# Skalierbarkeit von Ontology-Matching-Verfahren

Heiko Paulheim

MASTERARBEIT

eingereicht im Studiengang

Master of Science Informatik

an der TU Darmstadt

im Februar 2008

# Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Darmstadt, im Februar 2008

Heiko Paulheim

# Inhaltsverzeichnis

# 1. Einleitung

> *WARNING. Please note that the ontology size exceeds the capabilities of many current ontology tools (e.g. Jena, Protégé, vowlidator,...), and it is thus possible, depending on the configuration of your system, that trying to load the files will crash your system!*
>
> – Aus dem Benutzerhandbuch der eCl@ssOWL-Ontologie (Hepp, 2006, S. 2).

Im Jahr 2001 präsentierte eine Forschungsgruppe um einen der Mitbegründer des World Wide Web, Tim Berners-Lee, die Vision von der nächsten Generation des Internets: das Semantic Web, ein Netz von Informationen, die nicht nur von Menschen, sondern auch von Maschinen sinnvoll interpretiert und kombiniert werden können (Berners-Lee u. a., 2001). Ein zentraler Baustein dieses semantischen Netzes sind Ontologien, die die Bedeutung – eben die *Semantik* – von Begriffen definieren.

In den vergangenen Jahren wurde viel Forschung auf dem Gebiet semantischer Datenverarbeitung betrieben, die Idee der Ontologien auch auf Bereiche wie elektronischen Geschäftsverkehr und Web Services übertragen. Dabei wuchs zum einen die Anzahl der verfügbaren Ontologien, zum anderen deren Größe, d.h., die Menge des in einer Ontologie kodierten Wissens (Ding u. a., 2005, S. 65). Eine große Anzahl parallel genutzter Ontologien macht es nötig, diese aufeinander abzustimmen und Begriffe einer Ontologie in die andere zu übersetzen. Für diesen Zweck wurden sogenannte Ontology-Matching-Werkzeuge entwickelt. Die zunehmende Größe der Ontologien stellt zudem besondere Anforderungen an die Skalierbarkeit der Werkzeuge, mit denen diese Ontologien verarbeitet werden. In dieser Masterarbeit wird die Skalierbarkeit von Ontology-Matching-Werkzeugen untersucht.

## 1.1. Das Semantic Web

Wenn man im World Wide Web nach Informationen sucht, dann meist, um eine konkrete Frage zu beantworten – z.B. *Welche Literatur wurde im vergangenen Jahr zum Thema „Ontology Matching“ veröffentlicht?* Das World Wide Web kann diese Frage nicht direkt beantworten. Was man vielmehr tut, ist, in eine Suchmaschine (oder mehrere) einige Suchbegriffe einzugeben und in den vorgeschlagenen Seiten selbst nach der Antwort zu suchen.

Im Semantic Web dagegen, so die Vision der Autoren im Jahr 2001, suchen Maschinen Antworten auf Fragen. Anstelle einer Liste von mehr oder weniger passenden Webseiten zu einigen vom Nutzer eingegebenen Stichpunkten, die im besten Fall Indizien zur Beantwortung einer Frage enthalten, würde eine Suchmaschine im Semantic Web eine „echte“ Antwort auf „echte“ Fragen produzieren – im obigen Beispiel also eine Liste von Büchern und Artikeln (Berners-Lee u. a., 2001, S. 41 f.).

Dieses Ziel ist bislang noch nicht erreicht worden – noch immer finden Suchmaschinen Dokumente, deren Inhalt der Nutzer manuell durchsuchen muss, um seine Fragen selbst zu beantworten. Um das Semantic Web in der Form, wie es die Autoren in ihrer ursprünglichen Vision dargestellt haben, zu realisieren, müssten die Informationen im Internet für Maschinen lesbar und inhaltlich verarbeitbar hinterlegt werden – bislang jedoch besteht ein großer Teil des Internets aus Texten und Bildern, deren Inhalte nur von Menschen interpretierbar sind, von Maschinen jedoch nicht. Als Form der Wissensrepräsentation wurden *Ontologien* vorgeschlagen (Berners-Lee, 2006, S. 14). Ontologien sind semantische Netze, die Begriffe und ihre Zusammenhänge untereinander in einer formalisierten Art definieren. Mit Hilfe von Ontologien lassen sich Wissensbasen aufbauen, die auch von Maschinen genutzt werden können, um komplexe Anfragen zu beantworten.

Auch wenn die ganz große Vision des Semantic Web noch nicht Wirklichkeit geworden ist, so werden die in diesem Bereich entwickelten Technologien doch mittlerweile zumindest in kleineren Zusammenhängen – wie etwa der unternehmensinternen Datenintegration (Blumauer und Fundneider, 2006, S. 227 ff.), der verbesserten Nutzung von Web Services (Polleres u. a., 2006, S. 505 ff.) oder der Optimierung des elektronischen Geschäftsverkehrs (Rebstock u. a., 2008) – durchaus gewinnbringend eingesetzt.

## 1.2. Das Ontology-Matching-Problem

Wie auch im „herkömmlichen“ World Wide Web werden Inhalte im Semantic Web dezentral bereitgestellt (Tochtermann und Maurer, 2006, S. 3). Daher werden an verschiedenen Stellen unterschiedliche Ontologien entwickelt, die zur Auszeichnung der bereitgestellten Inhalte genutzt werden (Hendler, 2001, S. 31). Es ist unwahrscheinlich, dass es *die eine* Ontologie gibt, die sich weltweit durchsetzen wird. Vielmehr ist von einem dauerhaften Nebeneinander vieler Ontologien, die ähnliche Inhalte beschreiben, auszugehen (Fensel, 2004, S. 123 f.).

Berners-Lee u.a. erkannten dies bereits in ihrem Grundlagenartikel zum Semantic Web und schrieben, dass es durchaus möglich sei, auch verschiedene Ontologien zur Auszeichnung unterschiedlicher Webseiten zu verwenden. Um dennoch Interoperabilität zu gewährleisten, müssten in diesen Ontologien Verweise enthalten sein, die Entsprechungen eines Begriffs in anderen Ontologien definieren (Berners-Lee u. a., 2001, S. 41 f.). Nur so lassen sich Informationsquellen, die die Begriffe unterschiedlicher Ontologien verwenden, sinnvoll kombinieren. Solche Verweise werden *Ontology Mapping* genannt. In einer Umfrage im Jahr 2007 unter mehr als 600 Forschern und Anwendern von Semantic-Web-Technologien bezeichneten mehr als zwei Drittel der Befragten Ontology Mapping als wichtigste Technik im Umgang mit Ontologien (Cardoso, 2007, S. 87).

Oft sind diese Verweise jedoch nicht vorhanden – sei es, weil die Ontologien unabhängig voneinander entstehen (und deren Entwickler keine Kenntnis über alle anderen Ontologien haben), weil solche Verweise die Ontologien bei hinreichend vielen anderen Ontologien zu umfangreich machen würden, oder weil die Ontologien ursprünglich überhaupt nicht dafür entwickelt wurden, mit bestimmten anderen Ontologien gemeinsam eingesetzt zu werden. Daher müssen solche Verweise oft erst erzeugt werden, damit Interoperabilität hergestellt werden kann.

Solche Ontology Mappings können entweder manuell erstellt oder von automatisierten *Ontology-Matching*-Verfahren gefunden werden; für größere Datenmengen sind jedoch in den meisten Fällen ausschließlich automatisierte Verfahren praktikabel (Kalfoglou und Schorlemmer, 2005, S. 28). Um die Erstellung von Ontology Mappings so stark wie möglich zu vereinfachen, muss der Bedarf an menschlichen Eingriffen möglichst minimiert werden (Grobelnik und Mladenić, 2006, S. 13).

Das automatisierte Finden von Mappings ist noch immer einer der Hauptengpässe in der semantischen Integration (Noy, 2004, S. 66). Dabei stellt sich oft heraus, dass existierende Werkzeuge zum einen um Größenordnungen zu langsam sind, um in praktikabler Zeit sinnvolle Ergebnisse zu liefern (Giunchiglia und Shvaiko, 2003, S. 35). Zum anderen stellt die wachsende Größe der existierenden Ontologien neue Herausforderungen an die Entwicklung von Matching-Werkzeugen, die oft nicht mit großen Ontologien umgehen können (Hu u. a., 2006b, S. 72 f.).

Die Skalierbarkeit von Semantic-Web-Anwendungen im Allgemeinen und von Ontology-Matching-Verfahren im Speziellen spielte lange Zeit eine eher untergeordnete Rolle in der Semantic-Web-Forschung. Demzufolge zeigt sich in der Praxis oft, dass die verfügbaren Werkzeuge auf moderaten Datenmengen gute Ergebnisse produzieren, vor Problemen höherer Größenordnungen, etwa im Umgang mit sehr großen Ontologien, jedoch kapitulieren (Euzenat u. a., 2008, S. 184 f.). Erst in jüngerer Zeit wurde die herausragende Wichtigkeit der Skalierbarkeit erkannt (vgl. Hepp, 2008, S. 18).

## 1.3. Aufbau der Arbeit

Damit auch große Ontologien sinnvoll miteinander kombiniert werden können, ist es unerlässlich, dass skalierbare automatisierte Ontology-Matching-Systeme entwickelt werden. Diese Arbeit beschäftigt sich mit der Frage, wie sich bestehende Matching-Systeme auch mit großen Ontologien nutzen lassen.

In Kapitel 2 werden Ontologien einführend behandelt. Neben informellen und formalen Definitionen werden die wichtigsten Sprachen zur Beschreibung von Ontologien – RDF, RDF-S und OWL – vorgestellt. Anschließend wird ein Überblick über Einsatzbereiche und typische Werkzeuge zur Arbeit mit Ontologien gegeben.

Kapitel 3 beschäftigt sich mit dem Ontology-Matching-Problem. Nach einer Einführung gängiger Definitionen und Beschreibungsformalismen werden exemplarische Einsatzgebiete von Ontology Matching aufgezeigt. Anschließend werden Verfahren und einige bekannte Werkzeuge vorgestellt, die beim Ontology Matching zum Einsatz kommen, und Maße für ihre Qualität eingeführt. Eine kurze Untersuchung der Skalierbarkeit der vorgestellten Werkzeuge beendet das Kapitel.

Da die in Kapitel 3 untersuchten Werkzeuge nur sehr schlecht mit großen Ontologien umgehen können, wurde im Rahmen dieser Arbeit eine Lösung entwickelt, die auch auf große Ontologien anwendbar ist. Diese wird in Kapitel 4 beschrieben. Hauptmerkmale dieser Lösung sind die Verwendung von Partitionierungstechniken und die Möglichkeit, bestehende, nichtskalierbare Matching-Werkzeuge so einzusetzen, dass sie auch auf große Ontologien anwendbar sind. Nach der Erläuterung der Grundideen und der eingesetzten Techniken wird ein Überblick

über die Systemarchitektur gegeben. Daran anknüpfend wird die Qualität der entwickelten Lösung untersucht und verschiedene Optimierungsansätze aufgezeigt.

Die Arbeit schließt mit einem Ausblick, in dem weitere Ausbaumöglichkeiten der in Kapitel 4 vorgestellten Lösung sowie einige verwandte Probleme dargestellt werden.

# 2. Ontologien

Der Begriff *Ontologie* kommt vom Griechischen $\mathit{ovτος}$ ('Sein') und $\lambda\mathit{ογος}$ ('Lehre', 'Wissenschaft'). Die Ontologie ist ein Teilgebiet der Philosophie, das sich mit der Lehre vom Sein und der Existenz beschäftigt (Kunzmann u. a., 2003, S. 13).

In der Informatik ist eine Ontologie eine formale Beschreibung einer Begriffswelt. In einer Ontologie soll Information so repräsentiert werden, dass Computer damit in einer sinnvollen Weise arbeiten können (Hitzler u. a., 2008, S. 12).

## 2.1. Basistechnologien des Semantic Web

Ontologien sind insbesondere im Forschungsbereich des Semantic Web populär geworden. Hier wurden in den vergangenen Jahren Grundlagentechnologien entwickelt, die in einigen Bereichen, insbesondere in der Organisation des Wissens in der Forschung, etwa zur Verwaltung großer Mengen medizinischer Daten, genutzt werden (Shadbolt u. a., 2006, S. 96 ff.). Abbildung 2.1 zeigt einen Überblick der Technologien, wie sie im Semantic Web zum Einsatz kommen, den sogenannten *Semantic Web Stack*.

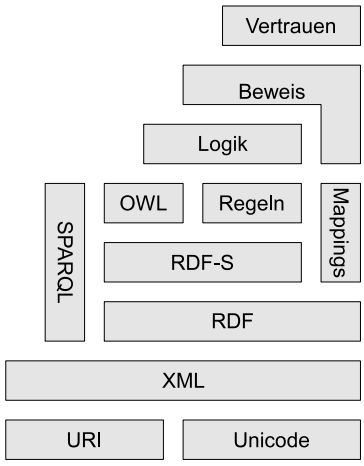



**Abbildung 2.1.:** Der Semantic Web Stack (in Anlehnung an Berners-Lee, 2006, S. 14)

Die wichtigsten Basistechnologien, die im Semantic Web eine Rolle spielen, sind die folgenden:

- Unicode und URIs bilden die syntaktische Grundlage des Semantic Webs. Unicode ist ein Zeichensatz, mit dem sich alle Sprachen der Welt darstellen lassen (Unicode Consortium, 2007). Damit liegt dem Semantic Web eine eindeutige Codierung zu Grunde, die keinen Sprachgrenzen unterliegt.

URIs (Uniform Resource Identifiers) sind Zeichenketten, die physikalischen oder abstrakten Einheiten zugeordnet werden, um diese zu benennen (Berners-Lee u. a., 1998). Sie dienen dazu, die Konzepte, über die im Semantic Web Informationen hinterlegt sind, eindeutig identifizieren zu können.

- XML (eXtensible Markup Language) ist eine Auszeichnungssprache, mit der Daten kodiert werden können. XML-Dokumente sind sowohl von Menschen als auch von Maschinen lesbar (W3C, 2006a). Nutzer von XML haben die Möglichkeit, ihre eigenen Sprachmittel (*Tags* genannt) zur Auszeichnung von Daten zu definieren. In Form von XML werden Daten im Semantic Web kodiert.

- RDF (Resource Description Framework) wird genutzt, um einfache Aussagen zu treffen und so Information darzustellen (W3C, 2004f). Es bildet die Grundlage dafür, Informationen im Semantic Web auszutauschen. RDF wird in Kapitel 2.3.1 beschrieben.

- RDF-S (RDF-Schema, W3C, 2004d) und OWL (Web Ontology Language, W3C, 2004c) werden genutzt, um die Bedeutung der in RDF-Aussagen genutzten Begriffe festzulegen und weitere Informationen über die Beziehungen dieser Begriffe untereinander darzustellen. Solche Sammlungen von Begriffsdefinitionen nennt man *Ontologien*. Die Begriffsdefinitionen lassen sich durch zusätzliche Regeln erweitern, die weiterführende Informationen über die Beziehungen zwischen den Einheiten, welche in Ontologien definiert sind, enthalten. Eine W3C-Arbeitsgruppe entwickelt derzeit einen neuen, umfassenden Standard zur Kodierung solcher Regeln (W3C, 2005a). RDF-S und OWL werden in den Kapiteln 2.3.2 und 2.3.3 beschrieben.

- Mappings sind Verweise zwischen verschiedenen Ontologien. Sie sind nötig, um Informationen aus verschiedenen Quellen kombinieren zu können. Mappings können auf verschiedene Arten beschrieben werden; ein Überblick darüber wird in Kapitel 3.1.4 gegeben.

- SPARQL (SPARQL Protocol and RDF Query Language[1], W3C, 2008) ist eine Sprache, mit der sich Abfragen auf RDF-Daten und Ontologien formulieren lassen (ähnlich wie SQL Abfragen auf relationalen Datenbanken ermöglicht). Erst durch eine solche Abfragesprache werden die im Semantic Web hinterlegten Informationen sinnvoll nutzbar.

- Auf der Basis dieser Informationen – RDF-Daten und Ontologien – lassen sich mit Hilfe von Schlussmechanismen der Logik auch komplexere Abfragen beantworten. Mit diesen Mechanismen kann auch Wissen explizit gemacht werden, das zuvor nur implizit in Ontologien enthalten war. Indem die Schlussregeln, die zu einer Antwort auf eine Abfrage führten, transparent gemacht werden, wird dem Nutzer auch ein Beweis für die Antwort einer semantischen Suchmaschine gegeben. Diese Schlussverfahren werden *Inferenz-* oder *Reasoning*-Verfahren genannt.

- Da zur Antwort auf eine Frage verschiedene Informationsquellen im Semantic Web herangezogen werden können, kann es schließlich noch sinnvoll sein, die Vertrauenswürdigkeit

[1] Bei *SPARQL* handelt es sich um ein rekursives Akronym.

der einzelnen Quellen zu untersuchen und daraus einen Gesamtwert der Vertrauenswürdigkeit der Antwort errechnen. Damit lässt sich eine Abfrage nicht nur beantworten, sondern zusätzlich mit einem Indikator versehen, wie glaubwürdig diese Antwort ist.

In der Semantic-Web-Forschung spielen insbesondere die mittleren Schichten des Semantic-Web-Stacks eine große Rolle. RDF, RDF-Schema und OWL haben sich zu einem weit verbreiteten Mittel zur Beschreibung von Ontologien entwickelt.

## 2.2. Ontologie-Definitionen

Für Ontologien wurden im Laufe der Zeit verschiedene Definitionen entwickelt. Diese umfassen zum einen informelle Formulierungen, zum anderen auch formale Definitionen.

### 2.2.1. Informelle Definitionen

Hitzler u. a. (2008, S. 12) bezeichnen eine Ontologie als ein „Dokument, welches Wissen einer Anwendungsdomäne modelliert“. Solches Wissen besteht zunächst aus Begriffen, einem *Vokabular*. Ontologien dienen dazu, die Bedeutung eines Vokabulars zu definieren, indem festgehalten wird, wie Begriffe aus einem Vokabular untereinander in Beziehung stehen (Daconta u. a., 2003, S. 181 f.). Eine häufig zitierte Definition stammt von Gruber (1993, S. 199): „An ontology is an explicit specification of a conceptualization“ – eine Ontologie spezifiziert eine Vorstellung, und zwar in einer Form, die von einer Maschine verarbeitet werden kann (Daconta u. a., 2003, S. 181 f.). Dazu enthalten Ontologien neben den Begriffen und ihren Beziehungen auch einfache Schlussfolgerungsregeln, die die automatisierte Verarbeitung der Ontologien erleichtern (Hendler, 2001, S. 30).

Von *Datenbanken*, die ebenfalls Wissen speichern, unterscheiden sich Ontologien dadurch, dass in Datenbanken nur Instanzen in festen, vordefinierten Strukturen gespeichert werden. Das Wissen darüber, zu welchen Klassen diese Instanzen gehören und wie diese Klassen miteinander zusammenhängen, ist in einer Datenbank nicht explizit gespeichert. *Wissensbasen* benötigen dagegen keine vordefinierten Strukturen. Reichert man Wissensbasen um Metainformationen über die in ihnen gespeicherten Konzepte an, so spricht man von Ontologien (Witbrock, 2007, S. 2).

Ein System von Begriffen, die durch Zusammenhänge verbunden sind, wird auch als *semantisches Netz* bezeichnet[2]. Die Begriffe der Ontologie können dabei sowohl in den Knoten als auch in den Kanten des semantischen Netzes stehen (Sowa, 2000, S. 51). Das wahrscheinlich erste semantische Netz der Geschichte, das in Abbildung 2.2 dargestellt ist, stammt vom griechischen Philosophen Porphyrios aus dem dritten Jahrhundert vor Christus (Sowa, 2000, S. 4).

Je nach Formalisierungsgrad und Ausdrucksmächtigkeit kann man unterschiedliche Ausprägungen von semantischen Netzen betrachten. Dabei lässt sich mit einem höheren Formalisierungsgrad auch eine größere Ausdrucksmächtigkeit erreichen. Abbildung 2.3 zeigt verschiedene Arten semantischer Netze:

[2]Dabei handelt es sich *nicht* um das Semantic Web, von dem es nur eines gibt, sondern um eine von vielen möglichen Darstellungen von Begriffszusammenhängen.

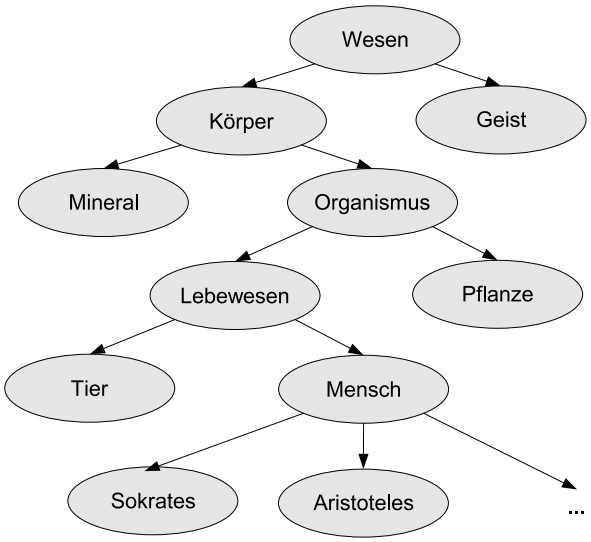



**Abbildung 2.2.:** Der Baum des Porphyrios (nach Sowa, 2000, S. 5)

- *Glossare* erklären Begriffe. Dabei ist jedem Begriff in der Regel nur eine natürlichsprachliche Erklärung hinzugefügt, eine Vernetzung der Begriffe untereinander existiert meist nicht.

