehrere regelbasierte Ähnlichkeiten einem Alignment zugewiesen bzw. geändert werden. An dieser Stelle sollte eine Liste mit allen zu diesem Alignment verfügbaren regelbasierten Ähnlichkeiten mit der Möglichkeit zum Hinzufügen, Ändern und Löschen angezeigt werden.

Im Rahmen der Implementierung dieser Arbeit wurden durch Einführung bzw. Umstrukturierung von Java-Packages und -Klassen ansatzweise eine lose Kopplung und eine höhere Kohäsion erreicht. Allerdings sind noch immer einige Komponenten in der Architektur von AlViz verankert, die eine enge Kopplung (z.B. die Einbindung des Small World Graph Toolkits) und teilweise geringe Kohäsion aufweisen. Diese Komponenten sollten in Zukunft einer Refaktorisierung unterzogen werden, damit AlViz zu einer wartbaren und leicht erweiterbaren Software wird.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[Alani 2003] Alani, Harith: TGVizTab: An Ontology Visualisation Extension for Protégé. In: *Proceedings of Knowledge Capture (K-Cap'03), Workshop on Visualization Information in Knowledge Engineering*. Sanibel Island, Florida, USA, 2003. – URL `http://eprints.ecs.soton.ac.uk/8326/1/Alani-VIKE-camera-ready.pdf`. – Zugriffsdatum: 30.05.2009

[Antoniou und Harmelen 2003] Antoniou, Grigoris ; Harmelen, Frank V.: Web ontology language: OWL. In: *Handbook on Ontologies in Information Systems*, Springer, 2003, S. 67–92. – URL `http://www.cs.vu.nl/~frankh/postscript/OntoHandbook03OWL.pdf`. – Zugriffsdatum: 27.12.2008

[Antoniou und van Harmelen 2008] Antoniou, Grigoris ; Harmelen, Frank van: *A Semantic Web Primer (2nd Edition)*. MIT Press, 2008. – ISBN: 978-0262012423

[Berners-Lee 2000] Berners-Lee, Tim: *Semantic Web - XML2000*. Internetseite. 2000. – URL `http://www.w3.org/2000/Talks/1206-xml2k-tbl/`. – Zugriffsdatum: 27.12.2008

[Berners-Lee u. a. 2005] Berners-Lee, Tim ; Fielding, R. ; Masinter, L.: *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986)*. Internetseite. 2005. – URL `http://www.ietf.org/rfc/rfc3986.txt`. – Zugriffsdatum: 27.12.2008

[Berners-Lee und Fischetti 1999] Berners-Lee, Tim ; Fischetti, Mark: *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper San Francisco, 1999. – ISBN: 978-0062515865

[Berners-Lee u. a. 2001] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora: The Semantic Web. (2001)

[Bosca u. a. 2005] Bosca, Alessio ; Bonino, Dario ; Pellegrino, Paolo: OntoSphere: more than a 3D ontology visualization tool. In: *Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop, Trento, Italy, December 14-16, 2005, CEUR Workshop Proceedings*, 2005

[Bouquet u. a. 2004] Bouquet, Paolo ; Ehrig, Marc ; Euzenat, Jérôme ; Franconi, Enrico ; Hitzler, Pascal ; Krötzsch, Markus ; Serafini, Luciano ; Stamou, Giorgos ; Sure, York ; Tessaris, Sergio: Specification of a common framework for characterizing alignment. (2004), DEC, Nr. 2.2.1v2. – URL `http://www.aifb.uni-karlsruhe.de/WBS/phi/pub/kweb-221.pdf`. – Zugriffsdatum: 14.02.2009

[Breitman u. a. 2007] Breitman, K.K. ; Casanova, M.A. ; Truszkowski, W.: *Semantic Web: Concepts, Technologies and Applications*. Springer Verlag London, 2007. – ISBN: 978-1846285813

[de Bruijn u. a. 2006] Bruijn, Jos de ; Ehrig, Marc ; Feier, Cristina ; Martíns-Recuerda, Francisco ; Scharffe, François ; Weiten, Moritz: *Ontology Mediation, Merging, and Aligning*. July 2006. – URL `http://www.dit.unitn.it/~p2p/RelatedWork/Matching/mediation-chapter.pdf`. – Zugriffsdatum: 07.02.2009