- *Taxonomien* sind hierarchische Begriffsanordnungen. In diesen werden Begriffe mit nur einer Relation verbunden. Dies kann z.B. eine Subkonzept- oder eine Teil-von-Relation sein (Daconta u. a., 2003, S. 158). Der oben abgebildete Baum des Porphyrios ist eine solche Taxonomie[3].

- *Thesauri* erlauben weitere Verbindungen zwischen Begriffen, z.B. Äquivalenz- oder Assoziationsbeziehungen, wie z.B. *„PKW“ ist ein Synonym zu „Auto“*, und *„Auto“ ist mit „Straße“ assoziiert* (Daconta u. a., 2003, S. 159 ff.). Der ISO-Standard 2788 definiert verschiedene erlaubte Beziehungen zwischen Begriffen in Thesauri (ISO, 1986).

- *Topic Maps* beschreiben Themen, die in Dokumenten enthalten sind. Sie sind in erster Linie als Metadaten zu Dokumentensammlungen konzipiert, um diese besser durchsuchbar und nutzbar zu machen. Dabei werden auch Beziehungen zwischen den Themen erfasst (Daconta u. a., 2003, S. 167 f.). Im ISO/IEC-Standard 13250 werden dazu Konzepte wie Themen, Begriffe und Themenbeziehungen definiert. Sogenannte „Scopes“ ermöglichen unterschiedliche Sichten auf dieselben Themen (ISO/IEC, 2002).

- *Ontologien* sind schließlich die reichhaltigsten semantischen Netze. Sie können als Wissenssammlungen im weitesten Sinne aufgefasst werden. Von den weniger formalisierten Netztypen grenzen sie sich dadurch ab, dass aus dem in Ontologien beschriebenen Wissen logische Schlüsse gezogen werden können (Blumauer und Pellegrini, 2006, S. 16).

[3]Streng genommen existieren in diesem Baum zwei verschiedene Relationsarten: die oberen fünf Ebenen sind durch eine Subkonzept-Relation, die unteren beiden dagegen durch eine Instanz-Relation verbunden.

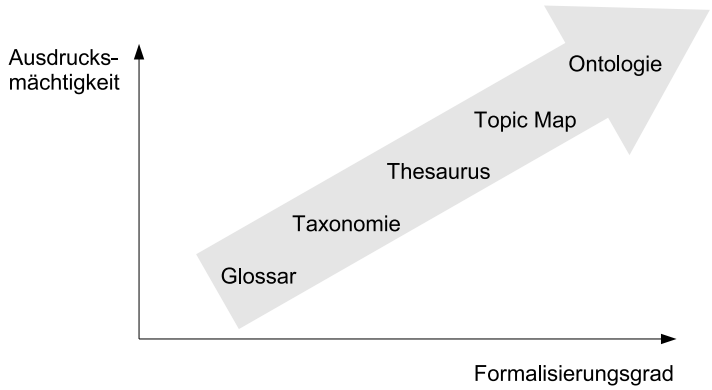



**Abbildung 2.3.:** Arten semantischer Netze (nach Blumauer und Pellegrini, 2006, S. 16)

Die in diesem Modell als „Ontologien“ bezeichneten Formen werden manchmal auch als „Heavy-weight Ontologies“, die darunter angesiedelten Formen als „Light-weight Ontologies“ bezeichnet (Gómez-Pérez u. a., 2004, S. 8).

Ontologien können auch nach dem Gebiet unterschieden werden, dessen Wissen in ihnen modelliert ist (Fensel, 2004, S. 5 f.):

- *Domänenontologien* beschreiben das Wissen einer bestimmten Anwendungsdomäne, z.B. elektronischen Handel, Genetik oder Tourismus.
- Dagegen versuchen *Upper-Level-Ontologien*, auch *Generic Ontologies* oder *Common Sense Ontologies* genannt, einen möglichst breit gestreuten Teil des Allgemeinwissens zu erfassen. Sie können als Grundlagen genutzt werden, um Domänenontologien geringerer Granularität zu definieren (Blumauer und Pellegrini, 2006, S. 17). Damit kann zwischen verschiedenen Domänenontologien ein Minimalkonsens gemeinsamer Begriffe gewährleistet werden (Noy, 2004, S. 66). Beispiele für Upper-Level-Ontologien sind SUMO (**S**uggested **U**pper **M**erged **O**ntology, Niles und Pease, 2001, S. 2 ff.), OpenCyc (von **Open** En**cyc**lopedia, Fischer, 2004) und DOLCE (**D**escriptive **O**ntology for **L**inguistic and **C**ognitive **E**ngineering, Masolo u. a., 2003).
- *Metadatenontologien* dienen zur allgemeinen Beschreibung von Daten und Dokumenten. Im Gegensatz zu Domänenontologien enthalten sie hauptsächlich domänenübergreifende Konzepte, wie z.B. `Autor`, `Erstellungsdatum` usw.
- *Repräsentationale Ontologien* stellen Grundbegriffe zur Verfügung, die zur Definition anderer Ontologien genutzt werden können, wie etwa *Klassen* und *Beziehungen*. Sie können daher als eine Art Metaontologie verstanden werden. Eine besondere Form dieser Ontologien sind *Methoden- und Aufgabenontologien*, die Grundbegriffe für bestimmte Klassen von Aufgaben definieren, ohne dabei auf deren eigentliche Inhalte einzugehen. So kann eine Aufgabenontologie für wissenschaftliche Experimente Konzepte wie *Beobachtung* und *Hypothese* definieren, ohne dass bekannt sein muss, ob es sich hierbei um medizinische, physikalische oder chemische Experimente handelt.

Sowa (2000, S. 15 f., S. 51) unterscheidet bei den in Ontologien enthaltenen Konzepten domänenspezifische und domänenunabhängige Prädikate. Dabei bildet die Logik die domänenunab-

hängigen Grundlagen – etwa, dass es Klassen und Subklassen gibt und die Subklassenbeziehung transitiv ist –, zu denen dann domänenspezifische Konzepte treten können. Mit Hilfe der Logik können Aussagen über die Existenz oder Nicht-Existenz von Instanzen eines Konzeptes gemacht werden; im domänenspezifischen Teil der Ontologie werden diese Konzepte und deren Eigenschaften definiert. Ein *Ontological Commitment* ist eine Sicht auf einen Weltausschnitt (Davis u. a., 1993, S. 5), es definiert die Kategorien von Dingen, die in einer Anwendungsdomäne existieren oder existieren können (Sowa, 2000, S. 134) und damit ein mögliches Modell einer Domäne.

### 2.2.2. Formale Definitionen

Die einfachste formale Definition von Kalfoglou und Schorlemmer (2005, S. 3) bezeichnet eine Ontologie $O$ als 2-Tupel

$$O := \langle S, A \rangle . \tag{2.1}$$

$S$ ist dabei die *Signatur*, die das Vokabular der Ontologie beschreibt, $A$ beschreibt die Menge der Aussagen über die Interpretation dieses Vokabulars, die *Axiome*.

Es gibt verschiedene Erweiterungen dieser Definition. Scharffe und de Bruijn (2005, S. 267) teilen die Signatur $S$ in drei Bereiche und definieren eine Ontologie als ein 4-Tupel

$$O := \langle C, R, I, A \rangle , \tag{2.2}$$

wobei $C$ eine Menge von Konzepten oder Klassen, $R$ eine Menge von Relationen, die zwischen diesen Konzepten bestehen können, $I$ eine Menge von Instanzen der Klassen und $A$ wieder die Menge der Axiome ist. Ontologien, die eine nicht-leere Menge von Instanzen enthalten, heißen auch *bestückte Ontologien* (Kalfoglou und Schorlemmer, 2005, S. 3).

Eine noch detailliertere Definition stammt von Ehrig und Sure (2004, S. 77). Sie erweitern eine Ontologie um Hierarchiebeziehungen und definieren

$$O := \langle C, \leq_C, R_C, \leq_R, I, R_I, A \rangle . \tag{2.3}$$

In dieser Definition wird angenommen, dass sowohl Klassen als auch Relationen zwischen Klassen in Hierarchiebeziehungen $\leq_C$ und $\leq_R$ stehen, die eine *partielle Ordnung* bilden (Ehrig, 2007b, S. 12). In einer Klassenhierarchie können z.B. `Angestellter` und `Arbeiter` Unterklassen von `Arbeitnehmer` sein, eine Relation `arbeitetBei` kann Unterrelationen `istAngestelltBei`, `istGeschaeftsfuehrerVon` usw. haben. Außerdem unterscheiden Ehrig und Sure zwischen der Definition einer Relation zwischen Klassen und den konkreten Instanzen dieser Relationen: die Relation `istAngestelltBei`, die zwischen `Arbeiter` und `Unternehmen` besteht, kann eine konkrete Ausprägung zwischen den Instanzen `HansMustermann` und `MusterGmbH` haben.

Eine ähnliche Erweiterung schlägt Ehrig (2007b, S. 12 ff.) vor, der eine Ontologie definiert als

$$O := \langle C, \leq_C, R, \leq_R, \sigma, I, \iota_I, \iota_R, A \rangle \tag{2.4}$$

In Erweiterung der Definition von Ehrig und Sure wird die *Signatur*-Funktion $\sigma : R \to C \times C$ eingeführt, die jeder Relation einen Definitions- und einen Wertebereich zuweist. Die Funktionen $\iota_I : C \to 2^I$ und $\iota_R : R \to 2^{I \times I}$ (wobei $\iota_R(r) \subseteq \sigma(r)$ gelten muss) heißen *Instanziierungs-*

*funktionen*[4]; sie definieren Instanzen von Klassen und Relationen. Weiterhin unterscheidet Ehrig die Begriffe *Core Ontology* und *Knowledge Base*: Die Core Ontology besteht aus den Definitionen für Konzepte und Relationen, die Knowledge Base enthält die konkreten Instanzen. Ein gebräuchlicheres Begriffspaar für diese Unterscheidung ist *T-Box* für die Definition der Terminologie, das sogenannte *terminologische Wissen* – hierzu gehören $C$, $\leq_C$, $R$, $\leq_R$ und $\sigma$ – und *A-Box* für die Instanzen, das sogenannte *assertionale Wissen* – hierzu gehören dann $I$, $\iota_I$, $\iota_R$ und $A$ (May, 2006, S. 493 f.).

In einigen Publikationen werden auch elementare Datentypen (wie Zahlen oder Zeichenketten) gesondert in die Ontologie-Definitionen mit aufgenommen (vgl. z.B. Ehrig u. a., 2005, S. 1511); sie können jedoch auch als Sonderfälle der Konzepte betrachtet werden, was eine gesonderte Aufführung überflüssig macht (Ehrig, 2007b, S. 13).

Maedche und Staab (2002, S. 252ff) und Bloehdorn u. a. (2005, S. 10) unterscheiden zwischen den Konzepten, Relationen und Instanzen einer Ontologie, wie sie oben definiert sind, und deren Bezeichnern, und führen zusätzliche Abbildungen zwischen der Signatur der Ontologie und einem *Lexikon* ein, das die Bezeichner enthält. Ein Lexikon wird von Bloehdorn u.a. definiert als

$$L := \langle S_C, S_R, S_I, Ref_C, Ref_R, Ref_I \rangle \tag{2.5}$$

wobei $S_C$ die Menge aller Bezeichner für Konzepte ist und $Ref_C \subseteq C \times S_C$ die Zuordnung von Konzepten zu Bezeichnern. Jedes Konzept kann damit mehrere Bezeichner haben, die parallel verwendet werden können, um z.B. mehrsprachige Ontologien zu beschreiben. $S_R$, $Ref_R$, $S_I$ und $Ref_I$ sind analog definiert.

Diese Definitionen orientieren sich an den Mengen der Konzepte, Axiome etc. Es ist auch möglich, Ontologien, genauso wie Klassifikationen und Datenbankschemata, als gerichtete Graphen aufzufassen (Giunchiglia und Shvaiko, 2003, S. 265f). Dabei bilden die Konzepte und Instanzen Knoten, die durch Relationen und ihre Instanzen als Kanten verbunden sind. Je nach Anwendungsfall kann die mengen- oder die graphenorientierte Betrachtungsweise sinnvoller sein: so ist z.B. für die Speicherung von Ontologien die mengenorientierte Betrachtung sinnvoller, für die Visualisierung die graphenorientierte.

### 2.2.3. Definitionen für Beschreibungslogiken

Neben der mengen- und der graphenorientierten Beschreibung von Ontologien ist es auch gebräuchlich, Ontologien als *Beschreibungslogiken* zu betrachten. Beschreibungslogiken sind fest definierte Untermengen der Prädikatenlogik erster Stufe (Hitzler u. a., 2008, S. 163 f.), die bestimmte Sprachkonstrukte bereithalten. Je nach Umfang sind diese Logiken unterschiedlich mächtig.

Legt man die Ontologiedefinition 2.2 zugrunde, so entspricht $C$ der Menge der einstelligen Prädikate, $R$ der Menge der mehrstelligen Prädikate und $I$ der Menge der Konstanten in der Prädikatenlogik (Hitzler u. a., 2008, S. 164). Die Menge der Axiome $A$ wird dann durch eine Menge von prädikatenlogischen Aussagen mit diesen Konstrukten gebildet.

[4] Engl. *instantiation function*. Ob die deutsche Schreibung korrekt *Instanziierung* (15.700 Treffer bei Google), *Istanzierung* (14.300 Treffer) oder *Instantiierung* (10.100 Treffer) lautet, ist ungeklärt.

Eine einfache Beschreibungslogik ist $\mathcal{ALC}$. $\mathcal{ALC}$ steht für *Attributive Language with Complement*. In dieser Beschreibungslogik stehen Sprachkonstrukte für die Definition von Konzepten und Relationen (die in der Beschreibungslogik meist als *Rollen* bezeichnet werden), die bezüglich Existenz und Wertebereich eingeschränkt sein können, für deren Hierarchien, sowie für Schnitt- und Komplementmengen. Fügt man die Möglichkeit der Definition von transitiven Rollen hinzu, so nennt man die sich ergebende Beschreibungslogik $\mathcal{ALC}_{R^+}$ (Gómez-Pérez u. a., 2004, S. 17).

Die Beschreibungslogik $\mathcal{ALC}_{R^+}$ lässt sich auf verschiedene Arten erweitern. Die so entstehenden Beschreibungslogiken werden nach folgendem Schema benannt: Jede Erweiterung wird mit einem Buchstaben bezeichnet und der Name durch Aneinanderreihung der Buchstaben der kombinierten Konzepte gebildet (vgl. Hitzler u. a., 2008, S. 172, Horrocks u. a., 2003, und Gómez-Pérez u. a., 2004, S. 17 ff.). Dabei bezeichnet der Buchstabe:

- $\mathcal{S}$ das Äquivalent zu $\mathcal{ALC}_{R^+}$.
- $\mathcal{H}$ die Möglichkeit, Hierarchien von Relationen zu definieren.
- $\mathcal{O}$ die Möglichkeit, Klassen durch die Aufzählung ihrer Instanzen zu definieren.
- $\mathcal{I}$ die Möglichkeit, Relationen als Inverse anderer Relationen zu definieren.
- $\mathcal{F}$ die Möglichkeit, funktionale Relationen[5] zu definieren.
- $\mathcal{N}$ die Möglichkeit, nicht qualifizierende Aussagen über Restriktionen von Rollen zu treffen, also Aussagen der Form *Bei einer Prüfung gibt es mindestens zwei Prüfer* zu treffen.
- $\mathcal{Q}$ die Möglichkeit, auch qualifizierende Aussage über Restriktionen von Rollen zu treffen, also Aussagen der Form *Bei einer Prüfung gibt es mindestens einen Prüfer, der ein Professor ist* zu treffen.

Lässt man zusätzlich zu den genannten Rollen auch elementare Datentypen (wie *int* oder $float$) zu, so wird das Kürzel $(D)$ angehängt. Häufig verwendete Beschreibungslogiken sind z.B. $\mathcal{SHIQ}$ und $\mathcal{SHOIN}(D)$.

Beschreibungslogiken dienen dazu, Aussagen über die Ausdrucksmächtigkeit von Ontologiesprachen machen zu können. Kann man beweisen, dass eine Ontologiesprache zu einer bestimmten Beschreibungslogik äquivalent ist, so lassen sich z.B. Aussagen über die Mächtigkeit dieser Sprache und über die Komplexität der Beantwortung von Abfragen von Daten in einer Ontologie in dieser Sprache treffen (Hitzler u. a., 2008, S. 163 ff.).

## 2.3. Beschreibungs- und Repräsentationssprachen

Es gibt verschiedene Sprachen, mit denen sich Ontologien darstellen lassen. Die mit Abstand am häufigsten eingesetzten sind RDF-S und OWL (Cardoso, 2007, S. 85). Beide basieren auf der Sprache RDF. Sowohl RDF als auch RDF-S und OWL sind als W3C-Standards verabschiedet.

[5] Ist eine Relation $r$ funktional, so bedeutet dies, dass aus $r(A) = X$ und $r(B) = X$ folgt, dass $A = B$ gilt (Hitzler u. a., 2008, S. 151). Ein Beispiel für eine funktionale Relation ist die Zuweisung einer Sozialversicherungsnummer zu einer Person.

### 2.3.1. Die Basis: RDF

RDF (Resource Description Framework, W3C, 2004f) ist eine einfache Sprache zur Definition von Ontologien. Mit RDF lassen sich Informationen in einer Form kodieren, die für Menschen und Maschinen gleichermaßen verarbeitbar ist.

Die kleinsten Einheiten von RDF sind Aussagen, auch *Statements* genannt. Eine Aussage besteht, wie ein einfacher natürlichsprachlicher Satz, aus einem Subjekt, einem Prädikat und einem Objekt. Subjekte und Objekte sind dabei meist Ressourcen, also im weitesten Sinne physikalische Dinge oder abstrakte Konzepte, über die Aussagen gemacht werden (McBride, 2004, S. 51).

Verwendet man das Objekt eines Satzes als Subjekt eines anderen, so lassen sich Zusammenhänge zwischen mehreren Ressourcen modellieren. Fasst man jedes Subjekt und jedes Objekt als einen beschrifteten Knoten und jedes Prädikat als eine gerichtete, beschriftete Kante (zwischen Subjekt und Objekt) auf, so lässt sich aus einer Menge von RDF-Aussagen auch als Graph darstellen. Abbildung 2.4 zeigt einen solchen Beispielgraphen.

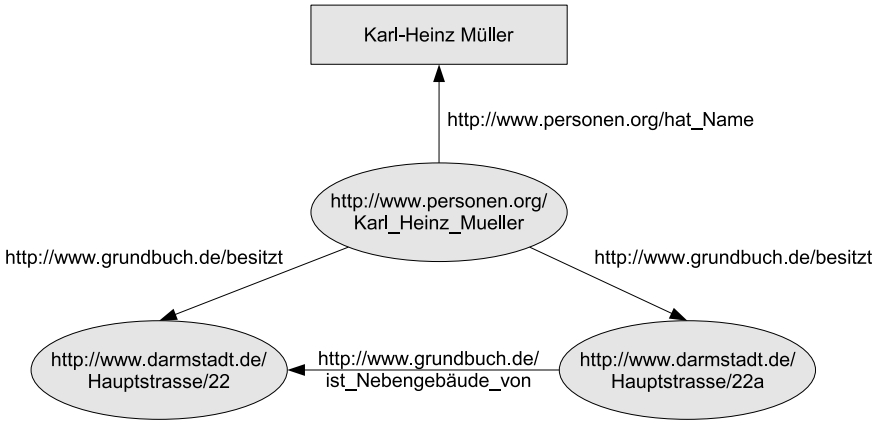



**Abbildung 2.4.:** Ein Beispiel für einen RDF-Graphen

Dieses Beispiel sagt aus, dass einer Person namens Karl-Heinz Müller die beiden Gebäude Hauptstraße 22 und Hauptstraße 22a in Darmstadt gehören, wobei das Gebäude 22a ein Nebengebäude von Gebäude 22 ist.

Subjekte und Prädikate in RDF sind stets URIs, die für bestimmte Ressourcen stehen – auch Prädikate sind damit Ressourcen im Sinne von RDF, die meist abstrakte Konzepte, wie im obigen Beispiel ein Besitzverhältnis, bezeichnen. Als Objekte sind URIs (im Graphen als Ovale gezeichnet) oder Literale (im Graphen als Rechtecke gezeichnet), also einfache Werte wie z.B. Strings (im Beispiel der Name „Karl-Heinz Müller“) oder numerische Werte zugelassen. Literale können auch Datentypen haben, z.B. die in XSD (XML Schema Definition) festgelegten (siehe W3C, 2006c). Sie können auch mit einem Sprachattribut versehen sein, so dass z.B. Bezeichner in mehreren Sprachen hinterlegt sein können. Ein Literal kann jedoch nicht sowohl einen Datentyp als auch ein Sprachattribut besitzen (Hitzler u. a., 2008, S. 54). Die Graphen, die sich aus RDF-Aussagenmengen ergeben, können auch Zyklen enthalten und sind damit, im Gegensatz z.B. zu XML-Dokumenten, keine Bäume (Hitzler u. a., 2008, S. 36 ff.).

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:personen="http://www.personen.org/"
         xmlns:grundbuch="http://www.grundbuch.de/">

<rdf:description rdf:about="http://www.personen.org/Karl_Heinz_Mueller">
  <personen:hat_Name>Karl-Heinz Müller</personen:hat_Name>
  <grundbuch:besitzt>
    <rdf:description rdf:about="http://www.darmstadt.de/Hauptstrasse/22"/>
  </grundbuch:besitzt>
  <grundbuch:besitzt>
    <rdf:description rdf:about="http://www.darmstadt.de/Hauptstrasse/22a"/>
  </grundbuch:besitzt>
</rdf:description>

<rdf:description rdf:about="http://www.darmstadt.de/Hauptstrasse/22a">
  <grundbuch:ist_NebengebäudeVon>
    <rdf:description rdf:about="http://www.darmstadt.de/Hauptstrasse/22"/>
  </grundbuch:ist_NebengebäudeVon>
</rdf:description>

</rdf:RDF>
```

**Quelltextbeispiel 2.1:** Beispiel RDF-XML

```
@prefix personen <http://www.personen.org/> .
@prefix grundbuch <http://www.grundbuch.de/> .

personen:Karl_Heinz_Mueller personen:hat_Name "Karl-Heinz Müller" .
personen:Karl_Heinz_Mueller grundbuch:besitzt <http://www.darmstadt.de/Hauptstrasse/22> ,
                                              <http://www.darmstadt.de/Hauptstrasse/22a> .
<http://www.darmstadt.de/Hauptstrasse/22a> grundbuch:istNebengebäudeVon
                                           <http://www.darmstadt.de/Hauptstrasse/22> .
```

**Quelltextbeispiel 2.2:** Beispiel RDF in Tripel-Notation

Um RDF-Dokumente zu speichern, haben sich zwei Formate etabliert: die XML-basierte Notation RDF-XML[6], die auch ein W3C-Standard ist (W3C, 2004e), und die kompaktere Tripel-Notation. Quelltextbeispiel 2.1 zeigt das obige Beispiel in RDF-XML, Beispiel 2.2 dasselbe Beispiel in Tripel-Syntax. Die Transformation eines Graphen in eine der beiden Notationen ist nicht eindeutig; zum einen ist die Reihenfolge der Tripel beliebig, zum anderen hat man meist in jeder Notation mehrere Möglichkeiten, eine Menge von Tripeln darzustellen, etwa durch Zusammenfassung, wie im Beispiel 2.2 an den beiden Tripeln mit Prädikat `grundbuch:besitzt` gezeigt (Hitzler u. a., 2008, S. 45). Umgekehrt lässt sich jedoch jeder Menge von Tripeln und jedem RDF-XML-Dokument eindeutig ein Graph zuordnen.

RDF-Tripel lassen sich nur verwenden, um binäre Beziehungen darzustellen, z.B. *Karl-Heinz Müller besitzt das Haus Hauptstrasse 22.* Zusätzlich gibt es auch Sprachmittel, um Listen von Daten eines Datentyps zu definieren.

[6]Obwohl der in einer XML-Datei kodierte RDF-Graph kein Baum sein muss, ist das XML-Dokument, das diesen Graphen speichert, sehr wohl ein Baum (Hitzler u. a., 2008, S. 42 f.)!

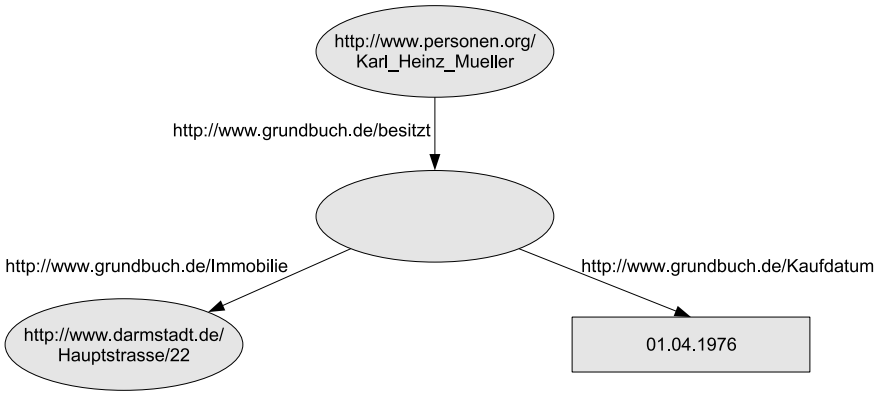



**Abbildung 2.5.:** RDF-Graph mit leerem Knoten

Um auch höherwertige Beziehungen zwischen Daten verschiedenen Typs darzustellen, müssen zusätzliche Knoten eingefügt werden. Diese brauchen allerdings keinen eigenen URI, sondern können als sogenannte *Blank Nodes* oder *Leere Knoten*, oft auch *anonyme Knoten* genannt, genutzt werden (Hitzler u. a., 2008, S. 56 ff.). Abbildung 2.5 zeigt einen Graphen mit leerem Knoten, der die Aussage *Karl-Heinz-Müller hat das Haus Hauptstrasse 22 am 01.04.1976 erworben* darstellt, Quelltext-Beispiel 2.3 diesen Graphen in Tripel-Syntax (der Unterstrich wird verwendet, um an leere Knoten temporäre URIs zu vergeben).

```
@prefix personen <http://www.personen.org/> .
@prefix grundbuch <http://www.grundbuch.de/> .

personen:Karl_Heinz_Mueller grundbuch:besitzt _:id1 .
_:id1 grundbuch:immobilie <http://www.darmstadt.de/Hauptstrasse/22> .
_:id1 grundbuch:kaufdatum "01.04.1976" .
```

**Quelltextbeispiel 2.3:** RDF-Definition mit leerem Knoten

Leere Knoten stellen meist Konstrukte dar, die man mit Formeln wie *es gibt mindestens eine Ressource, für die etwas gilt* umschreiben würde – im obigen Beispiel *es gibt ein Besitzverhältnis, das am 01.04.1976 zwischen Karl-Heinz Müller und dem Haus Hauptstraße 22 in Darmstadt geschlossen wurde* (McBride, 2004, S. 54).

Mit diesen Sprachmitteln lassen sich in RDF beliebige Informationen kodieren. Diese können von Maschinen ausgewertet, aber auch von Menschen gelesen und verstanden werden.

### 2.3.2. RDF-S

Ein Mensch, der ein RDF-Dokument liest, konstruiert meist implizit weitere Bedeutungszusammenhänge, die letztlich zu einem Verstehen der kodierten Information führen. Am leichtesten wird dies klar, wenn man (für Menschen) unsinnige Informationen betrachtet, wie in Abbildung 2.6 dargestellt: es ist einem menschlichen Betrachter klar, dass ein Haus keine Person besitzen kann, sondern nur eine Person ein Haus. Einer Maschine ist dies dagegen nicht klar, weil ihr die Hintergrundinformation der Konzepte `Person`, `besitzt` und `Haus` fehlen. Und auch wir können letztlich nur aufgrund unserer Erfahrung, unseres Weltwissens sagen, dass sich hinter


**Abbildung 2.6.:** Ein RDF-Graphen, der Menschen unsinnig erscheint

`Karl-Heinz Müller` wahrscheinlich eine Person verbirgt und hinter `Darmstadt, Hauptstraße 22` ein Haus – und nicht etwa umgekehrt, weil diese Benennungen typischen Konventionen für Personen und Häuser entsprechen. Maschinen dagegen fehlt dieses Wissen.

Um solches Wissen für Maschinen nutzbar zu machen, werden Ontologien eingesetzt. Sie liefern das Hintergrundwissen, das Vokabular für Informationen, also z.B. für RDF-Graphen (Hitzler u. a., 2008, S. 67). Eine einfache Ontologiesprache ist RDF-Schema, kurz RDF-S (W3C, 2004d). Eine Ontologie in RDF-S wird selbst in RDF kodiert und beschreibt mit Hilfe einiger vordefinierter Grundkonzepte die Elemente, die in einem RDF-Dokument verwendet werden können.

Die beiden wichtigsten dieser Grundkonzepte sind Klassen und Propertys. Klassen fassen ähnliche Konzepte zusammen; im Wesentlichen gehören Subjekte und Objekte von RDF-Tripeln zu einer Klasse, sie werden dann Instanzen dieser Klasse genannt. Propertys definieren Beziehungen zwischen verschiedenen Instanzen, sie bilden die Prädikate von RDF-Tripeln.

Klassen können Hierarchien bilden. So lässt sich z.B. definieren, dass `Haus` eine Unterklasse von `Immobilie` ist. Um anzuzeigen, dass eine Klasse Unterklasse einer anderen Klasse ist, wird das vordefinierte Prädikat `rdfs:subclassOf` verwendet; um zu definieren, dass eine Instanz zu einer Klasse gehört, verwendet man das Prädikat `rdf:type`.

Für Propertys lassen sich die beiden Eigenschaften `rdfs:domain` und `rdfs:range` festlegen; sie bestimmen Definitions- und Wertebereich eines Propertys. So ist es beispielsweise sinnvoll, für das Property `grundbuch:besitzt` festzulegen, dass der Definitionsbereich vom Typ `Person`, der Wertebereich vom Typ `Immobilie` ist. Mit dieser Definition werden unsinnige Aussagen, wie die in Abbildung 2.6 dargestellte, ausgeschlossen.

Außerdem können Propertys, wie auch Klassen, Hierarchien bilden: so kann man z.B. für das Property `grundbuch:besitzt` die Spezialisierungen `grundbuch:hat_erworben` und `grundbuch:hat_geerbt` definieren. Abbildung 2.7 zeigt eine Erweiterung des obigen Grundbuch-Beispiels, das diese Zusammenhänge in Form von RDF-S modelliert.

An diesem Beispiel sieht man auch die Unterscheidung in T-Box und A-Box (siehe Kapitel 2.2.2): Die Beschreibung der Zusammenhänge zwischen den Klassen ist die T-Box, die Beschreibung der konkreten Instanzen ist die A-Box (vgl. Hitzler u. a., 2008, S. 83f). Die Verbindungen zwischen T-Box und A-Box, im Bild durch gestrichelte Pfeile gekennzeichnet, werden durch Instanziierungsbeziehungen mittels `rdf:type` festgelegt. Bisweilen werden diese Bereiche auch als *Daten* und *Metadaten* bezeichnet: die Daten stehen in der A-Box, die Metadaten in der T-Box. In einem RDF-S-Dokument können beide Bereiche gemeinsam notiert werden (McBride, 2004, S. 57).

Ein weiteres Sprachmittel von RDF-S ist die *Reifikation*. Sie erlaubt, Aussagen über Aussagen zu machen, etwa der Form: *Im Darmstädter Grundbuch ist eingetragen, dass Karl-Heinz Müller das Haus Nr. 22 in der Haupstraße besitzt.* Bei einer solchen Aussage wird ein RDF-Statement Subjekt oder Objekt eines weiteren Statements. Quelltextbeispiel 2.4 zeigt diese

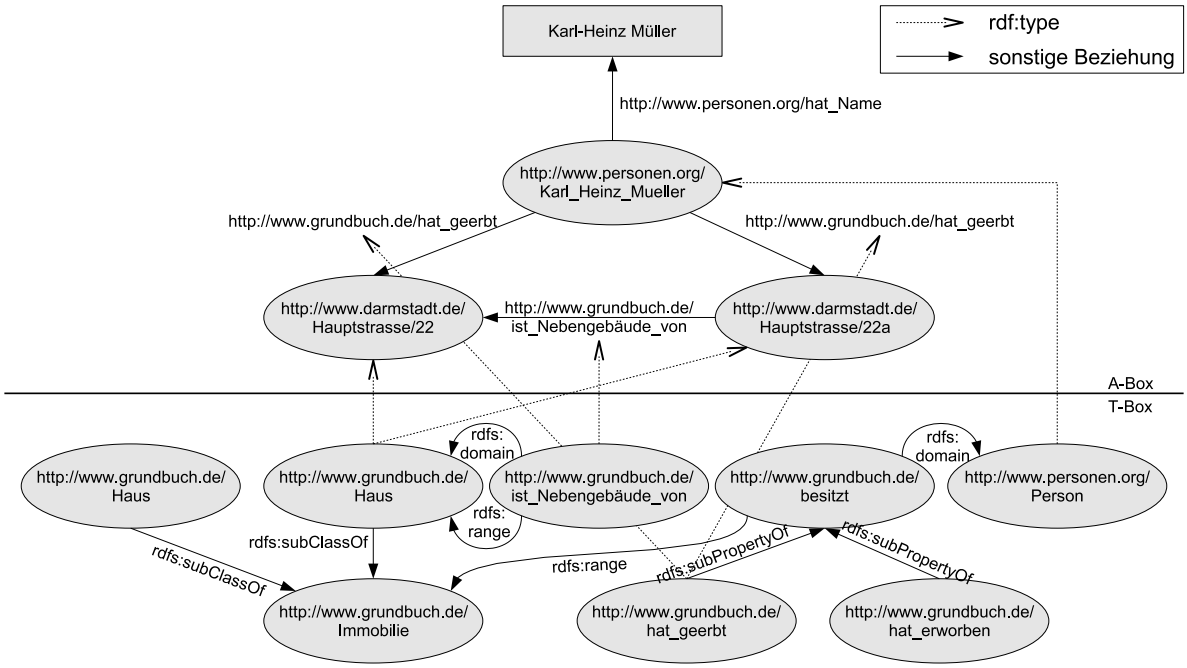



**Abbildung 2.7.:** RDF- und RDFS-Konstrukte in einer Ontologie

Aussage in RDF-S. Sinnvolle Anwendungen der Reifikation sind z.B., den Autor und das Datum einer Aussage festzuhalten, um so zu errechnen, welches Vertrauen man in eine Folgerung aus verschiedenen Aussagen hat (McBride, 2004, S. 62).

```
grundbuch:eintrag rdf:type rdf:Statement
darmstadt:grundbuch/eintrag_nr_521 rdf:type grundbuch:eintrag ;
                                   rdf:subject personen:Karl_Heinz_Mueller ;
                                   rdf:predicate grundbuch:hat_geerbt ;
                                   rdf:object darmstadt:Hauptstrasse/22 .
darmstadt:grundbuch grundbuch:hat_Eintrag darmstadt:grundbuch/eintrag_nr_521;
```

**Quelltextbeispiel 2.4:** Beispiel für Reifikation in N3

Der Beschreibung der Zusammenhänge in der T-Box sind in RDF-S jedoch enge Grenzen gesetzt. So können für Beziehungen z.B. keine Kardinalitäten festgelegt werden (z.B., dass ein Gebäude nur das Nebengebäude *eines* anderen Gebäudes sein kann), es ist nicht möglich, negierte Aussagen zu treffen (z.B., dass ein Gebäude keine Person sein kann) oder mengentheoretische Aussagen zu machen (McBride, 2004, S. 52 u. 63). Das Fehlen der Negation verhindert die Formulierung komplexerer Zusammenhänge, da in der Interpretation von RDF-S grundsätzlich die *Open World Assumption* gilt (Hitzler u. a., 2008, S. 118 ff.), d.h., die Annahme, dass zu den formulierten Aussagen stets potentiell noch weitere hinzutreten könnten. Daher lässt sich eine Negation auch nicht durch Nicht-Formulierung der positiven Aussage darstellen.

Diese Limitierungen werden in der mächtigeren Ontologiesprache OWL, die auf RDF-S aufbaut, beseitigt.

```
<?xml version="1.0"?>
<rdf:RDF xml:base="http://www.personen.org/" 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#">
  <owl:Class rdf:about="#Mensch">
    <owl:unionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="#Mann"/>
      <rdf:Description rdf:about="#Frau"/>
    </owl:unionOf>
  </owl:Class>
  <owl:Class rdf:about="#Mann">
    <owl:disjointWith>
      <rdf:Description rdf:about="#Frau"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#Frau">
  </owl:Class>
</rdf:RDF>
```

**Quelltextbeispiel 2.5:** Beispiel für Klassendefinitionen in OWL

### 2.3.3. OWL Lite, OWL DL und OWL Full

OWL ist kein echtes Akronym, da es für *Web Ontology Language* steht, der Arbeitsgruppe des W3C das Akronym *WOL* jedoch nicht gefiel (W3C, 2004a). Eine verbreitete Erklärung für das Akronym ist ein Bezug zu A. A. Milnes Kinderbuch „Winnie the Pooh“, in dem eine Eule (englisch *Owl*) ihren Namen schreiben kann, dabei jedoch stets den Fehler macht, *Wol* zu schreiben (vgl. Milne, 2000, S. 43).

In der Ontologiesprache OWL stehen zusätzlich zu den Sprachmitteln von RDF-S weitere Möglichkeiten bereit, um Zusammenhänge zwischen Konzepten zu beschreiben. Lassen sich Zusammenhänge zwischen Klassen in RDF-S mit der `subclassOf`-Beziehung nur sehr eingeschränkt beschreiben, so bietet OWL mit Beziehungen wie `intersectionOf`, `unionOf`, `disjointWith` und `complementOf` auch die Möglichkeit, Schnitt- und Vereinigungsmengen sowie disjunkte und komplementäre Mengen zu beschreiben (Hitzler u. a., 2008, S. 132 ff.). Diese Konstrukte führen implizit die Negation ein: ein Element, das *nicht* in einer Menge enthalten ist, ist in deren Komplement enthalten.

Zur Formulierung von OWL-Ontologien wird meist das Format OWL-RDF verwendet, das OWL-Sprachmittel in RDF darstellt. OWL-RDF kann folglich in XML oder Tripel-Notation gespeichert werden. Beispiel 2.5 zeigt eine einfache OWL-Ontologie: Es werden drei Klassen `Mensch`, `Mann` und `Frau` definiert. Es wird festgelegt, dass jeder Mensch entweder ein Mann oder eine Frau ist. Dazu sind zwei Anweisungen notwendig: zum einen wird die Klasse `Mensch` als Vereinigungsmenge von Männern und Frauen definiert, zum anderen wird festgelegt, dass die Klassen `Mann` und `Frau` disjunkt sein sollen.

Ein weiteres wichtiges Sprachmittel von OWL sind *Restriktionen*. Mit Hilfe von Restriktionen lassen sich zum einen minimale und maximale Kardinalitäten für Prädikate festlegen, zum anderen Klassen dadurch definieren, dass Prädikate einen bestimmten Wert haben. Beispiel 2.6 demonstriert diese Möglichkeiten: zu der Klasse `Mensch` werden zwei Prädikate `hatGeschlecht` und `istVolljaehrig` definiert, die jeweils die Kardinalität `1` zugewiesen bekommen (damit für

jeden Menschen genau ein Geschlecht und genau ein Volljährigkeitsprädikat definiert werden muss). Mit Hilfe dieser Prädikate kann man nun über Restriktionen die Klasse `Erwachsener` als Klasse aller volljährigen Menschen und die Klasse `Mann` als Klasse aller männlicher Erwachsener definieren.

Um solche Definitionen mittels OWL treffen zu können, werden in der Regel anonyme Klassen verwendet, die in RDF als anonyme Knoten (siehe Kapitel 2.3.1) realisiert werden. Die Festlegung, dass ein Mensch genau ein Geschlecht haben muss, wird in OWL z.B. so getroffen, dass zunächst eine anonyme Klasse definiert wird, die alle Instanzen mit genau einem Geschlecht enthält, und die Klasse `Mensch` dann als Subklasse dieser Klasse festgelegt wird. Da die anonyme Klasse keinen Namen hat, sondern nur als Restriktion dient, wird dafür das Schlüsselwort `Restriction` anstelle von `Class` verwendet.

Dieses Beispiel zeigt auch eine Einschränkung von OWL: es wäre eine natürliche Vorgehensweise, anstelle des Prädikats `istVolljaehrig` ein Prädikat `hatAlter` einzuführen und dann Erwachsene als Personen ab 18 Jahren zu definieren. OWL erlaubt jedoch nicht die Verwendung von arithmetischen (Vergleichs-)Operatoren, weshalb diese Definition in OWL nicht möglich ist (Horrocks u. a., 2003, S. 25).

Neben Sprachmitteln zur detaillierteren Beschreibung von Klassen hält OWL auch Möglichkeiten bereit, Propertys näher zu charakterisieren (Hitzler u. a., 2008, S. 147ff.). So lassen sich Prädikate als symmetrisch (z.B. `istVerheiratetMit`) und transitiv (z.B. `istVorfahrVon`) kennzeichnen oder ein Property als Inverses eines anderen definieren (z.B. `istElternteilVon` und `istKindVon`).

Darüber hinaus bietet OWL verschiedene Sprachmittel, um die Versionierung von Ontologien zu unterstützen: Es können Verweise auf frühere Versionen und Kompatibilität zu diesen hinzugefügt werden, außerdem können Klassen und Propertys als veraltet gekennzeichnet werden (Antoniou und van Harmelen, 2004, S. 81 f.).

Die Sprache OWL existiert in drei verschiedenen Dialekten: OWL Lite, OWL DL (DL steht für *Description Logics*) und OWL Full. OWL Lite ist eine Untermenge von OWL DL, und OWL DL ist eine Untermenge von OWL Full. Das bedeutet, dass jede gültige Ontologie in OWL Lite auch eine gültige Ontologie in OWL DL und jede gültige Ontologie in OWL DL auch eine gültige Ontologie in OWL Full ist (Antoniou und van Harmelen, 2004, S. 71). Außerdem gibt es eine Menge vorgeschlagener Erweiterungen von OWL DL und OWL Full, die als OWL 1.1 geführt werden, jedoch noch nicht den Status eines Standards haben (W3C, 2006b).

Die drei Dialekte haben eine unterschiedliche Ausdrucksmächtigkeit, denn nicht alles, was in OWL Full ausgedrückt werden kann, ist auch in OWL DL oder OWL Lite möglich. Umgekehrt ist die Entscheidung von Abfragen auf diesen Ontologien umso komplexer, je mächtiger die Sprache ist. Man kann beweisen, dass es nicht möglich ist, in OWL Full alle Abfragen auf Ontologien zu entscheiden (Antoniou u. a., 2005, S. 19). Dagegen sind Abfragen auf Ontologien in OWL DL immer entscheidbar. OWL Lite ist eine Untermenge von OWL DL, für die das Problem der Entscheidung von Abfragen eine niedrigere Komplexität hat (W3C, 2004c).

In OWL Lite sind insbesondere diejenigen Sprachkonstrukte nur eingeschränkt verwendbar, die implizit eine Negation einführen, also `unionOf`, `complementOf` und `disjointWith`. Außerdem dürfen Kardinalitätsrestriktionen nur mit den Zahlen 0 und 1 definiert werden. Andere Konstrukte dürfen nur mit bestimmten Subjekt- oder Objekttypen verwendet werden (Hitzler

```
<?xml version="1.0"?>
<rdf:RDF xml:base="http://www.personen.org/" 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#">
  <owl:Class rdf:about="#Mensch">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hatGeschlecht"/>
        <owl:cardinality>1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    ...
  </owl:Class>

  <owl:DatatypeProperty rdf:about="#hatGeschlecht">
    <rdfs:domain>
      <rdf:Description rdf:about="#Mensch"/>
    </rdfs:domain>
    <rdfs:range>
      ...
    </rdfs:range>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#istVolljaehrig">
    ...
  </owl:DatatypeProperty>

  <owl:Class rdf:about="#Erwachsener">
    <rdfs:subClassOf>
      <rdf:Description rdf:about="#Mensch"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue>true</owl:hasValue>
        <owl:onProperty>
          <rdf:Description rdf:about="#istVolljaehrig"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#Mann">
    <rdfs:subClassOf>
      <rdf:Description rdf:about="#Erwachsener"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue>männlich</owl:hasValue>
        <owl:onProperty>
          <rdf:Description rdf:about="#hatGeschlecht"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
</rdf:RDF>
```

**Quelltextbeispiel 2.6:** Beispiel für Restriktionen in OWL

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:base="http://www.personen.org" 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#">
  <owl:Class rdf:about="#Beruf">
  </owl:Class>
  <owl:Class rdf:about="#Maurer">
    <rdf:type>
      <rdf:Description rdf:about="#Beruf"/>
    </rdf:type>
  </owl:Class>
  <owl:Description rdf:about="#Tom">
    <rdf:type>
      <rdf:Description rdf:about="#Maurer"/>
    </rdf:type>
  </owl:Description>
</rdf:RDF>
```

**Quelltextbeispiel 2.7:** Beispiel für Metamodellierung in OWL

u. a., 2008, S. 153f.). Die Ausdrucksmächtigkeit von OWL Lite entspricht der Beschreibungslogik $\mathcal{SHIF}(D)$ (Horrocks u. a., 2003, S. 23, siehe auch Kapitel 2.2.3).