[de Bruijn u. a. 2005] Bruijn, Jos de ; Martín-Recuerda, Francisco ; Ehrig, Marc ; Polleres, Axel ; Predoiu, Livia: Ontology Mediation Management V1. (2005), JAN, Nr. 4.4.1. – URL `http://www.sti-innsbruck.at/fileadmin/documents/deliverables/Sekt/D4.4.1.pdf`. – Zugriffsdatum: 10.01.2009

[Caracciolo u. a. 2008] Caracciolo, Caterina ; Euzenat, Jérôme ; Hollink, Laura ; Ichise, Ryutaro ; Isaac, Antoine ; Malaisé, Véronique ; Meilicke, Christian ; Pane, Juan ; Shvaiko, Pavel ; Stuckenschmidt, Heiner ; Sváb-Zamazal, Ondrej ; Svátek, Vojtech: *Results of the Ontology Alignment Evaluation Initiative 2008*. 2008. – URL `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-431/oaei08_paper0.pdf`. – Zugriffsdatum: 01.03.2009

[Castano u. a. 1998] Castano, Silvana ; Antonellis, Antonellis D. ; Fugini, Maria G. ; Pernici, Barbara: Conceptual schema analysis: techniques and applications. In: *ACM Transactions on Database Systems* 23 (1998), Nr. 3, S. 286–333. – ISSN 0362-5915

[David u. a. 2007] David, Jérôme ; Guillet, Fabrice ; Briand, Henri: *Association Rule Ontology Matching Approach*. Internetseite. 2007. – URL `http://exmo.inrialpes.fr/people/jdavid/publies/JDavid_IJSWIS_2007.pdf`. – Zugriffsdatum: 01.03.2009

[DCMI 2009] DCMI: *Dublin Core Metadata Initiative - Making it easier to find information*. Internetseite. 2009. – URL `http://dublincore.org/`. – Zugriffsdatum: 12.05.2009

[Ding u. a. 2002] Ding, Ying ; Fensel, Dieter ; Klein, Michel ; Omelayenko, Borys: The Semantic Web: Yet Another Hip. In: *Data & Knowledge Engineering* 41 (2002), S. 205–227

[Do u. a. 2002] Do, Hong-Hai ; Melnik, Sergey ; Rahm, Erhard: Comparison of Schema Matching Evaluations. In: *Proceeding GI-Workshop "Web and Databases"* (2002), S. 221–237. – URL `http://lips.informatik.uni-leipzig.de/files/2002-28.pdf`. – Zugriffsdatum: 14.02.2009

[Ehrig 2006] Ehrig, Marc: *Ontology Alignment - Bridging the Semantic Web Gap*. Springer Verlag Berlin, 2006. – ISBN: 978-0387328058

[Ehrig u. a. 2005a] Ehrig, Marc ; Haase, Peter ; Stojanovic, Nenad ; Hefke, Mark: Similarity for Ontologies - A Comprehensive Framework. In: *13th European Conference on Information Systems*. Regensburg, MAY 2005. – URL `http://www.aifb.uni-karlsruhe.de/WBS/meh/publications/ehrig05similarity.pdf`. – Zugriffsdatum: 24.01.2009

[Ehrig und Staab 2004] Ehrig, Marc ; Staab, Steffen: QOM - Quick Ontology Mapping. volume 3298 of LNCS (2004), NOV, S. 683–697. – URL `http://www.aifb.uni-karlsruhe.de/WBS/meh/publications/ehrig04qomISWC.pdf`. – Zugriffsdatum: 13.02.2009

[Ehrig u. a. 2005b] Ehrig, Marc ; Staab, Steffen ; Sure, York: Bootstrapping Ontology Alignment Methods with APFEL. In: *Proceedings of ISWC*, Springer, 2005, S. 186–200

[Ehrig u. a. 2006a] Ehrig, Marc ; Staab, Steffen ; Sure, York: *Framework for Ontology Alignment and Mapping*. Internetseite. 2006. – URL `http://www.uni-koblenz.de/~staab/Research/Publications/2006/alignmentJournal-submitted.pdf`. – Zugriffsdatum: 10.01.2008

[Ehrig u. a. 2006b] Ehrig, Marc ; Studer, Rudi ; Sure, York: Automatische Wissensintegration mit Ontologien. In: *Modellierung für Wissensmanagement* Institut AIFB (Veranst.), Ulrich Reimer and Knut Hinkelmann, 2006

[Ehrig und Sure 2004] Ehrig, Marc ; Sure, York: Ontology mapping - An integrated approach. In: *Proceedings of the European Semantic Web Symposium*, Springer Verlag, 2004, S. 76–91

[Ehrig und Sure 2005] Ehrig, Marc ; Sure, York: Active ontology alignment. In: *Proceedings of the Collaboration Workshop for the Future Semantic Web at ESWC-2005*, 2005

[Euzenat 2006] Euzenat, Jérôme: *An API for Ontology Alignment*. 2006. – URL `http://gforge.inria.fr/docman/view.php/117/251/align.pdf`. – Zugriffsdatum: 29.05.2009

[Euzenat u. a. 2004] Euzenat, Jérôme ; Bach, Thanh L. ; Barrasa, Jesús ; Bouquet, Paolo ; Bo, Jan D. ; Dieng, Rose ; Ehrig, Marc ; Hauswirth, Manfred ; Jarrar, Mustafa ; Maynard, Ruben L. andDiana ; Napoli, Amedeo ; Stamou, Giorgos ; Stuckenschmidt, Heiner ; Shvaaiko, Pavel ; Tessaris, Sergio ; Acker, Sven V. ; Zaihrayeu, Ilya: State fo the art on ontology alignment. (2004), AUG, Nr. 2.2.3. – URL `http://www.aifb.uni-karlsruhe.de/WBS/meh/publications/euzenat04state.pdf`. – Zugriffsdatum: 10.01.2009

[Euzenat u. a. 2005a] Euzenat, Jérôme ; Ehrig, Marc ; Castro, Raúl G.: Specification of a benchmarking methodology for alignment techniques. (2005), JAN, Nr. 2.2.2. – URL `http://www.aifb.uni-karlsruhe.de/WBS/meh/publications/kweb-222.pdf`. – Zugriffsdatum: 03.03.2009

[Euzenat u. a. 2006] Euzenat, Jérôme ; Loup, David ; Touzani, Mohamed ; Valtchev, Petko: *Ontology alignment with OLA*. 2006. – URL `http://www.iro.umontreal.ca/~owlola/index.html`. – Zugriffsdatum: 30.05.2009

[Euzenat u. a. 2005b] Euzenat, Jérôme ; Stuckenschmidt, Heiner ; Yatskevich, Mikalai: Introduction to the ontology alignment evaluation 2005. (2005), S. 61–71. – URL `http://oaei.ontologymatching.org/2005/results/oaei2005.pdf`. – Zugriffsdatum: 01.03.2009

[Gennari u. a. 2002] Gennari, John H. ; Musen, Mark A. ; Fergerson, Ray W. ; Grosso, William E. ; Crubzy, Monica ; Eriksson, Henrik ; Noy, Natalya F. ; Tu, Samson W.: The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. In: *International Journal of Human-Computer Studies* 58 (2002), S. 89–123

[Giunchiglia u. a. 2004] Giunchiglia, Fausto ; Shvaiko, Pavel ; Yatskevich, Mikalai: S-Match: an Algorithm and an Implementation of Semantic Matching, Springer, 2004, S. 61–75

[Gruber 1995] Gruber, Thomas: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In: *International Journal Human-Computer Studies* 43 (1995), November, S. 907–928. – URL `http://tomgruber.org/writing/onto-design.pdf`. – Zugriffsdatum: 27.12.2008

[van Hage 2008] Hage, Willem R. van: *Evaluating Ontology-Alignment Techniques*. University Amsterdam, Dissertation, 2008. – URL `http://www.few.vu.nl/~wrvhage/papers/wrvh_thesis_20080724.pdf`. – Zugriffsdatum: 15.02.2009

[van Ham und van Wijk 2004] Ham, Frank van ; Wijk, Jarke J. van: Interactive Visualization of Small World Graphs. In: *Information Visualization, IEEE Symposium on* 0 (2004), S. 199–206

[Hariri u. a. 2006] Hariri, Babak B. ; Sayyadi, Hassan ; Esmaili, Hassan Abolhassaniand Kyumars S.: Combining Ontology Alignment Metrics Using the Data Mining Techniques. In: *Proceedings of the 2nd International Workshop on Contexts and Ontologies: Theory, Practice and Applications*, 2006

[Herman 2003] Herman, Ivan: *Introduction to the Semantic Web*. Internetseite. November 2003. – URL `http://www.w3.org/2003/Talks/1112-BeijingSW-IH/`. – Zugriffsdatum: 27.12.2008