Die wichtigsten Einschränkungen von OWL DL (die gleichzeitig auch für OWL Lite gelten) sind, dass jede Ressource, die verwendet wird, einen Typ haben muss, wobei bestimmte Typen unverträglich sind (so können z.B. Instanzen nicht gleichzeitig Klassen sein). Außerdem dürfen nicht alle Konstrukte aus RDF-S in OWL DL verwendet werden (Hitzler u. a., 2008, S. 153). Die Ausdrucksmächtigkeit von OWL DL entspricht der Beschreibungslogik $\mathcal{SHOIN}(D)$ (Horrocks u. a., 2003, S. 23 siehe auch Kapitel 2.2.3).

In OWL Full dagegen sind alle Konstrukte aus RDF-S erlaubt; jedes gültige RDF- und jedes gültige RDF-S-Dokument ist daher auch ein gültiges OWL-Full-Dokument (aber nicht notwendigerweise ein gültiges OWL-DL-Dokument). Außerdem können Klassen gleichzeitig Instanzen sein, was es erlaubt, auch Metamodelle in OWL Full zu definieren. Beispiel 2.7 zeigt, wie Metaklassen in OWL Full definiert werden: Die Klasse `Maurer` ist zugleich Instanz der Klasse `Beruf`, die damit zu einer Metaklasse wird. Die Instanz `Tom` gehört dann zur Klasse `Maurer`, aber nicht zur Metaklasse `Beruf`, denn die Aussage „Tom ist ein Beruf“ ist im Gegensatz zur Aussage „Tom ist Maurer“ unsinnig (vgl. Sowa, 2000, S. 29). In OWL Full lassen sich auch Aussagen treffen, die nicht in Prädikatenlogik formulierbar sind, daher ist OWL Full zu keiner Beschreibungslogik äquivalent (Horrocks u. a., 2003, S. 23 f.).

Zu den vorgeschlagenen Erweiterungen für OWL 1.1 zählen, neben einigen syntaktischen Vereinfachungen, etwa die Einführung von reflexiven und disjunkten Eigenschaften und die Möglichkeit, benutzerdefinierte Datentypen zu verwenden (W3C, 2006b).

### 2.3.4. Weitere Repräsentationssprachen

Neben den am weitesten verbreiteten Ontologiesprachen RDF-S und OWL gibt es eine Reihe weiterer Sprachen, mit denen Ontologien ebenfalls beschrieben werden können, die jedoch weitaus seltener eingesetzt werden (Cardoso, 2007, S. 85). An dieser Stelle sollen mit

DAML+OIL und F-Logic noch zwei dieser Sprachen kurz vorgestellt werden, die ebenfalls häufig verwendet werden.

*DAML+OIL*, Abkürzung für **DA**RPA **M**arkup **L**anguage + **O**ntology **I**nference **L**ayer (W3C, 2001), ist der historische Vorläufer von OWL[7]. Es entstand aus der in Amerika entwickelten Ontologiesprache *DAML-ONT* und der europäischen Entwicklung *OIL* (Fensel, 2002, S. 216 ff.).

DAML+OIL ist, wie auch OWL, eine auf RDF aufgebaute Sprache (W3C, 2001). Da OWL stark an DAML+OIL angelehnt wurde, gibt es nur wenige Unterschiede zwischen den beiden Sprachen, die über eine unterschiedliche Benennung von Konzepten hinausgehen (W3C, 2004c). DAML+OIL hat, wie auch OWL DL, die Mächtigkeit der Beschreibungslogik $\mathcal{SHOIQ}(D)$ (Baader u. a., 2005, S. 243 f.[8], siehe auch Kapitel 2.2.3).

*F-Logic* (für **F**rame-Logic) wurde ursprünglich für die logische Programmierung entwickelt, kann aber auch zur Definition von Ontologien eingesetzt werden. Frames sind kleinste Bausteine von Ontologien wie Klassendefinitionen mit einer Vererbungshierarchie, Attribute von Klassen (auch „Konzepte“ oder „Slots“ genannt), die mit Restriktionen versehen werden, und Instanzdefinitionen. F-Logic erlaubt die Definition von Klassen und deren Eigenschaften sowie von zusätzlichen Regeln, die für Instanzen dieser Klassen gelten. F-Logic hat eine höhere Mächtigkeit als Beschreibungslogiken, ist dafür aber nicht immer entscheidbar (Kifer, 2005, S. 30).

Beispiel 2.8 zeigt einen Ausschnitt aus einer F-Logic-Ontologie, in der ähnliche Sachverhalte modelliert sind wie in Beispiel 2.6. Im Gegensatz zu OWL besteht hier die Möglichkeit, auch arithmetische Vergleichsoperatoren zu nutzen und so einen Erwachsenen als eine Person ab 18 Jahren zu definieren.

*SWRL* (für **S**emantic **W**eb **R**ule **L**anguage) ist eine Erweiterung von OWL, mit der sich auch logische Schlussregeln abbilden lassen. Auch arithmetische Funktionen und Vergleiche sowie Datumsarithmetik sind in SWRL möglich, so dass sich auch Definitionen wie „Ein Erwachsener ist ein Mensch, der mindestens 18 Jahre alt ist“, ähnlich wie in F-Logic, formulieren lassen (Horrocks u. a., 2004).

## 2.4. Einsatzmöglichkeiten von Ontologien

Das „klassische“ Einsatzgebiet von Ontologien ist das Semantic Web. Durch die *Annotation* von Dokumenten mit Metadaten kann eine gegenüber der stichwortbasierten Suche verbesserte Suchmöglichkeit für das Internet realisiert werden: irrelevante Dokumente können aus der Er-

[7] Auch die Ontologiesprache *SHOE* (**S**imple **H**TML **O**ntology **E**xtensions) hatte Einfluss auf die Entwicklung von OWL, dieser ist jedoch nicht so stark erkennbar wie der von DAML+OIL (Horrocks u. a., 2003, S. 9). SHOE wird heute kaum noch verwendet (Cardoso, 2007, S. 85) und daher an dieser Stelle nicht weiter behandelt.

[8] Baader u.a. schreiben genauer, dass DAML+OIL in $\mathcal{SHIQ}$ überführt werden kann, wenn man Datentypen und Klassen als Aufzählungen von Individuen nicht berücksichtigt. Daraus folgt unmittelbar, dass DAML+OIL die Mächtigkeit von $\mathcal{SHOIQ}(D)$ hat, da dies die Erweiterung von $\mathcal{SHIQ}$ um diese Merkmale ist.

```
:- prefix ="http://www.personen.org#".
:- prefix xsd="http://www.w3.org/2001/XMLSchema#".
:- module =#personen.

#Mann::#Mensch.
#Frau::#Mensch.
#weiblich::#Geschlecht.
#maennlich::#Geschlecht.
#Erwachsener::#Mensch.
#Mensch[].
#Geschlecht[].

#Mensch[#hatGeschlecht=>#Geschlecht].
#Mensch[#alter=>xsd#integer].

RULE #defmann: FORALL X,A
  X:#Mann <- X:#Mensch AND X[#hatGeschlecht->>#maennlich]
                       AND X[#alter->>A] AND greaterorequal(A, 18).
```

**Quelltextbeispiel 2.8:** Beispielontologie in F-Logic

gebnismenge entfernt, zusätzliche, nicht von einem einzelnen Stichwort erfasste, aber dennoch relevante Dokumente hinzugefügt werden (Davies u. a., 2006a, S. 3). Dabei gibt es verschiedene Möglichkeiten, die Metadaten mit den Dokumenten zu verknüpfen sowie unterschiedliche Werkzeuge, die die semantische Annotation von Webseiten teilautomatisiert unterstützen; einen Überblick darüber gibt Reif (2006, S. 405 ff.).

Während dieses semantische Web sich noch nicht sehr weit durchgesetzt hat – noch immer werden die meisten Internetsuchen mit stichwortbasierten Suchmaschinen wie *Google* durchgeführt, und nur wenige Internetquellen sind mit Ontologien annotiert – haben sich Ontologien in einigen anderen Gebieten etabliert und dort die Entwicklung nützlicher Anwendungen ermöglicht. Ontologien können zur Kommunikation, zur Wissensrepräsentation und für das logische Schlussfolgern eingesetzt werden (Gruninger und Lee, 2002, S. 40). In diesem Kapitel werden exemplarisch einige Einsatzgebiete vorgestellt.

### 2.4.1. Datenintegration

Wenn auch die Integration der Information im World Wide Web in einem weltumspannenden semantischen Netz auf absehbare Zeit nicht erreicht werden wird, so gibt es dennoch Lösungen, die diese Integration lokal – etwa bezogen auf die Informationen eines Unternehmens – ermöglichen. Dazu werden verschiedenartige Informationsquellen – Datenbanken, Textdokumente usw. – über eine gemeinsame Schnittstelle verfügbar gemacht.

Für die Integration wird zunächst eine Ontologie manuell entwickelt, welche die für ein Unternehmen wesentlichen Begriffe und deren Relationen enthält. Alternativ kann eine erste Version automatisiert aus bereits vorhandenen strukturierten Datenquellen, wie etwa Datenbankschemas oder der Verweisstruktur des Firmenintranets, erstellt werden (Abecker und van Elst, 2004, S. 439).

Im nächsten Schritt werden die Datenquellen erschlossen. Dabei werden mit Hilfe von Data- und Text-Mining-Verfahren Begriffe aus den Datenquellen extrahiert und mit den Konzep-

ten der Ontologie verknüpft. Experten bearbeiten diese Verknüpfungen und erweitern so die zugrunde liegende Ontologie (John und Drescher, 2006, S. 249). Strukturierte Datenquellen wie Datenbanken oder XML-Dokumente werden mit Hilfe von sogenannten *Wrappern*, die diese Information anhand der gegebenen Verknüpfungen in ein Semantic-Web-Format wie RDF übersetzen, an das System angebunden (Oberle und Spyns, 2004, S. 502 f.).

Indem die Konzepte der Ontologie für den Nutzer geeignet dargestellt werden, wird eine flexible, navigierbare Sicht auf die Informationen, die in den unterschiedlichen Datenquellen gespeichert ist, erzeugt (Fensel, 2004, S. 84 ff.). Dabei können z.B. graphische Übersichten über das Unternehmenswissen, sogenannte *Wissenslandkarten*, oder intelligente Suchmaschinen zum Einsatz kommen (Abecker und van Elst, 2004, S. 444 ff.). Auch sogenannte *Push-Dienste*, die Nutzer automatisch benachrichtigen, wenn Informationen zu einem bestimmten Thema neu bereitgestellt wurden, werden so ermöglicht (Blumauer und Fundneider, 2006, S. 231 ff.). Abbildung 2.8 zeigt diese Prozesse.

Die in den Ontologien modellierte Information über den Zusammenhang von Dokumenten kann genutzt werden, um dem Nutzer zusätzliche Hilfe, wie das Vorschlagen von verwandten Themenfeldern oder die Nennung von Themenexperten im Unternehmen, anzubieten. Indem Begriffe in Dokumenten markiert und mit Verweisen auf Ontologien versehen werden, können auch Beziehungen zwischen Dokumenten hergestellt und Informationen aus verschiedenen Datenquellen verknüpft werden (Nagarajan, 2006, S. 36 ff.). Mit Hilfe von Ontologien können auch Beziehungen zwischen Daten automatisiert gefunden werden, die nicht offensichtlich, etwa durch gemeinsame Schlagworte, erkennbar sind (Cardoso und Sheth, 2006a, S. 23).

Auf diese Weise werden heterogene Datenquellen über eine gemeinsame Schnittstelle nutzbar, was die Lokalisierung von Information stark vereinfacht (Blumauer und Fundneider, 2006, S. 232 ff.). Abecker und van Elst (2004) geben einen Überblick über solche Integrationslösungen.

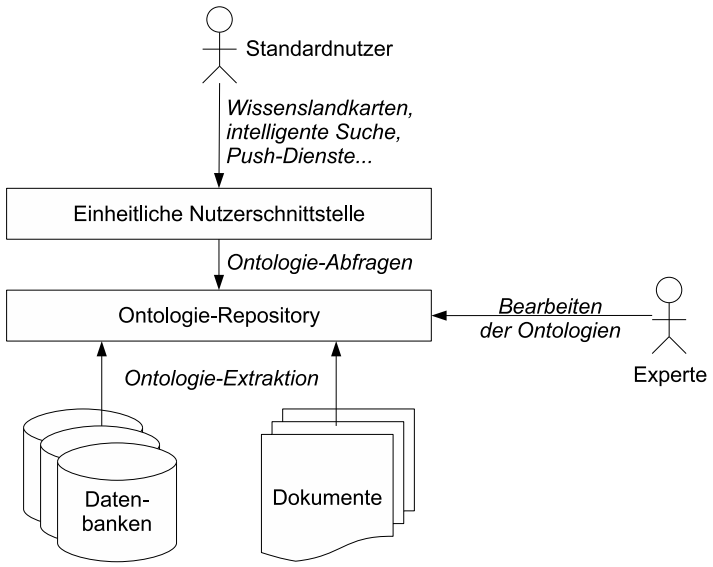



**Abbildung 2.8.:** Ontologiebasierte Informationsintegration (in Anlehnung an Fensel, 2004, S. 85)

### 2.4.2. Web Services und Agenten

Web Services sind ein Kommunikationsstandard, mit dem Daten zwischen unterschiedlichen, an verschiedenen Orten installierten Anwendungen ausgetauscht werden können. Hierbei werden üblicherweise die Konzepte UDDI (**U**niversal **D**escription, **D**iscovery and **I**ntegration) und WSDL (**W**eb **S**ervice **D**escription **L**anguage) eingesetzt. Diese erfassen jedoch lediglich die technischen Daten eines Web Services, nicht aber eine inhaltliche Beschreibung dessen, was genau ein Dienst eigentlich tut (Dostal und Jeckle, 2004, S. 59 f.). Das Auffinden eines Dienstes, der eine bestimmte Aufgabe erfüllt, ist daher schwierig und vor allem kaum zu automatisieren (Cardoso und Sheth, 2006a, S. 18). Diese Lücke kann mit semantischer Beschreibung auf der Basis von Ontologien geschlossen werden. Zu diesem Zweck müssen die Web-Service-Beschreibungen semantisch annotiert werden, indem Funktionen und Parameter mit Verweisen auf Ontologien, die die zu Grunde liegenden Konzepte beschreiben, versehen werden (Nagarajan, 2006, S. 35 ff.).

Hierzu wurden beim W3C verschiedene Methoden vorgeschlagen, die bekanntesten sind OWL-S (W3C, 2004b), die Web Service Modeling Language WSMO (W3C, 2005b) und WSDL-S (W3C, 2005c), von denen jedoch bislang keiner den Status eines Standards hat. Polleres u. a. (2006, S. 513 ff.) geben einen detaillierten Überblick über die einzelnen Merkmale dieser Ansätze.

Eine mögliche Anwendung von Web Services sind Agenten. Agenten sind Computerprogramme, die bestimmte Ziele verfolgen und sich dabei dynamisch an ihre Umgebung anpassen können, um diese Ziele zu erreichen (Russell und Norvig, 2003, S. 4). Dabei interagieren sie häufig mit anderen Agenten, die zusätzliche Informationen bereitstellen oder die Erfüllung bestimmter Teilaufgaben übernehmen (Gruber, 1995, S. 907).

Komplexere Agenten bauen oftmals eine interne Weltrepräsentation auf (Russell und Norvig, 2003, S. 48 ff.). Für diese Repräsentation kann eine Ontologie zum Einsatz kommen. Mit Hilfe geeigneter Schlussfolgerungsmechanismen kann diese Ontologie dazu genutzt werden, Wege zu finden, um die Ziele des Agenten zu erreichen (Sycara und Paolucci, 2004, S. 344).

Um miteinander – etwa durch die Nutzung von Web Services – interagieren zu können, müssen sich Agenten zunächst gegenseitig ihre Fähigkeiten, also ihre Schnittstellen beschreiben. Dies geschieht mit Hilfe einer gemeinsamen Ontologie, die die Begriffe einer Domäne definiert (Hendler, 2001, S. 32). Fähigkeiten werden in der Regel als Funktionen dargestellt, die bestimmte Parameter erwarten und verschiedene Änderungen der gemeinsamen Umgebung der Agenten durchführen, also Vor- und Nachbedingungen erfüllen (Sycara und Paolucci, 2004, S. 346 ff.).

Damit die Schnittstellenbeschreibungen eindeutig definiert sind, werden hierin die Begriffe der gemeinsamen Ontologie verwendet. Spricht nun ein Agent die Schnittstelle eines anderen Agenten an, verwendet er hierzu ebenfalls Begriffe der gemeinsamen Ontologie. Damit sind auch die Nachrichten vom Empfänger eindeutig dekodierbar (Sycara und Paolucci, 2004, S. 354).

Während die Schnittstellenbeschreibungen nur einzelne Funktionen und Nachrichten definieren, können komplexere Interaktionen zwischen Agenten, die einem bestimmten Protokoll folgen, wie etwa Auktionen, auf der pragmatischen Ebene ebenfalls in einer Ontologie definiert werden. Hierin wird festgelegt, welche Nachrichten ein Agent in einer bestimmten Situation

verwenden kann (Sycara und Paolucci, 2004, S. 356 f., und Burstein und McDermott, 2005, S. 1). Abbildung 2.9 zeigt das Zusammenspiel zweier Agenten und der verwendeten Ontologien.

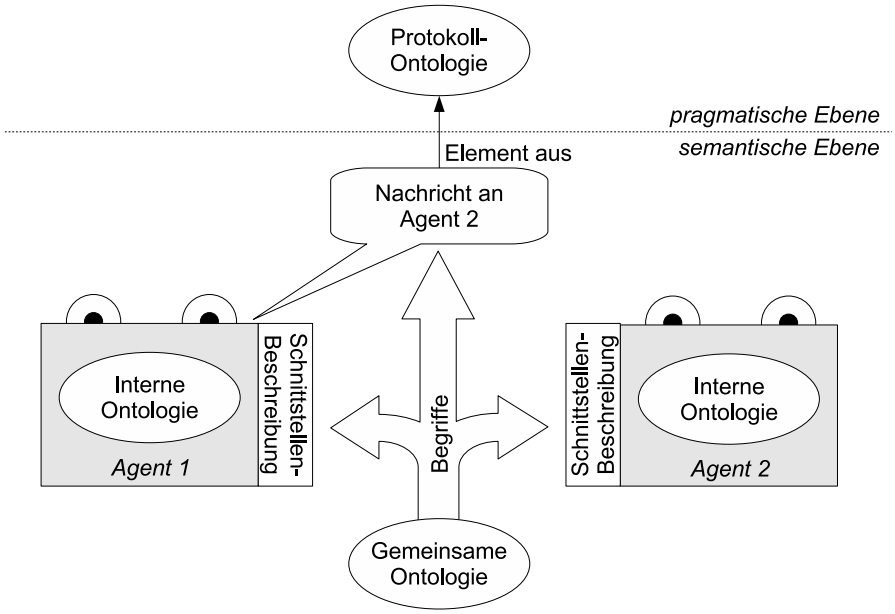



**Abbildung 2.9.:** Ontologiebasierte Agenten

### 2.4.3. E-Business

Geschäfte im E-Business laufen üblicherweise in drei Phasen ab: zunächst werden potentielle Geschäftspartner gesucht, dann werden mit diesen Verhandlungen geführt, und am Ende entsteht ein Vertrag, dessen Verpflichtungen von beiden Seiten erfüllt werden. In jeder dieser drei Phasen können Ontologietechniken eingesetzt werden (Schoop und Jertila, 2004, S. 138).

Bei der Suche nach einem Geschäftspartner, der ein Produkt oder eine Dienstleistung mit bestimmten Merkmalen anbietet, führt der gängige Weg derzeit meist über die zeitaufwändige manuelle Volltextsuche (Fensel, 2004, S. 96 f.). Indem man beispielsweise auf einer ontologiebasierten E-Commerce-Plattform diese Informationen in Ontologien codiert, können detaillierte Suchanfragen zielgerichteter verarbeitet werden. Werden zudem einzelne Produktmerkmale gewichtet, können die besten Produkte für die eigenen Bedürfnisse vollautomatisch ermittelt werden (Schreder u. a., 2007, S. 3 ff.). Auch automatisierte Agenten können so zur Beschaffung eingesetzt werden, da die manuelle Interpretation der Produktbeschreibungen entfallen kann (Fensel, 2004, S. 97 ff.).

Während der Verhandlungsphase ist es wichtig, auf ein gemeinsames Vokabular und ein gemeinsames Grundverständnis der verhandelten Sachverhalte zurückgreifen zu können. Je komplexer die verhandelten Güter und Dienstleistungen, desto umfangreicher und komplexer wird auch das eingesetzte Vokabular (Malucelli u. a., 2005, S. 27). Wenn Menschen die Verhandlung führen, kann dieses Vokabular auch in einer weniger formalen Beschreibung vorliegen, z.B. als Glossar. Werden jedoch intelligente Agenten eingesetzt, entfällt die Möglichkeit, Angebote und Gegenangebote durch menschliches Hintergrundwissen zu interpretieren (Ding u. a., 2004, S. 594). In diesem Fall müssen die Agenten auf einer gemeinsamen Wissensbasis aufbauen, in der

die verhandelten Konzepte eindeutig und in maschinenlesbarer Form beschrieben sind; wofür sich Ontologien ebenfalls anbieten (Corcho und Gómez-Pérez, 2001, S. 131 ff.).

Ontologien, die Produkte, Dienstleistungen und Vertragsbedingungen definieren, können auch zum Einsatz kommen, um komplexe Verträge semantisch eindeutig zu codieren. Indem in Verträge Verweise auf Ontologien eingefügt werden, können diese semantisch eindeutig definiert werden (Rebstock u. a., 2008, S. 153 ff.). Insbesondere können die genutzten Ontologien dazu verwendet werden, Uneinigkeiten über Vertragsinhalte zu lösen oder schon im Vorfeld zu vermeiden (Schoop und Jertila, 2004, S.139 f.).

## 2.5. Werkzeuge

Um ontologiebasierte Systeme zu entwickeln, sind unterschiedliche Werkzeuge nötig. Diese reichen von Editoren zum Entwickeln von Ontologien bis hin zu performanten Persistenz- und Inferenzlösungen. Dabei kommt es vor, dass Werkzeuge mehrere Funktionalitäten abdecken und sich daher nicht immer eindeutig einer Klasse zuordnen lassen: Manche Editoren enthalten Reasoning-Funktionalitäten, einige Reasoning-Werkzeuge besitzen Persistenz-Mechanismen, und viele Open-Source-Applikationen lassen sich auch als Frameworks zur Entwicklung eigener Anwendungen nutzen (vgl. Waterfeld u. a., 2008, S. 63 ff.). Daher werden einige Werkzeuge in der folgenden Übersicht mehrfach aufgeführt.

### 2.5.1. Editoren und Visualisierungswerkzeuge

Da Ontologien, wie in Kapitel 2.3 dargestellt, Textdateien sind, lassen sie sich mit jedem einfachen Texteditor bearbeiten. Obwohl dies in der Praxis oft auch geschieht – einfache Texteditoren zählen zu den fünf am weitesten verbreiteten Ontologieeditoren (Cardoso, 2007, S. 85) – gibt es auch spezielle Werkzeuge, die das Bearbeiten von Ontologien, z.B. durch GUI-Unterstützung, Visualisierung und Validierung, vereinfachen.

#### 2.5.1.1. Protégé

*Protégé* ist der mit großem Abstand am häufigsten genutzte Ontologieeditor (Cardoso, 2007, S. 85). Zugleich ist es eines der ältesten Werkzeuge seiner Art – die erste Version wurde bereits 1987 als System zur Entwicklung von wissensbasierten Systemen im medizinischen Bereich konzipiert (Gennari u. a., 2003, S. 90).

Die mittlerweile aktuelle Version des Werkzeugs basiert auf der vierten Generation von Protégé, Protégé-2000. Das interne Repräsentationsmodell von Protégé ist ein frame-basiertes Wissensmodell (vgl. Kapitel 2.3.4). Protégé kann Ontologien in unterschiedlichen Formaten, z.B. RDF, OWL und UML, importieren und in das interne Wissensmodell überführen; Ontologien können auch in relationalen Datenbanken abgelegt werden (Knublauch u. a., 2004, S. 231).

Das Werkzeug ist nach einer Plug-In-Architektur implementiert, die die Entwicklung unterschiedlichster Plug-Ins ermöglicht. Eines der bekanntesten ist das Plugin *Protégé-OWL*, das zur Erstellung, Bearbeitung und Validierung von OWL-Ontologien verwendet werden kann. Es unterstützt OWL Lite, OWL DL und eine Teilmenge von OWL Full (Knublauch u. a., 2004, S.

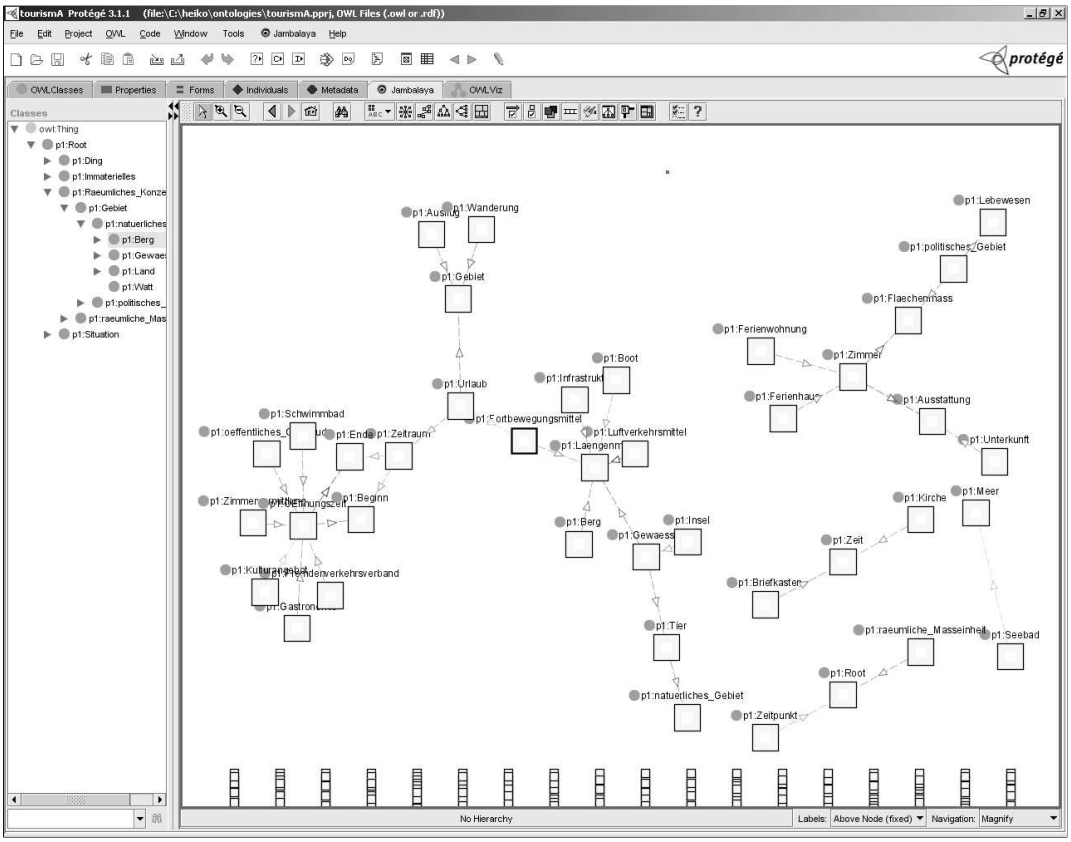



**Abbildung 2.10.:** Screenshot des Editors Protégé mit dem Plug-In Jambalaya

229 ff.). Die Verarbeitung von Ontologien in SWRL ist ebenfalls möglich. Weitere Plug-Ins dienen z.B. zur Versionierung von Ontologien, zum Matching oder zur Visualisierung. Abbildung 2.10 zeigt einen Screenshot des Visualisierungs-Plug-Ins *Jambalaya* (Storey u. a., 2001).

Protégé ist als Open-Source-API in Java verfügbar und kann daher auch als Framework zur Entwicklung ontologiebasierter Applikationen genutzt werden (Cardoso, 2006, S. 356). Auf die Eigenschaften von Protégé als Programmierframework wird in Kapitel 2.5.2.4 eingegangen.

#### 2.5.1.2. SWOOP

*SWOOP* ist der nach Protégé am häufigsten eingesetzte Ontologie-Editor, der ebenfalls als Open-Source-Produkt auf Java-Basis verfügbar ist. Er wurde an der University of Maryland entwickelt, die aktuelle Version ist 2.3.

Bei SWOOP werden Ontologien ähnlich wie Webseiten visualisiert: in einem Browserfenster werden Konzepte angezeigt, die über Hyperlinks mit verwandten Konzepten verbunden sind (siehe Abbildung 2.11). Die Ähnlichkeit zu einem Webbrowser soll eine intuitive Bedienung ermöglichen (Kalyanpur u. a., 2006, S. 1 ff.).

Mit SWOOP können Ontologien in allen drei OWL-Dialekten verarbeitet werden. Diese können in verschiedenen Formaten wie RDF-XML oder N-Triples angezeigt werden. Wie Protégé folgt auch SWOOP einem Plug-In-basierten Architekturmodell. Verfügbare Plug-Ins unterstüt-

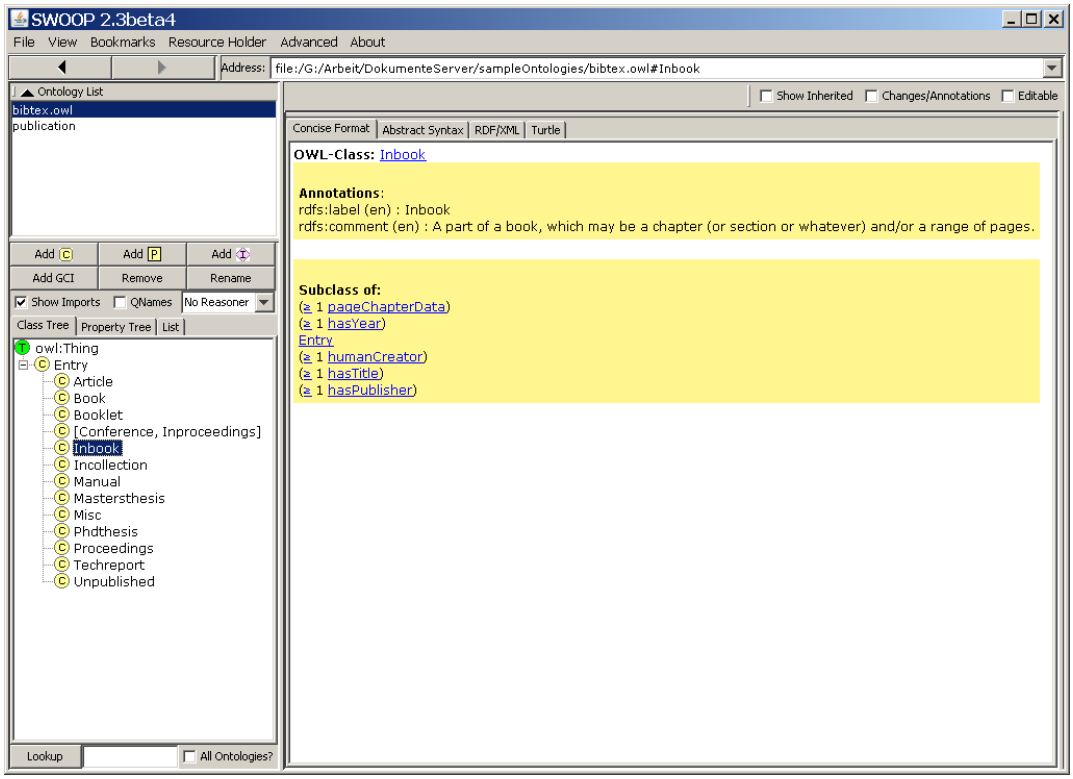



**Abbildung 2.11.:** Screenshot des Editors Swoop

zen z.B. die Konsistenzprüfung und die Fehlersuche sowie die Visualisierung von Ontologien (Kalyanpur u. a., 2006, S. 3 ff.).

#### 2.5.1.3. OntoStudio

Die Anwendung *OntoStudio*, der Nachfolger von *OntoEdit*, wird von der deutschen Firma ontoprise GmbH vertrieben und ist das am weitesten verbreitete kommerzielle Werkzeug zum Bearbeiten von Ontologien (Cardoso, 2007, S. 85). OntoStudio ist eine Applikation auf Eclipse-Basis, die aktuell in der Version 2.0 vorliegt.

OntoStudio wurde zur Erstellung von Ontologien in F-Logic entwickelt (siehe Kapitel 2.3.4), unterstützt aber den Import und Export weiterer Formate, wie RDF/RDFS und OWL in XML- und Tripelnotation und UML. Abbildung 2.12 zeigt einen Screenshot der Anwendung.

Ein großer Schwerpunkt des Werkzeugs liegt auf der Erstellung und Bearbeitung von Regeln und deren Anwendung zum automatisierten Folgern neuer Information. Neben einem graphischen Editor und einem Quelltexteditor für Ontologien und Regeln F-Logic zeichnet sich das Werkzeug vor allem durch eine Vielzahl von Schnittstellen zu Verzeichnisdiensten, Datenbanksystemen und Office-Produkten aus (ontoprise GmbH, 2007, S. 6 f.).

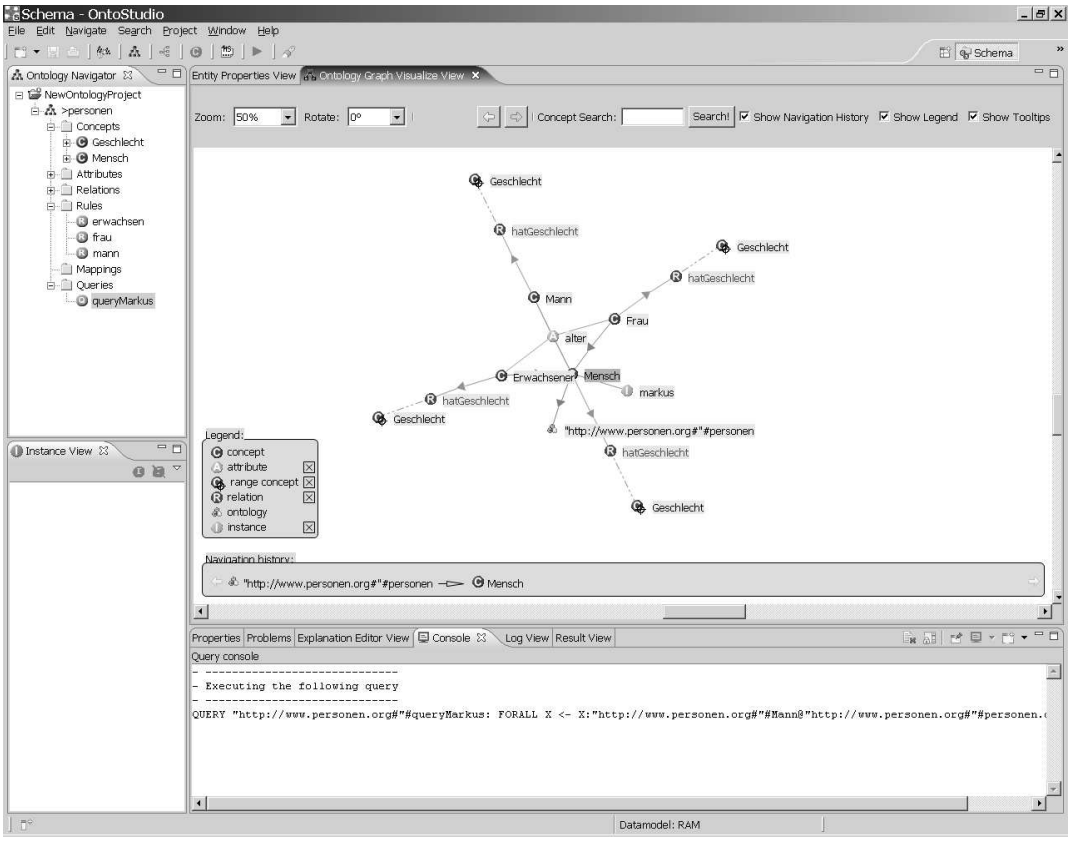



**Abbildung 2.12.:** Screenshot des Werkzeugs OntoStudio

#### 2.5.1.4. Altova SemanticWorks

Der Editor *SemanticWorks* der Firma Altova ist nach OntoStudio das am weitesten verbreitete kommerzielle Werkzeug zur Bearbeitung von Ontologien (Cardoso, 2007, S. 85). Die erste Version wurde im Jahr 2005 veröffentlicht, die aktuelle Version ist SemanticWorks 2008.

Im Gegensatz zu OntoStudio liegt der Schwerpunkt des Werkzeugs weniger auf der logikbasierten Nutzung von Ontologien, sondern stärker auf der Unterstützung des Nutzers beim Modellieren und Editieren von Ontologien. Es bietet eine graphische Schnittstelle, mit der sich Ontologien in RDF, RDF-S und allen Dialekten von OWL erstellen und validieren lassen. Darüber hinaus besteht die Möglichkeit, Ontologien in OWL Lite und OWL DL auf ihre Konsistenz zu prüfen (Altova, 2007). Abbildung 2.13 zeigt einen Screenshot der Anwendung.

Einen detaillierten Vergleich verschiedener Ontologie-Editoren geben Waterfeld u. a. (2008, S. 63 ff.).

### 2.5.2. Programmierframeworks

Um Anwendungen zu entwickeln, die mit Ontologien arbeiten, benötigt man APIs und Frameworks, die entsprechende Funktionalitäten bereitstellen.

Grundfunktionen, die jedes Framework besitzt, sind das Einlesen und Schreiben von Ontologie-Dateien mit Hilfe von Parsern und Serialisierern, sowie Möglichkeiten, die in On-

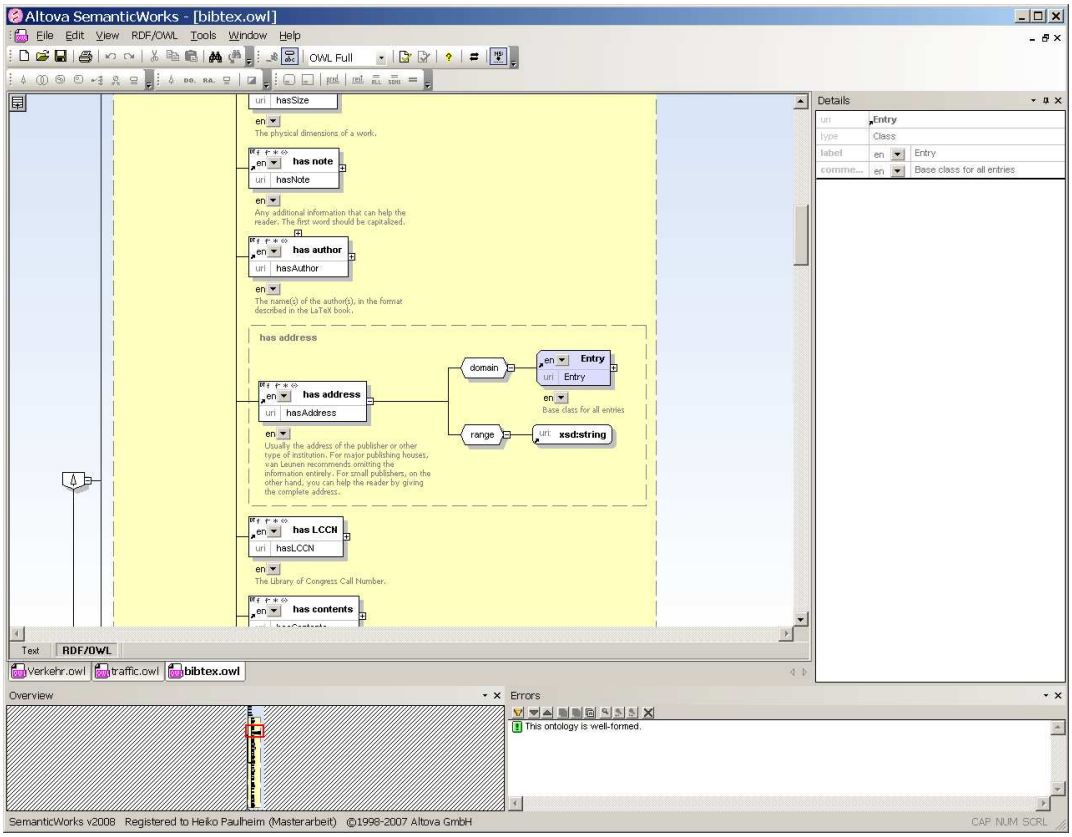



**Abbildung 2.13.:** Screenshot des Editors Altova Semantic Works 2008

tologien enthaltenen Konzepte abzufragen und zu manipulieren. Weitere Funktionen können etwa die Unterstützung von Reasoning (siehe Kapitel 2.5.4) und die Anbindung verschiedener Persistenzsysteme (siehe Kapitel 2.5.3) sein (Bechhofer u. a., 2003b, S. 625 ff.).