[Huber 2009] Huber, Martin: *Visuelle Unterstützung des Ontologie-Alignments*. TU Wien, Master Thesis, 2009

[Hughes und Ashpole 2005] Hughes, Todd C. ; Ashpole, Benjamin C.: *The semantics of ontology alignment*. 2005. – URL `www.atl.lmco.com/projects/ontology/`. – Zugriffsdatum: 31.01.2008

[Jean-Mary und Kabuka] Jean-Mary, Yves R. ; Kabuka, Mansur R.: ASMOV: Results for OAEI 2008, URL `http://www.dit.unitn.it/~p2p/OM-2008/oaei08_paper3.pdf`. – Zugriffsdatum: 01.03.2009

[Katifori u. a. 2007] Katifori, Akrivi ; Halatsis, Constantin ; Lepouras, George ; Vassilakis, Costas ; Giannopoulou, Eugenia: Ontology visualization methods: a survey. In: *ACM Comput. Surv.* 39 (2007), Nr. 4

[Klein 2001] Klein, Michel: Combining and relating ontologies: an analysis of problems and solutions. In: *Proceedings of Workshop on Ontologies and Information Sharing at IJCAI-01*, 2001

[Kolli und Doshi 2008] Kolli, Ravikanth ; Doshi, Prashant: OPTIMA: Tool for Ontology Alignment with Application to Semantic Reconciliation of Sensor Metadata for Publication in SensorMap. In: *Proceedings of the 2008 IEEE International Conference on Semantic Computing*. Washington, DC, USA : IEEE Computer Society, 2008, S. 484–485

[Lanzenberger 2008] Lanzenberger, Monika: *Visualization for Ontology Alignment*. Slides of the Presentation at Ontology and Semantic Web Workshop, 2008 Korea & Austria. 2008. – URL `http://cilab.kaist.ac.kr/workshop/2008y/pds/day2_1_Monika%20Lanzenberger.pdf`. – Zugriffsdatum: 01.03.2009

[Lanzenberger und Sampson 2006a] Lanzenberger, Monika ; Sampson, Jennifer: AlViz - A Tool for Visual Ontology Alignment. In: *IV '06: Proceedings of the conference on Information Visualization*. Washington, DC, USA : IEEE Computer Society, 2006, S. 430–440. – ISBN 0-7695-2602-0

[Lanzenberger und Sampson 2006b] Lanzenberger, Monika ; Sampson, Jennifer: Visual Ontology Alignment for Semantic Web Applications. In: *Proceedings of the 2006 International Conference on Conceptual Modeling (ER) Tucson, Arizona, USA - Workshops*, 2006, S. 405–414

[Levenshtein 1966] Levenshtein, I.V.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In: *Cybernetics and Control Theory* 10 (1966), February, Nr. 8, S. 707–710

[Li u. a. 2007] Li, Yi ; Tang, Jie ; Zhang, Duo ; Li, Juanzi: Toward Strategy Selection for Ontology Alignment, 2007

[Maedche und Staab 2002] Maedche, Alexander ; Staab, Steffen: Measuring Similarity between Ontologies. In: *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*. London, UK : Springer-Verlag, 2002, S. 251–263. – ISBN 3-540-44268-5

[Martínez-Gil u. a. 2008] Martínez-Gil, Jorge ; Alba, Enrique ; Aldana-Montes, José F.: Optimizing Ontology Alignments by Using Genetic Algorithms. In: Guéret, Christophe (Hrsg.) ; Hitzler, Pascal (Hrsg.) ; Schlobach, Stefan (Hrsg.): *NatuReS* Bd. 419, CEUR-WS.org, 2008. – URL `http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-419/paper2.pdf`. – Zugriffsdatum: 14.02.2009

[Milgram 1967] Milgram, Stanley: The small world problem. In: *Psychology Today* 22 (1967), S. 61–67

[Noy 2004] Noy, Natalia F.: Semantic Integration: A Survey Of Ontology-Based Approaches. In: *SIGMOD Record, Special Issue on Semantic Integration* 33 (2004), December, Nr. 4. – URL `http://www.dit.unitn.it/~p2p/RelatedWork/Matching/13.natasha-10.pdf`. – Zugriffsdatum: 07.02.2009

[Pinto u. a. 1999] Pinto, H. S. ; Gómez-Pérez, Asunción ; Martins, Joao P.: Some issues on ontology integration, 1999

[Rada u. a. 1989] Rada, Roy ; Mili, Hafedh ; Bicknell, Ellen ; Blettner, Maria: Development and application of a metric on semantic nets. In: *IEEE Transactions on Systems, Man and Cybernetics* 19 (1989), Nr. 1, S. 17–30

[Shvaiko und Euzenat 2005] Shvaiko, Pavel ; Euzenat, Jerome: A survey of schema-based matching approaches. In: *Journal on Data Semantics* 4 (2005), S. 146–171. – URL `http://www.dit.unitn.it/~p2p/RelatedWork/Matching/JoDS-IV-2005_SurveyMatching-SE.pdf`. – Zugriffsdatum: 16.02.2009

[Sintek 2007] Sintek, Michael: *OntoViz tab: Visualization Protégé Ontologies*. Internetseite. 2007. – URL `http://protegewiki.stanford.edu/index.php/OntoViz`. – Zugriffsdatum: 30.05.2009

[Storey u. a. 2002] Storey, Margaret-Anne ; Noy, Natasha F. ; Musen, Mark ; Best, Casey ; Fergerson, Ray ; Ernst, Neil: Jambalaya: an interactive environment for exploring ontologies. In: *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*. New York, NY, USA : ACM, 2002, S. 239

[Stuart u. a. 1999] Stuart, Card K. ; Mackinlay, Jock D. ; Shneiderman, Ben: *Readings in information visualization: using vision to think*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999

[Su 2004] Su, Xiaomeng: *Semantic Enrichment for Ontology Mapping*. Norwegian University of Science and Technology, Dissertation, DEC 2004. – URL `http://www.idi.ntnu.no/grupper/su/publ/phd/xiaomeng-su-thesis.pdf`. – Zugriffsdatum: 14.02.2009

[Tufte 2001] Tufte, Edward R.: *The Visual Display of Quantitative Information*. Graphics Press, 2001. – ISBN 0961392142

[Unicode, Inc. 2008] Unicode, Inc.: *About the Unicode Standard*. Internetseite. 2008. – URL `http://www.unicode.org/standard/standard.html`. – Zugriffsdatum: 27.12.2008

[W3C u. a. 2006a] W3C ; Bray, Tim ; Hollander, Dave ; Layman, Andrew ; Tobin, Richard: *Namespaces in XML 1.0 (Second Edition)*. Internetseite. 2006. – URL http://www.w3.org/TR/REC-xml-names/. – Zugriffsdatum: 27.12.2008

[W3C u. a. 2006b] W3C ; Bray, Tim ; Paoli, Jean ; Sperberg-McQueen, C. M. ; Maler, Eve ; Yergeau, François: *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. Internetseite. 2006. – URL http://www.w3.org/TR/2006/REC-xml-20060816. – Zugriffsdatum: 27.12.2008

[W3C u. a. 2004a] W3C ; Brickley, Dan ; Guha, R.V.: *RDF Vocabulary Description Language 1.0: RDF Schema*. Internetseite. 2004. – URL http://www.w3.org/TR/rdf-schema/. – Zugriffsdatum: 03.01.2008

[W3C u. a. 2004b] W3C ; Fallside, David C. ; Walmsley, Priscilla: *XML Schema Part 0: Primer Second Edition*. Internetseite. 2004. – URL http://www.w3.org/TR/xmlschema-0/. – Zugriffsdatum: 27.12.2008

[W3C u. a. 2004c] W3C ; Manola, Frank ; Miller, Eric: *RDF Primer*. Internetseite. 2004. – URL http://www.w3.org/TR/rdf-primer/. – Zugriffsdatum: 30.12.2008

[Wang und Xu 2008] Wang, Peng ; Xu, Baowen: *Lily: Ontology Alignment Results for OAEI 2008*. 2008. – URL http://www.dit.unitn.it/~p2p/OM-2008/oaei08_paper7.pdf. – Zugriffsdatum: 01.03.2009

[Watts und Strogatz 1998] Watts, Duncan J. ; Strogatz, Steven H.: Collective dynamics of 'small-world' networks. In: *Nature* 393 (1998), June, Nr. 6684, S. 440–442. – ISSN 0028-0836

[Wu und Storey 2000] Wu, Jingwei ; Storey, Margaret-Anne D.: A multi-perspective software visualization environment. In: *Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, 2000, S. 15

[Yu 2007] Yu, Liyang: *Introduction to Semantic Web and Semantic Web Services*. 2007. – ISBN: 978-1584889335