Die bekanntesten Semantic-Web-Frameworks sind WonderWeb, JENA, KAON2 und Protégé, die alle auf der Programmiersprache Java basieren.

#### 2.5.2.1. WonderWeb OWL API

Das *WonderWeb OWL API*, oft nur kurz als *OWL API* bezeichnet, ist ein Open-Source-Framework zur Verarbeitung von Ontologien. Es besteht im Kern aus einem umfangreichen Objektmodell, in dem es Java-Interfaces für alle Sprachkonstrukte aus OWL gibt. Die Objekte der Klassen, die diese Interfaces implementieren, sind nicht direkt änderbar, sondern werden mit Hilfe von *Visitor*-Objekten (vgl. Gamma u. a., 1996, S. 301 ff.) manipuliert.

Die Interfaces werden in der Distribution durch Klassen implementiert, die eine Verarbeitung im Hauptspeicher erlauben. Sie können aber auch durch benutzerdefinierte Klassen implementiert werden, um z.B. über dieselbe Schnittstelle Zugriff auf eine persistent gespeicherte und nur in Teilen in den Hauptspeicher geladene Ontologie zu ermöglichen.

Zum Umfang von OWL API gehören des weiteren verschiedene Paare von Parsern und Serialisierern; für ein Reasoning-System besteht eine Schnittstelle, es ist aber keines enthalten (Bechhofer u. a., 2003b, S. 667 ff.). Mit OWL API können Ontologien in allen drei Dialekten

von OWL verarbeitet werden; es existiert auch eine Erweiterung, die die Sprachkonstrukte von OWL 1.1 unterstützt (Horridge u. a., 2007).

#### 2.5.2.2. JENA

*JENA* ist ein Open-Source-Framework zur Arbeit mit RDF-Daten, das seit 2000 von Hewlett Packard entwickelt wird. Im Gegensatz zu OWL API, bei welchem sich die Sprachkonstrukte von OWL direkt in der Architektur des APIs wiederfinden, verfolgt JENA einen sprachunabhängigen Ansatz: die Grundeinheit von JENA sind die Graphen von RDF-Modellen, die verschiedene spezialisierte Ausprägungen haben können. In der ersten Version wurden die Ausprägungen RDF und DAML+OIL unterstützt, seit der Version JENA2, die im Jahr 2003 veröffentlicht wurde, gibt es auch spezielle Routinen zur Verarbeitung von RDFS und allen drei Dialekten von OWL (Carroll u. a., 2004, S. 74 f.).

RDF-Modelle können sowohl im Hauptspeicher als auch persistent verarbeitet werden. JENA ermöglicht die Speicherung von Modellen in einer relationalen Datenbank; weitere Persistenzdienste können hinzugefügt werden. Auch die Verarbeitung von virtuellen Modellen, die z.B. aus Verzeichnisdiensten oder durch das Parsen einer Webseite erzeugt werden, ist möglich (Carroll u. a., 2004, S. 76). Für die Abfrage von Modellen wird die Abfragesprache SPARQL (W3C, 2008) unterstützt.

JENA enthält eine einfache Reasoning-Implementierung; es besteht auch eine Schnittstelle für externe Reasoning-Werkzeuge (Carroll u. a., 2004, S. 79).

#### 2.5.2.3. KAON2

KAON2 ist ein für den nichtkommerziellen Einsatz frei verfügbares, aber nur als Binärdistribution erhältliches API zur Entwicklung von ontologiebasierten Anwendungen. Es wurde vom Forschungszentrum Informatik, der Universität Karlsruhe und der Universität Manchester entwickelt. KAON2 kann auch als Reasoning-System verwendet werden; die Eigenschaften von KAON2 als Reasoning-System werden in Kapitel 2.5.4.3 dargestellt.

Da der Schwerpunkt von KAON2 auf Reasoning liegt, wird nur OWL Lite und OWL DL, nicht aber OWL Full unterstützt. Außerdem können Ontologien in den Sprachen F-Logic und SWRL verarbeitet werden. Die Ontologien können in Dateien oder in einer relationalen Datenbank abgelegt werden. Wie JENA unterstützt auch KAON2 die Abfrage von Ontologien mit SPARQL (Motik und Sattler, 2006, S. 230).

#### 2.5.2.4. Protégé

Das Open-Source-Werkzeug Protégé kann als Editor für Ontologien verwendet werden (siehe Kapitel 2.5.1.1); da das Werkzeug in Form von Open-Source-Software vorliegt und nach einem strikt modularen Ansatz entwickelt ist, kann es darüber hinaus auch als Framework für die Entwicklung von ontologiebasierten Anwendungen eingesetzt werden. Tatsächlich war dies auch die ursprüngliche Intention bei der Entwicklung von Protégé:

> „Protégé is neither an expert system itself nor a program that builds expert systems directly; instead, Protégé is a tool that helps users build *other tools* that are

> custom-tailored to assist with knowledge acquisition for expert systems in specific application areas.“ (Musen, 1989, S. 2).

Der Fokus von Protégé liegt auf der Bereitstellung von GUI-Komponenten, mit denen Ontologien bearbeitet werden können. Anwendungen, die dem Nutzer direkten Zugriff auf Ontologien bieten sollen, können daher mit Protégé rasch entwickelt werden, da es als einziges der hier behandelten Frameworks fertige GUI-Komponenten enthält (Cardoso, 2006, S. 356). Die Ontologien können entweder als Dateien oder in einer relationalen Datenbank gespeichert werden.

Ein umfangreicher Vergleich der in diesem Kapitel vorgestellten Frameworks findet sich bei Rebstock u. a. (2008, S. 139 ff.).

### 2.5.3. Persistenzlösungen

Die einfachste Möglichkeit, Ontologien zu speichern, ist die Ablage in Dateiform. Allerdings ist diese Lösung nicht besonders performant, da für jeden Zugriff auf Elemente einer Ontologie die gesamte Datei neu gelesen und verarbeitet werden muss. Daher werden performantere Persistenzlösungen, auch *Triple Stores* genannt (Shadbolt u. a., 2006, S. 98), entwickelt. Diese ermöglichen meist den Zugriff auf persistente Ontologien mit Hilfe einer standardisierten Abfragesprache wie SPARQL (W3C, 2008).

#### 2.5.3.1. Datenbankbasierte Lösungen

Einige Frameworks ermöglichen es, Ontologien direkt in relationalen Datenbanken abzulegen. Dabei wird meist die Tripel-Darstellung verwendet; die Tripel werden dabei in Datenbanktabellen mit drei Spalten abgelegt. Da fast alle relationalen Datenbanksysteme über die Möglichkeit verfügen, einen Index über Spalten einer Tabelle aufzubauen, um diese schneller durchsuchen zu können, kann so effizient auf jedes Tripel zugegriffen werden. Weitere Optimierungen können erfolgen, indem für Aussagen mit häufig benötigten Prädikaten, z.B. `rdf:type`-Statements, gesonderte Tabellen aufgebaut werden (Heymans u. a., 2008, S. 90 ff.).

Neben der Optimierung der Antwortzeit bei Abfragen auf persistenten Ontologien können bei der Speicherung von Ontologien in Datenbanken auch Aspekte wie Transaktionsmanagement bei Änderungen, die mehrere Tripel betreffen, oder eine performante Möglichkeit, eine große Ontologie initial in eine Datenbank zu laden (sogenanntes *Bulk Loading*), eine Rolle spielen (Wilkinson u. a., 2003, S. 145).

#### 2.5.3.2. Sesame

Dedizierte Persistenzlösungen verwenden meist auch Datenbanken zur Speicherung von Ontologien, verfügen aber über zusätzliche Optimierungsmechanismen, um Anfragen schneller beantworten zu können. Eine häufig eingesetzte Persistenzlösung ist das Open-Source-System *Sesame*, ein Persistenzdienst für RDF-Daten. Sesame ist ein Serversystem, in dem Ontologien gespeichert werden können und das Anfragen auf diesen Ontologien bearbeitet. Dazu werden verschiedene Schnittstellen wie HTTP, RMI und SOAP bereitgestellt, die über einen Application Server angesprochen werden.

Die eigentliche Speicherung kann bei Sesame in verschiedenen Datenbanksystemen erfolgen. Eine Abfrage auf Ontologien wird dazu in eine oder mehrere Datenbankabfragen umgesetzt, wobei ein Optimierungsalgorithmus dafür sorgt, dass möglichst wenige und möglichst effizient bearbeitbare Datenbankabfragen entstehen (Broekstra u. a., 2002, S. 60 f.).

*OWLIM* ist ein System, das Sesame um Inferenz-Verfahren für OWL und RDFS erweitert. Damit können auch Fakten, die in der Ontologie nicht explizit enthalten sind, abgefragt werden. Es existiert in zwei Varianten: bei *SwiftOWLIM* wird die Inferenz im Hauptspeicher durchgeführt, was die Skalierbarkeit begrenzt, bei *BigOWLIM* wird die Inferenz auf persistenten Daten durchgeführt (Ognyanoff u. a., 2006, S. 18 ff.). Die Speicherung erfolgt bei OWLIM nicht in einer relationalen Datenbank, sondern in Tripel- und Binärdateien (Kiryakov u. a., 2005, S. 182 ff.).

#### 2.5.3.3. 3Store

Das Open-Source-System *3Store* wurde mit dem Ziel entwickelt, einen auch für große Datenmengen performanten Persistenzdienst zu schaffen. Es ist in C geschrieben und verwendet eine MySQL-Datenbank zur Speicherung (Harris und Gibbins, 2003, S. 2 ff.).

Um die Antwortzeit von 3Store zu optimieren, wird mit Hilfe zusätzlicher Hash-Tabellen ein effizienter Zugriff auf Tripel umgesetzt, und Abfragen werden, wie auch bei Sesame, in optimierte SQL-Abfragen übersetzt (Harris und Shadbolt, 2005, S. 236 ff.). Außerdem werden einige aus den gegebenen Aussagen abgeleitete Informationen zusätzlich in der Datenbank gespeichert, um die Abfragen weiter zu optimieren (Harris und Gibbins, 2003, S. 7 ff.).

Es wurden bereits umfangreiche Vergleiche verschiedener Persistenzlösungen, insbesondere zur Performance und zur Skalierbarkeit, durchgeführt, siehe dazu z.B. Lee (2004) und Liu und Hu (2005). Heymans u. a. (2008) geben einen Überblick über Strategien zur Optimierung der Performance von Persistenzlösungen.

### 2.5.4. Reasoning- und Inferenz-Werkzeuge

*Reasoning* oder *Inferenz* bezeichnet das logische Schließen aus vorhandener Information. Um Schlussfolgerungen aus der in Ontologien definierten Information zu ermöglichen, werden Reasoning- oder Inferenzwerkzeuge verwendet[9].

Als Ursprung des Reasoning kann man Aristoteles' Syllogismen ansehen (Sowa, 2000, S. 3ff). Diese Syllogismen sind Regeln, mit denen aus vorhandenen Aussagen neue abgeleitet werden können. So lässt sich z.B. aus *Alle Menschen sind sterblich* und *Sokrates ist ein Mensch* die Aussage *Sokrates ist sterblich* ableiten (Kunzmann u. a., 2003, S. 47).

Reasoning-Werkzeuge können Anfragen an Ontologien beantworten. Da nur Ontologien in OWL Lite und OWL DL entscheidbar sind, können in der Regel nur Ontologien in diesen Dialekten verarbeitet werden. Anfragen können z.B. mit Anfragesprachen wie SPARQL (W3C,

[9] Die Begriffe *Reasoning* und *Inferenz* werden teilweise synonym verwendet. In der deutschsprachigen Literatur findet sich bisweilen auch der Begriff *Ontologie-Beweiser*. In dieser Arbeit wird im Folgenden stets der Begriff *Reasoning* verwendet.

2008) formuliert werden. Typische Anfragen sind die Überprüfung der Konsistenz, also der Widerspruchsfreiheit, einer Ontologie, die Darstellung der Beziehung zwischen zwei Klassen oder die Aufzählung aller Instanzen einer Klasse. Diese Anfragen können auf dasselbe logische Problem, nämlich die Überprüfung der Widerspruchsfreiheit, zurückgeführt werden (Hitzler u. a., 2008, S. 176 f.).

Ein gängiger Mechanismus zur Überprüfung der Widerspruchsfreiheit ist das sogenannte *Tableau-Verfahren*. Bei diesen werden so lange weitere Aussagen aus den in einer Ontologie vorhandenen Aussagen gefolgert und in einem *Tableau* gespeichert, bis entweder ein Widerspruch entsteht oder keine weiteren Konsequenzen mehr erzeugt werden können. Die meisten Reasoning-Werkzeuge arbeiten nach diesem Verfahren (Hitzler u. a., 2008, S. 177 ff.).

Neben einem oder mehreren APIs in bestimmten Programmiersprachen und/oder einer GUI-basierten Schnittstelle unterstützen die meisten Reasoning-Werkzeuge die *DIG-Schnittstelle* (von **D**L **I**mplementation **G**roup). Dabei handelt es sich um eine HTTP-basierte Schnittstelle, mit Hilfe derer Reasoning-Dienste genutzt werden können (Bechhofer u. a., 2003a).

#### 2.5.4.1. Pellet

*Pellet* ist ein Open-Source-Reasoning-Werkzeug, das an der Universität Maryland entwickelt wurde. Es basiert auf dem oben dargestellten Tableau-Verfahren.

Neben den Standard-Reasoning-Funktionen bietet Pellet einige weitere Funktionalitäten an. So kann das Werkzeug z.B. auch Aussagen über Literale mit XSD-Typen (siehe Kapitel 2.3.1) erschließen. Außerdem bietet es die Möglichkeit, Schlüsse über Aussagen zu ziehen, die in verschiedenen, mit $\varepsilon$-Connections (siehe dazu Kapitel 3.1.4.5) verbundenen Ontologien enthalten sind. Außerdem können Ontologien, die *keine* gültigen OWL-DL-Ontologien sind, geladen und automatisch repariert werden (Sirin und Parsia, 2004).

#### 2.5.4.2. Racer

*Racer* wurde in einem gemeinsamen Projekt der Concordia University in Montreal und der TU Hamburg-Harburg entwickelt. Mittlerweile wird das Werkzeug als kommerzielle Lösung von der Firma Racer Systems vertrieben.

Racer kann Ontologien in DAML+OIL und OWL verarbeiten. Die Anwendung läuft als Server-Applikation, die über eine TCP- und eine HTTP-Schnittstelle angesprochen werden kann; es gibt Bibliotheken in C++ und Java zur Nutzung der Schnittstelle. Es existieren auch verschiedene graphische Clients, mit denen die Funktionalität von Racer genutzt werden kann (Haarslev und Möller, 2003a, S. 28 ff.).

Darüber hinaus enthält das System verschiedene Funktionalitäten, die zur Arbeit mit Ontologien, die sich im Laufe der Zeit ändern, dienen. Dazu zählt ein Subscribe-Mechanismus, mit Hilfe dessen ein Client automatisch informiert werden kann, wenn eine vorher bestimmte Abfrage ein geändertes Ergebnis liefert. Auch die Synchronisation von (lesenden und schreibenden) Zugriffen mehrerer Clients auf dieselbe Ontologie ist möglich (Haarslev und Möller, 2003b, S. 94 f.).

#### 2.5.4.3. KAON2

Das Reasoning-System *KAON2* (siehe auch Kapitel 2.5.2.3) verwendet keinen Tableau-Algorithmus. Stattdessen wird eine Eingabeontologie zunächst in ein disjunktives Datalog-Programm übersetzt. Auf diesem Datalog-Programm wird dann ein vorhandener Datalog-Beweiser aufgerufen Dieses Verfahren ist prinzipiell äquivalent zum Tableau-Algorithmus, in der Praxis jedoch oft performanter (Hitzler u. a., 2008, S. 182 ff.).

KAON2 kann neben den Formaten in OWL Lite und OWL DL auch Ontologien in F-Logic und SWRL verarbeiten, die teilweise eine höhere Ausdrucksmächtigkeit haben (vgl. Kapitel 2.3.4). Anders als z.B. Pellet sieht es jedoch nur die Bearbeitung von Standardabfragen vor (Motik und Sattler, 2006, S. 230 ff.).

Einen umfassenden Vergleich verschiedener Reasoning-Werkzeuge hinsichtlich der Ergebnisqualität und der Performance gibt Liebig (2006).

### 2.5.5. Matching-Werkzeuge

Werden mehrere Ontologien parallel eingesetzt, können sie mit Hilfe von Ontology-Matching-Werkzeugen aufeinander abgebildet werden. Dies ist in Anwendungszusammenhängen sinnvoll, in denen mehrere Ontologien für die Erfüllung einer Aufgabe benötigt werden (Kalfoglou und Schorlemmer, 2005, S. 1), z.B. wenn verschiedene Datenquellen abgefragt werden sollen, die unterschiedliche als Ontologien vorliegende Metadatenformate verwenden, oder wenn Agenten kooperieren sollen, die unterschiedliche Ontologien zur Wissensrepräsentation verwenden. In diesen Fällen wird das Interoperabilitätsproblem, das mit Hilfe von Ontologien eigentlich gelöst werden soll, lediglich auf eine höhere Ebene verschoben (Euzenat und Shvaiko, 2007, S. 1).

Dass auf die Dauer die Einigung auf eine einzige global genutzte Ontologie erzielt werden kann, ist unrealistisch (Noy, 2004, S. 65). Daher müssen Wege gefunden werden, mit der parallelen Existenz verschiedener Ontologien umzugehen – Ontology-Matching-Werkzeuge sind hier ein wichtiger Baustein. Sie werden im folgenden Kapitel detailliert behandelt.

# 3. Ontology Matching

Wenn Konzepte derselben Domäne in unterschiedlichen Ontologien modelliert werden, so können sich diese Ontologien auf verschiedene Art unterscheiden. Diese Unterschiede bezeichnet man als *semantische Heterogenität*. Neben verschiedenen Ontologiesprachen können Unterschiede in den gewählten Begriffen, verschiedene Detaillierungsgrade und unterschiedliche Perspektiven auftreten (Bouquet u. a., 2004, S. 7f). Um dennoch Interoperabilität zu gewährleisten, müssen diese Ontologien aufeinander abgestimmt werden.

## 3.1. Grundlagen

Ziel des Ontology Matchings ist es, Entsprechungen zwischen verschiedenen Ontologien derselben Domäne zu finden. Solche Entsprechungen können verwendet werden, um die Ontologien parallel zu nutzen oder zusammenzuführen.

### 3.1.1. Arten von semantischer Heterogenität

Ausgangspunkt des Ontology Matchings sind zwei oder mehr Ontologien, die dieselbe Domäne mit unterschiedlichen Mitteln beschreiben, die also *semantisch heterogen* sind. Nach Klein (2001) lassen sich zwei Ebenen identifizieren, auf denen semantische Heterogenität vorliegen kann: die Sprach- und die Ontologieebene. Heterogenität auf der Sprachebene entsteht, wenn zwei Ontologien in verschiedenen Repräsentationssprachen vorliegen (vgl. Kapitel 2.3). Auf der Sprachebene unterscheidet Klein folgende Typen von Heterogenität:

*Syntaktische Heterogenität:* Der einfachste Fall von Heterogenität ist die syntaktische Heterogenität. Sie tritt auf, wenn zur Beschreibung derselben Konzepte in verschiedenen Sprachen unterschiedliche syntaktische Konstrukte verwendet werden. So werden z.B. Klassen in RDF-S mit `rdfs:class`, in OWL mit `owl:Class` definiert. Auch das Vorliegen von Ontologien in OWL-XML einerseits und in Tripel-Notation andererseits lässt sich als syntaktische Heterogenität auffassen. Diese Arten von Heterogenität sind am einfachsten aufzulösen (Klein, 2001, S. 53).

*Heterogenität in der logischen Repräsentation:* Haben zwei Sprachen unterschiedliche Sprachkonstrukte zur Modellierung zur Verfügung, so muss derselbe Sachverhalt unter Umständen auf verschiedene Arten beschrieben werden. So haben manche Sprachen die Möglichkeit, die Disjunktheit von zwei Klassen direkt zu modellieren, etwa mit einem Statement der Form `Class_A disjoint Class_B`, während dies in anderen Sprachen nur auf Umwegen möglich ist, z.B. mit Statements wie `Class_A subclass of (complement of Class_B)` (vgl. Klein, 2001, S. 53).

*Heterogenität von primitiven Sprachelementen:* Eine weitere, eher subtile Ursache von Heterogenität auf der Sprachebene ist die möglicherweise uneinheitliche Belegung von primitiven Ausdrücken. So werden bei der Syntax von DAML+OIL mehrere Statements der Form `<rdfs:domain>` zu einer Eigenschaft so interpretiert, dass diese Eigenschaft einen Wert aus der *Schnittmenge* der Domänen enthalten muss, während bei RDF Schema dieselbe Definition so interpretiert wird, dass der Wert aus der *Vereinigung* der Domänen stammen muss (Klein, 2001, S. 54).

Neben den Uneinheitlichkeiten auf sprachlicher Ebene existieren weitere mögliche Ursachen für Heterogenität – selbst dann, wenn zwei Ontologien in derselben Sprache modelliert sind. Diese Arten von Heterogenität auf der Ontologieebene werden von Klein wie folgt benannt:

*Heterogene Kodierung:* Heterogenität in der Kodierung bezeichnet unterschiedliche Darstellungsarten von Daten, z.B. verschiedene Datumsformate oder die Verwendung von Dezimalkommas oder -punkten.

*Terminologische Heterogenität:* Werden für ein Konzept in verschiedenen Ontologien verschiedene Begriffe, sogenannte Synonyme, verwendet, z.B. `Straßenbahn` und `Tram`, so liegt terminologische Heterogenität vor. Auch der umgekehrte Fall, die Verwendung von Homonymen, ist eine Art von terminologischer Heterogenität: hier wird derselbe Begriff für die Benennung unterschiedlicher Konzepte verwendet. So kann der Begriff `Bahn` in einer Ontologie für ein Verkehrssystem aus Schienen und Zügen verwendet werden, in einer anderen Ontologie nur für einen einzelnen Zug.

*Heterogenität in der Modellierung:* Ontologien, die dasselbe Sachgebiet beschreiben, werden meist von unterschiedlichen Personen (oder Personengruppen) entwickelt, die unterschiedliche Blickweisen auf den selben Sachverhalt haben. Daher werden dieselben Konzepte heterogen modelliert: Der Detailgrad (die Granularität) der Modellierung kann sich ebenso unterscheiden wie der Abdeckungsgrad einzelner Teilgebiete.

*Heterogenität der Konventionen:* Verschiedene Modellierungskonventionen können ebenfalls zu Heterogenitäten führen. Ein Beispiel hierfür ist die Verwendung von Attributen oder Subklassen: so kann in einer Ontologie die Klasse `Verkehrsmittel` mit den beiden disjunkten Subklassen `Massenverkehrsmittel` und `Individualverkehrsmittel` modelliert sein, während in einer anderen Ontologie lediglich die Klasse `Verkehrsmittel` mit einem Attribut existiert, über welches Massen- und Individualverkehrsmittel unterschieden werden. Dies kann dazu führen, dass ein Mapping heterogene Elemente, also z.B. eine Klasse auf einen Attributwert, abbilden muss (Ghidini u. a., 2007, S. 234 ff.). Ein weiteres Beispiel, das beim Ontology Matching Probleme bereiten kann, sind Konventionen in der Namensgebung: so können z.B. „sprechende“ ID-Tags, also ID-Tags mit einem aussagekräftigen Inhalt, verwendet werden, oder es kommen kryptische ID-Tags (z.B. laufende Nummern) in Verbindung mit „sprechenden“ Labels zum Einsatz (Massmann u. a., 2006, S. 111).

Ontology Matching beschäftigt sich ausschließlich mit der zweiten Art von Heterogenität, der Heterogenität auf Ontologieebene. Heterogenität auf sprachlicher Ebene wird dagegen durch

*Ontologienormalisierung* behoben (Noy, 2004, S. 65), indem z.B. Ontologien von einer Repräsentationssprache in eine andere konvertiert werden. Allerdings verfügen manche Ontology-Matching-Werkzeuge auch über integrierte Normalisierungsfunktionen, die als Vorverarbeitungsschritt aufgerufen werden (vgl. Kalfoglou und Schorlemmer, 2005, S. 5ff).

### 3.1.2. Begriffe

Auf dem Gebiet des Ontology Matching werden, wie auf vielen anderen Forschungsgebieten auch, in der Literatur oft verschiedene Begriffe für dieselben Ideen und gleiche Begriffe für unterschiedliche Ideen eingesetzt (Kalfoglou und Schorlemmer, 2005, S. 1). So wird z.B. die Abstimmung verschiedener Ontologien aufeinander im weitesten Sinne als *Ontology Mediation* (de Bruijn u. a., 2006, S. 95) oder *Ontology Coordination* (Bouquet u. a., 2004, S. 5) bezeichnet.

Ein gängiger Weg, um Ontologien aufeinander abzustimmen, ist die Nutzung von *Ontology Mappings*. Ein Mapping ist dabei die Aussage, dass bestimmte Elemente einer Ontologie zu Elementen einer anderen Ontologie in Beziehung stehen (Bouquet u. a., 2004, S. 13). Im einfachsten Fall ist ein Mapping eine Sammlung von Aussagen der Form, dass ein Element in einer Ontologie einem Element in einer anderen Ontologie entspricht (Euzenat, 2004, S. 700), aber auch andere Aussagetypen sind möglich. Abbildung 3.1 zeigt ein einfaches Mapping: das Element `Bus` aus Ontologie 1 wird mit dem gleichnamigen Element in Ontologie 2 gleichgesetzt, das Element `Bahn` mit dem Element `Zug`. Ein Mapping kann daher auch als Abbildung von Elementen in einer Ontologie auf Elemente einer anderen Ontologie betrachtet werden (Kalfoglou und Schorlemmer, 2005, S. 3).

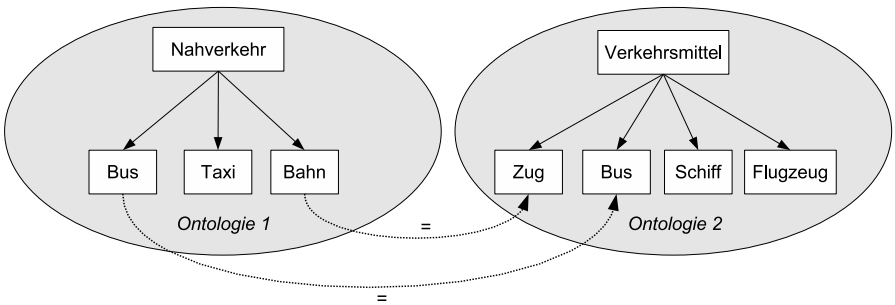



**Abbildung 3.1.:** Ein Beispiel für ein Mapping zwischen zwei Ontologien

Erzeugt man für jedes einzelne Mapping-Element der Form „`a` in $O_1$ entspricht `b` in $O_2$“ ein neues Element in einer virtuellen Ontologie $O_0$, so dass die Aussage in zwei Teilaussagen *a in* $O_1$ *entspricht x in* $O_0$ und *x in* $O_0$ *entspricht b in* $O_1$ zerlegt werden kann, so bezeichnet man die virtuelle Ontologie $O_0$ als *Artikulation* des Mappings (Kalfoglou und Schorlemmer, 2005, S. 4). Daher werden Matching-Werkzeuge gelegentlich auch als *Artikulationsgeneratoren* bezeichnet (Mitra und Wiederhold, 2002, S. 46)

Ein Ontology Mapping kann manuell erstellt oder automatisiert gefunden werden. Zum automatisierten Auffinden eines Mappings werden Matching-Verfahren eingesetzt (Shvaiko und Euzenat, 2005, S. 149). Abbildung 3.2 zeigt eine Black-Box-Darstellung von Matching-Systemen: Die Eingaben des Systems bestehen aus zwei Ontologien $O_1$ und $O_2$, optional einer Menge bereits bekannter Mapping-Elemente $M_0$ zwischen den Eingabeontologien sowie Para-

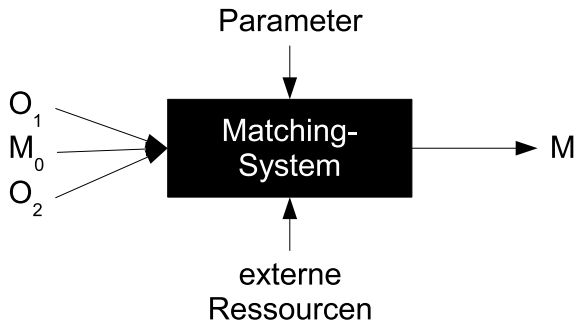



**Abbildung 3.2.:** Match-Operator als Black Box (nach Shvaiko und Euzenat, 2005, S. 149)

metern und weiteren externen Ressourcen. Als Ausgabe erzeugt das System ein Mapping $M$. Euzenat und Valtchev (2004, S. 333) definieren Matching wie folgt: „Gegeben zwei Ontologien, die je eine Menge von diskreten Einheiten beschreiben, finde die Beziehungen, die zwischen diesen Einheiten existieren“.

Die bereits bekannten Mapping-Elemente können von Experten manuell definiert worden sein, oder sie stammen aus einem anderen Matching-Vorgang. Auf diese Weise lassen sich mehrere Matching-Werkzeuge in Reihenschaltung kombinieren, wobei die nachgeschalteten die Ergebnisse der vorherigen weiter verfeinern (Euzenat u. a., 2008, S. 186). Als Spezialfall dieser Technik kann ein Werkzeug auch in mehreren Iterationen arbeiten, wobei die in der $n$-ten Iteration gefundenen Mapping-Elemente als Eingabe der $(n + 1)$-ten Iteration genutzt werden (Ehrig, 2007b, S. 74 f.).

Ein weiterer Begriff, der in der Literatur oftmals verwendet wird, ist *Ontology Alignment*. Damit wird einerseits eine Menge von Mapping-Elementen bezeichnet (z.B. Shvaiko und Euzenat, 2005, S. 149), andererseits die Ausführung des Matching-Verfahrens (Ehrig und Staab, 2004a, S. 683). Auch die Begriffe *Ontology Mapping* (z.B. Ehrig und Sure, 2004, S. 76, oder Kalfoglou und Schorlemmer, 2005, S. 3) und *Mapping Discovery* (z.B. Noy, 2004, S. 66) werden gelegentlich für das Matching-Verfahren verwendet.

Ontology Mappings können zusätzlich zu den unveränderten Ontologien gespeichert werden. Ausgangsontologien und Mappings zusammen können als virtuelle globale Ontologie aufgefasst werden (Kalfoglou und Schorlemmer, 2005, S. 8). Die Zusammenfassung kann aber auch in einer tatsächlichen, nicht-virtuellen neuen Ontologie erfolgen – diesen Vorgang nennt man *Ontology Merging*. Bei einem kompletten Merging wird eine von Grund auf neue Ontologie erzeugt, die Elemente aus beiden Ausgangsontologien enthält. Bei einer Bridge-Ontologie werden die beiden Original-Ontologien unverändert importiert und durch eine Reihe von Entsprechungsregeln, sogenannten *bridge axioms*, die auch als Mapping-Elemente aufgefasst werden können, ergänzt (de Bruijn u. a., 2006, S. 102ff., siehe auch Kapitel 3.1.4).

In dieser Arbeit wird *Ontology Matching* für den Prozess des Abbildens zweiter Ontologien verwendet. Das Resultat ist ein *Ontology Mapping*, eine Menge von *Mapping-Elementen*, die jeweils zwei oder mehr Elemente verschiedener Ontologien zueinander in Beziehung setzen.

### 3.1.3. Arten von Mappings

Grundsätzlich lassen sich einfache Mappings und komplexe Mappings zwischen Ontologien unterscheiden. Abbildung 3.3 zeigt ein Beispiel: Die Beziehung zwischen den beiden Elementen mit dem Bezeichner `Name` ist ein einfaches Mapping. Zwischen `Adresse` in Ontologie 1 und den drei Elementen `Straße`, `PLZ` und `Ort` in Ontologie 2 besteht dagegen eine komplexere Beziehung: das Element `Adresse` entspricht der Sequenz der drei anderen Elemente. Diese Beziehung lässt sich nicht als 1:1-Beziehung ausdrücken. Andere komplexe Beziehungen können auch arithmetische Ausdrücke mit einschließen, so dass z.B. `Bruttopreis` in einer Ontologie auf `Nettopreis * (1 + MwStSatz)` in einer anderen Ontologie abgebildet wird (Hu und Qu, 2006, S. 311).

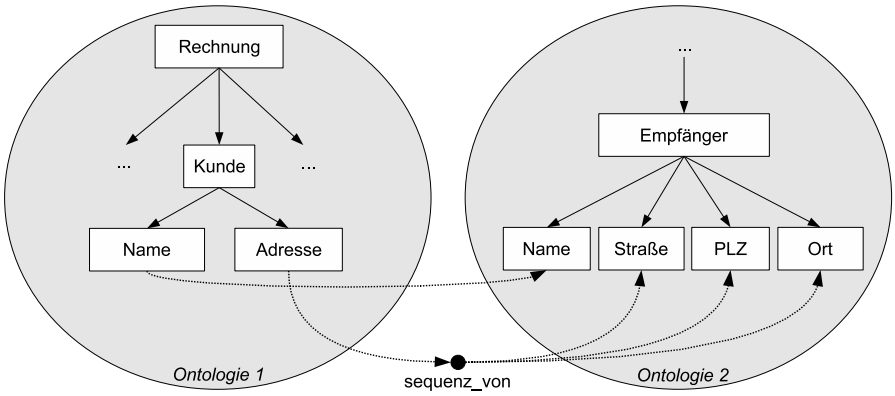



**Abbildung 3.3.:** Einfache und komplexe Mappings

Die meisten existierenden Matching-Verfahren sind darauf angelegt, einfache Mappings, also 1:1-Entsprechungen zwischen Elementen zu finden. Das Problem des Auffindens von komplexen Mappings wird, zur Abgrenzung vom einfachen Matching-Problem, manchmal auch als *Block Matching* bezeichnet (Hu und Qu, 2006). In dieser Arbeit werden nur Matching-Verfahren für einfache Mappings betrachtet.

Einfache Mappings lassen sich weiter in homogene und heterogene Mappings unterteilen (Ghidini u. a., 2007, S. 234 ff.). Homogene Mappings setzen nur gleichartige Elemente zueinander in Beziehung – Klassen zu Klassen, Instanzen zu Instanzen usw. Mit heterogenen Mappings lassen sich auch Beziehungen zwischen Elementen unterschiedlicher Art ausdrücken: so kann z.B. eine Ontologie die Klassen `Besitzverhältnis`, `Besitzer` und `Immobilie` definieren, während eine zweite Ontologie nur die Klassen `Besitzer` und `Immobilie` und ein Property `besitzt` definiert. Die Beziehung zwischen der Klasse `Besitzverhältnis` einerseits und dem Property `besitzt` andererseits lässt sich nur als heterogenes Mapping beschreiben.

### 3.1.4. Repräsentation von Mappings

Zur Beschreibung von Mappings haben sich verschiedene Formalismen etabliert. Neben den weit verbreiteten elementaren Definitionen, die Mappings als Menge von Tupeln zueinander in Beziehung gesetzter Elemente auffassen, existieren auch Ansätze, bestehende Ontologiesprachen sowie Erweiterungen dieser Sprachen einzusetzen.

#### 3.1.4.1. Elementare Mapping-Definitionen

Einfache Mappings setzen immer zwei Elemente aus je einer Ontologie zueinander in Beziehung. Ein Mapping lässt sich damit als Menge von 4-Tupeln der folgenden Form beschreiben:

$$mapping := \langle id, entity_1, entity_2, R \rangle \tag{3.1}$$

In dieser Form ist *id* ein eindeutiger Bezeichner für das Mapping (in Form eines URI o.ä.), $entity_1$ und $entity_2$ sind die beiden Elemente, die durch das Mapping in Beziehung gesetzt werden (und die aus verschiedenen Ontologien stammen), und $R$ ist die Art der Beziehung, die durch das Mapping beschrieben wird. Die Entitäten, die durch ein Mapping in Beziehung gesetzt werden, werden in der Regel durch URIs identifiziert und referenziert (Euzenat, 2004, S. 700). Dabei kann es sich z.B. um Klassen, Instanzen oder Attribute handeln, wobei auch heterogene Mappings möglich sind (Bouquet u. a., 2004, S. 13, Shvaiko und Euzenat, 2005, S. 149). Die Art der Beziehung kann neben der Gleichheit auch „allgemeiner“, „weniger allgemein“ oder „disjunkt“ sein (Giunchiglia u. a., 2007, S. 4).

Die Definition lässt sich um ein Stärke- oder Konfidenzmaß $c$ erweitern, um auch Beziehungstypen wie „zu 90% gleich“ ausdrücken zu können (vgl. Bouquet u. a., 2004, S. 13, und Euzenat, 2004, S. 700):

$$mapping := \langle id, entity_1, entity_2, R, c \rangle \tag{3.2}$$

Das Stärkemaß $c$ ist dabei als ein Fuzzy-Grad zu verstehen (Bouquet u. a., 2004, S. 30f).

Weil Mappings in verschiedenen Kontexten unterschiedlich sinnvoll sein können, kann es hilfreich sein, diese Definition noch um einen weiteren Parameter zu erweitern, der die Angemessenheit des Mappings in einem bestimmten Kontext bezeichnet (Rebstock u. a., 2008, S. 155f. und S. 162ff.):

$$mapping := \langle id, entity_1, entity_2, R, c, a \rangle \tag{3.3}$$

Der Parameter $a$ bezeichnet dabei, wie angemessen das Mapping in einem bestimmten Kontext ist. Während die ersten fünf Parameter stets konstant sind, kann sich $a$ je nach Abfragekontext ändern. Der Wert von $a$ kann dabei z.B. aus Nutzerfeedback gewonnen werden (Rebstock u. a., 2008, S. 162ff.).

Diese elementaren Definitionen wurden von verschiedenen Autoren dazu genutzt, Repräsentationssprachen für Ontology Mappings zu finden. Die bekanntesten davon werden im Folgenden vorgestellt. Dabei wird jeweils das Beispiel in Abbildung 3.1 auf Seite 39 verwendet.

#### 3.1.4.2. RDF

Euzenat und Shvaiko (2007, S. 226ff.) schlagen vor, die 4-Tupel, wie in Formel 3.2 definiert, direkt in RDF abzulegen. Beispiel 3.1 zeigt eine mögliche Kodierung.

In diesem Beispiel werden jeweils zwei Elemente mit einer Gleichheitsrelation und dem Stärkemaß 1 verbunden. Außerdem werden einige Metadaten zum Mapping hinzugefügt: *level* beschreibt, ob es sich um einfache oder komplexe Mappings handelt, und mit *type* lassen sich bestimmte Einschränkungen treffen, z.B., dass jedes Element der ersten Ontologie höchstens zu einem Element der zweiten Ontologie in Beziehung gesetzt werden darf. Da nur URIs miteinan-

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY ontology1 "http://www.example.org/ontology1.owl"'>
  <!ENTITY ontology2 "http://www.example.org/ontology2.owl"'>
]>
<rdf:RDF xmlns="http://www.example.org/mapping">
<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>**</type>
  <onto1>&ontology1;</onto1>
  <onto2>&ontology2;</onto2>
  <map>
    <Cell>
      <entity1 rdf:resource="&ontology1;#Bus"/>
      <entity2 rdf:resource="&ontology2;#Bus"/>
      <measure rdf:datatype="xsd:float">1.0</measure>
      <relation>=</relation>
    <Cell>
    <Cell>
      <entity1 rdf:resource="&ontology1;#Bahn"/>
      <entity2 rdf:resource="&ontology2;#Zug"/>
      <measure rdf:datatype="xsd:float">1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
</Alignment>
</rdf>
```

**Quelltextbeispiel 3.1:** Mapping in RDF

der in Beziehung gesetzt werden, können auf diese Weise auch heterogene Mappings abgebildet werden.

#### 3.1.4.3. OWL

In einer in OWL kodierten Ontologie (siehe Kapitel 2.3.3) hat man die Möglichkeit, Mappings zu anderen Ontologien darzustellen (vgl. Euzenat und Shvaiko, 2007, S. 222f.). Dazu können Konzepte der Ontologie zum Beispiel als äquivalent zu Konzepten in anderen Ontologien definiert werden, wie Beispiel 3.2 zeigt. Eine solche Referenzierung ist prinzipiell in allen drei OWL-Dialekten (siehe Kapitel 2.3.3) möglich, wobei in OWL DL und OWL Full auch komplexere Beziehungen zwischen Klassen verschiedener Ontologien formuliert werden können.

Zunächst werden die beiden Ontologien, zwischen denen das Mapping besteht, in eine neue *Bridge-Ontologie* importiert. Mit Hilfe des `owl:equivalentClass`-Statements werden dann die Mappings hergestellt. Mit `owl:subClassOf` und, im Falle von OWL DL mit `owl:disjointWith`, können auch andere Relationstypen dargestellt werden. Komplexe Mappings lassen sich auf diese Weise so weit darstellen, wie OWL selbst die komplexen Beziehungen darstellen kann. So kann z.B., wie in Beispiel 3.3 gezeigt, definiert werden, dass eine Klasse in einer Ontologie der Schnittmenge zweier Klassen in einer anderen Ontologie entspricht.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY ontology1 "http://www.example.org/ontology1.owl"'>
  <!ENTITY ontology2 "http://www.example.org/ontology2.owl"'>
]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="&ontology1;">

  <owl:imports rdf:resource="&ontology1;"/>
  <owl:imports rdf:resource="&ontology2;"/>

  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:about="&ontology1;#Bus">
   <owl:equivalentClass rdf:resource="&ontology2;#Bus">
  </owl:Class>

  <owl:Class rdf:about="&ontology1;#Bahn">
   <owl:equivalentClass rdf:resource="&ontology2;#Zug">
  </owl:Class>
</rdf:RDF>
```

**Quelltextbeispiel 3.2:** Mapping in OWL

```
<owl:Class rdf:about="&ontology1;#AmphibienFahrzeug">
  <owl:equivalentClass>
    <owl:intersectionOf>
      <owl:Class rdf:about="&ontology2;#Auto"/>
      <owl:Class rdf:about="&ontology2;#Boot"/>
    </owl:intersectionOf>
  </owl:equivalentClass>
</owl:Class>
```

**Quelltextbeispiel 3.3:** Komplexes Mapping in OWL

Legt man OWL Full zu Grunde, so ist auch die Definition heterogener Mappings möglich, da Konzepte in OWL Full gleichzeitig als Klassen und Instanzen genutzt werden können (vgl. Kapitel 2.3.3). Die Nutzung von OWL hat u.a. den Vorteil, dass es als Standard etabliert ist und somit keine neuen Werkzeuge entwickelt werden müssen, um Mappings zu verarbeiten; vielmehr können vorhandene Werkzeuge für OWL wiederverwendet werden.

#### 3.1.4.4. C-OWL

*C-OWL* steht für *Context OWL*. Kontexte werden als lokale Modelle betrachtet, die mit Hilfe von *Bridge Rules* verbunden sind und in einer Sprache, die eine Erweiterung von OWL ist, notiert werden können (Bouquet u. a., 2003, S. 164ff.). Ein Mapping in C-OWL besteht dabei immer aus einem URI, Verweisen auf die Quell- und die Zielontologie sowie einer Menge von Bridge Rules, die Mapping-Elemente nach Definition 3.1 repräsentieren (Bouquet u. a.,

2003, S. 177). Das einfache Mapping aus Abbildung 3.1 in C-OWL ist in Quelltextbeispiel 3.4 dargestellt.

```
<cowl:mapping>
  <rdfs:comment>Example showing a simple mapping</rdfs:comment>
  <cowl:sourceOntology rdf:resource="&ontology1;"/>
  <cowl:targetOntology rdf:resource="&ontology2;"/>

  <cowl:bridgeRule cowl:br-type="equiv">
    <cowl:sourceConcept rdf:resource="&ontology1;#Bus"/>
    <cowl:targedConcept rdf:resource="&ontology2;#Bus"/>
  </cowl:bridgRule>

  <cowl:bridgeRule cowl:br-type="equiv">
    <cowl:sourceConcept rdf:resource="&ontology1;#Bahn"/>
    <cowl:targedConcept rdf:resource="&ontology2;#Zug"/>
  </cowl:bridgRule>
</cowl:mapping>
```

**Quelltextbeispiel 3.4:** Mapping in C-OWL

Der Ansatz mag dem OWL-Ansatz ähnlich erscheinen. Ein wichtiger Unterschied ist aber, dass für die Beschreibung der Entsprechungen keine OWL-Ausdrücke, sondern Konstrukte in der speziellen Sprache C-OWL verwendet werden. Dies ermöglicht die Definition eigener Beziehungstypen, die sich mit OWL nicht ausdrücken lassen (Euzenat und Shvaiko, 2007, S. 223f.). Auch heterogene Mappings sind auf diese Art beschreibbar. Allerdings werden stets nur einzelne, mit URI identifizierte Elemente aus Quell- und Zielontologie referenziert werden, daher unterstützt C-OWL keine komplexen Mappings.

Da C-OWL, anders als OWL, kein Standard ist, existieren bislang keine Werkzeuge, die C-OWL unterstützen (Stuckenschmidt u. a., 2004, S. 100).

#### 3.1.4.5. $\varepsilon$-Connections

Ähnlich wie C-OWL lassen sich $\varepsilon$-Connections verwenden, um Beziehungen zwischen OWL-Ontologien darzustellen. Bei $\varepsilon$-Connections handelt es sich um eine Erweiterung von OWL DL (Grau u. a., 2005, S. 5). Quelltextbeispiel 3.5 zeigt das Beispiel in $\varepsilon$-Connections.

In diesem Beispiel werden in Ontologie 1 die Verbindungen zu Ontologie 2 angelegt. Es ist aber prinzipiell ebenso möglich, mit $\varepsilon$-Connections eine Bridge-Ontologie anzulegen. Im Gegensatz zu der OWL-Lösung müssen jedoch keine ganzen Ontologien mittels `owl:import` importiert werden, statt dessen werden Elemente aus anderen Ontologien mit den Sprachkonstrukten `owl:ForeignClass` und `owl:foreignOntology` angesprochen. Damit werden nur genau die Teile der fremden Ontologien referenziert, die auch tatsächlich benötigt werden, ohne dass die nicht benötigten Teile importiert werden müssten. Dies ist insbesondere dann hilfreich, wenn die referenzierten Ontologien sehr groß sind (Grau u. a., 2005, S. 2).

Anders als C-OWL werden $\varepsilon$-Connections von einigen Tools unterstützt. Es gibt eine Erweiterung für das Programmierframework OWL-API (siehe Kapitel 2.5.2.1), für den Ontologie-Editor SWOOP (siehe Kapitel 2.5.1.1) und für den Reasoner Pellet (siehe Kapitel 2.5.4.1) (Grau u. a., 2006, S. 17 ff.).

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY ontology1 "http://www.example.org/ontology1.owl">
  <!ENTITY ontology2 "http://www.example.org/ontology2.owl">
]>
<rdf:RDF xml:base="&ontology1;"
 xmlns="&ontology1;#"
 xmlns:rdf="&rdf;#"
 xmlns:rdfs="&rdfs;#"
 xmlns:owl="&owl;#">

  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:about="#Bus">
    <owl:equivalentClass>
      <owl:ForeignClass rdf:about="&ontology2;#Bus">
        <owl:foreignOntology rdf:resource="&ontology2;"/>
      </owl:ForeignClass>
    </owl:equivalentClass>
  </owl:Class>

  <owl:Class rdf:about="#Bahn">
    <owl:equivalentClass>
      <owl:ForeignClass rdf:about="&ontolgoy2;#Zug">
        <owl:foreignOntology rdf:resource="&ontology2;"/>
      </owl:ForeignClass>
    </owl:equivalentClass>
  </owl:Class>

  <!-- further definitions of ontology 1 -->
  ...

</rdf:RDF>
```

**Quelltextbeispiel 3.5:** Mapping in $\varepsilon$-Connections

#### 3.1.4.6. SWRL

Die Semantic Web Rule Language (SWRL) ist eine Sprache, die Konstrukte aus OWL Lite und OWL DL um die Möglichkeit erweitert, Regeln zu definieren, die in OWL nicht direkt unterstützt werden, wie z.B. logische Implikationen und arithmetische Beziehungen (Horrocks u. a., 2004, siehe auch Kapitel 2.3.4). Mit Hilfe dieser Regeln lassen sich auch Beziehungen zwischen Ontologien darstellen.

Da die Sprachkonstrukte aus OWL auch in SWRL zur Verfügung stehen, lassen sich einfache Mappings genauso formulieren wie in OWL (siehe Quelltextbeispiel 3.2). Quelltextbeispiel 3.6 demonstriert, dass sich mit der Mächtigkeit von SWRL jedoch auch komplexe Mappings ausdrücken lassen.

In diesem Beispiel wird ausgedrückt, dass alle Objekte vom Typ `Book` in der zweiten Ontologie, deren Größe höchstens 14 ist, ein Objekt vom Typ `Pocket_book` in der ersten Ontologie

```
<ruleml:imp>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="&ontology2;#Book"/>
      <ruleml:var>b</ruleml:var>
    </swrlx:classAtom>
    <swrlx:datavaluedPropertyAtom swrlx:property="&ontology2;#size">
      <ruleml:var>b</ruleml:var>
      <ruleml:var>s</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:builtinAtom swrlx:builtin="&swrlb;#lessThanOrEqual">
      <ruleml:var>s</ruleml:var>
      <owlx:DataValue owlx:datatype="&xsd;#int">14</owlx:DataValue>
    </swrlx:builtinAtom>
  <ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom swrlx:property="ontology1;#Pocket_book">
    <ruleml:var>b</ruleml:var>
  </ruleml:_head>
</ruleml:imp>
```

**Quelltextbeispiel 3.6:** Komplexes Mapping in SWRL (aus Euzenat und Shvaiko, 2007, S. 225)

sind. Dies ist in OWL wegen der fehlenden arithmetischen Operatoren nicht möglich (vgl. Kapitel 2.3.3).

SWRL wird z.B. von dem Editor Protégé (siehe Kapitel 2.5.1.1) und dem Reasoning-System KAON2 (siehe Kapitel 2.5.4.3) unterstützt.

## 3.2. Einsatzbereiche

In Kapitel 2.4 wurden verschiedene Einsatzgebiete von Ontologien aufgezeigt. Dabei wurde stets stillschweigend davon ausgegangen, dass dem jeweiligen Einsatzszenario nur eine Ontologie zugrunde liegt. In der Praxis tritt dieser Fall jedoch selten ein – die Integration verschiedener Ontologien, bei der Ontology Matching eine zentrale Rolle spielt, ist daher eine der wichtigsten Aufgaben im Umgang mit Ontologien (Cardoso, 2007, S. 87)

Ontology-Matching-Verfahren können also überall dort zum Einsatz kommen, wo auch Ontologien eingesetzt werden – nämlich dann, wenn mehrere Ontologien von verschiedenen Parteien zur Beschreibung derselben Dinge verwendet werden. In diesem Kapitel werden die Szenarien aus Kapitel 2.4 noch einmal aufgegriffen und der Einsatz von Ontology Matching im jeweiligen Gebiet dargestellt.

### 3.2.1. Datenintegration

Daten können im Semantic Web in verschiedenen heterogenen Quellen – Dokumentensammlungen, Datenbanken usw. – vorliegen. In Kapitel 2.4.1 wurde beschrieben, wie Ontologien helfen können, diese Quellen zu vereinheitlichen.

Nicht immer wird für alle Quellen dieselbe Ontologie verwendet. Insbesondere, wenn Informationsquellen unabhängig voneinander, etwa in verschiedenen Unternehmen, existieren,

kommen auch verschiedene Ontologien zum Einsatz. Ein vereinheitlichter Zugriff auf diese Informationen muss dann verschiedene Ontologien handhaben können. Hierzu wird dem Nutzer eine auf nur *einer* Ontologie basierende Schnittstelle angeboten.

Abfragen, die der Nutzer mit Begriffen aus dieser Ontologie formuliert, werden dann mit Hilfe von Ontology Mappings in Abfragen in Begriffen der Ontologien der verschiedenen Datenquellen übersetzt. Die Rückgaben aus den einzelnen Quellen werden gesammelt und ebenfalls mit Hilfe von Ontology Mappings wieder in Begriffe der Nutzerontologie rückübersetzt. Diesen Vorgang bezeichnet man als *Query and Answer Rewriting* (Ehrig, 2007b, S. 41). Abbildung 3.4 zeigt den Gesamtprozess.

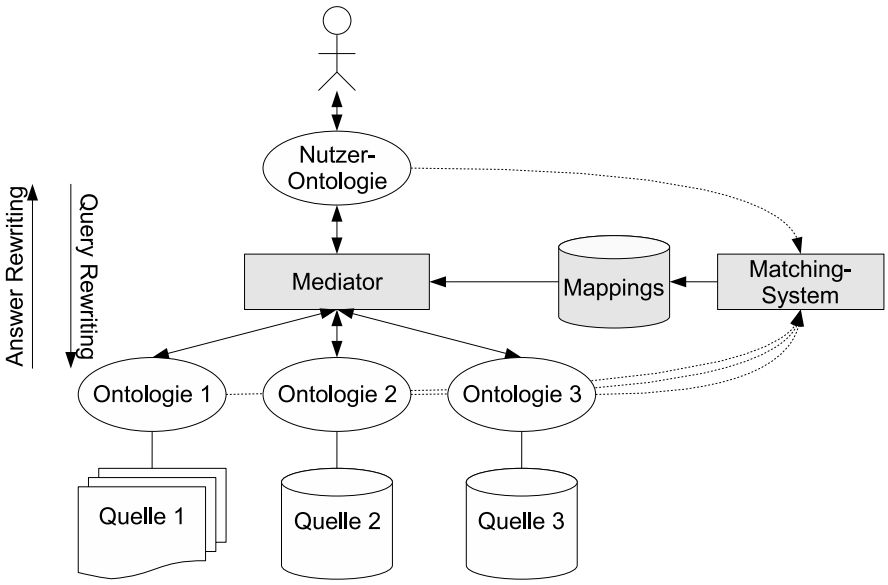



**Abbildung 3.4.:** Einsatz von Ontology Matching in der Informationsintegration (in Anlehnung an Euzenat und Shvaiko, 2007, S. 263)

Zur Durchführung der Übersetzung von Abfragen und Rückgaben, die mit Hilfe verschiedener Ontologien formuliert sind, kommt ein Mediator zum Einsatz. Dieser kann automatisch aus den von einem Matching-System gefundenen Mappings erzeugt werden (Euzenat und Shvaiko, 2007, S. 262 ff.).

### 3.2.2. Web Services und Agenten

In Kapitel 2.4.2 wurde dargestellt, wie Ontologien genutzt werden können, um Web Services semantisch zu beschreiben und so die Komposition zu komplexen Anwendungen zu vereinfachen und automatisierbar zu machen. Werden für verschiedene Web Services unterschiedliche Ontologien eingesetzt, so können diese nicht ohne zusätzliche Schritte interoperieren (Sycara und Paolucci, 2004, S. 360). Es müssen Mappings verwendet werden, um diese zu kombinieren (Meyer, 2005, S. 2f.). Diese können mit Ontology-Matching-Verfahren gewonnen werden.

Dabei können Ontology Mappings an drei verschiedenen Stellen zum Einsatz kommen: beim Auffinden eines passenden Dienstes, bei deren Aufruf und bei der Komposition von Diensten zur Lösung eines Problems (Burstein und McDermott, 2005, S. 1).

Abbildung 3.5 zeigt den Einsatz von Ontology-Matching-Werkzeugen für das Auffinden von Diensten: Anbieter von Diensten registrieren ihre Dienstangebote bei einem Vermittler und setzen jeweils eine Ontologie ein, um diese Dienste zu beschreiben. Ein Nachfrager, der einen bestimmten Dienst sucht und für die Beschreibung seiner Suchanfrage ebenfalls eine Ontologie einsetzt, wendet sich an den Vermittler. Dieser führt ein Matching der Ontologien von Anbietern und Nachfragern durch und kann so die für den Nachfrager relevanten Dienste ermitteln.

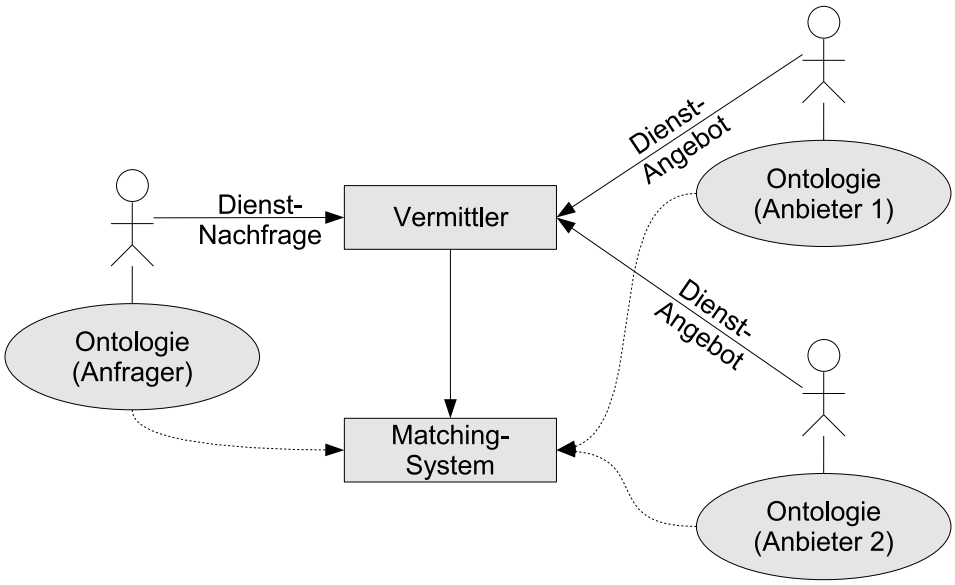



**Abbildung 3.5.:** Anwendung von Ontology Matching beim Auffinden von Web Services

Auch für den Aufruf von Web Services können Ontology Mappings zum Einsatz kommen. Ein Aufruf, in dem Konzepte aus der Ontologie des Clients verwendet werden, wird dabei in einen Aufruf umgesetzt, in dem Konzepte aus der Ontologie des Dienstanbieters zum Einsatz kommen. Dies entspricht dem oben dargestellten Query and Answer Rewriting in der Informationsintegration (vgl. Burstein und McDermott, 2005, S. 5).

Agenten, die automatisiert bestimmte komplexere Aufgaben lösen, können dazu Dienste in Anspruch nehmen. Werden zu jedem Dienst Vorbedingungen und Effekte beschrieben, können Agenten Pläne finden, wie sie ihre Aufgaben unter Zuhilfenahme verschiedener Dienste lösen können (siehe Russell und Norvig, 2003, S. 375 ff.). Indem die Ontologien der infrage kommenden Dienstanbieter und des Agenten mit Hilfe von Mappings kombiniert werden, können solche Pläne automatisch gefunden und ausgeführt werden (Burstein und McDermott, 2005, S. 5 ff.). Dies setzt voraus, dass die Mappings schnell verfügbar und korrekt sind (Ehrig, 2007b, S. 39).

Der Bereich kommunizierender und mobiler Kleincomputer, das sogenannte *Ambient Computing*, stellt einen weiteren Anwendungsbereich für Agentenkommunikation dar, in dem Ontology Matching eine Rolle spielt. In ihrem oft zitierten Artikel stellen Berners-Lee u. a. (2001, S. 45) ein solches Szenario dar: Das Telefon klingelt, und automatisch werden alle Geräte, die eine Lautstärkeregelung besitzen, leiser gestellt. Hier werden über ein drahtloses Netz verschiedene Dienste angesprochen, die ein bestimmtes Konzept, nämlich eine Lautstärkeregelung, enthalten. Da dieses Konzept bei Geräten verschiedener Hersteller unterschiedlich beschrieben

sein kann, muss auch hier, wenn man Ontologien als grundlegenden Beschreibungsmechanismus verwendet, Ontology Matching eingesetzt werden (Euzenat und Shvaiko, 2007, S. 22).

### 3.2.3. E-Business

Um Daten, die im elektronischen Geschäftsverkehr ausgetauscht werden, eindeutig interpretierbar zu machen, werden oft E-Business-Standards genutzt. Diese enthalten z.B. Begriffsdefinitionen für Produkte und Dienstleistungen und ermöglichen die Erstellung standardisierter, eindeutiger Geschäftsdokumente. Werden nun von zwei Geschäftspartnern verschiedene Standards benutzt, so müssen die genutzten Begriffe von beiden Partnern zweifelsfrei verstanden werden können. Dazu ist es nötig, die Begriffe von einem Standard in den anderen zu übersetzen (Rebstock u. a., 2005, S. 27).

E-Business-Standards können als Ontologien angesehen werden (Gómez-Pérez u. a., 2004, S. 86). Damit wird die Frage, was ein Begriff aus einem Standard in einem anderen bedeutet, zu einem Ontology-Matching-Problem (Omelayenko, 2001, S. 119 ff.). Einige E-Business-Standards liegen bereits in Ontologiesprachen vor, etwa die Produktklassifikationsstandards UNSPSC in RDF-S und DAML+OIL (Klein, 2002) und eCl@ss in OWL (Hepp, 2006). Am Beispiel solcher Produktklassifikationsstandards lässt sich zeigen, wie Ontology-Matching-Verfahren im E-Business eingesetzt werden können.

Auf elektronischen Marktplätzen können verschiedene Anbieter Waren anbieten. Damit die Waren vergleichbar bleiben, ist es sinnvoll, einen gemeinsamen E-Business-Standard (z.B. eCl@ss) zur Auszeichnung der Waren zu nutzen. Dies ermöglicht bei geeigneten Schnittstellen auch eine automatisierte Suche durch Agenten. Da sich mit eCl@ss zusätzlich Produktattribute beschreiben lassen, können auch detaillierte Suchanfragen formuliert werden (z.B. *Finde Digitalkameras mit mindestens fünf Megapixel Auflösung und achtfachem optischem Zoom*) (Schreder u. a., 2007, S. 3 ff.).

Damit Anbieter und Nachfrager den Marktplatz sinnvoll nutzen können, ist es notwendig, dass sie ihre eigenen Produktstandards mit dem vom Marktplatz unterstützten Standard in Form eines Ontology Mappings abgleichen. Auf diese Weise kann ein Konsens über die Produktbeschreibungen hergestellt werden, selbst wenn mehrere Produktstandards parallel im Einsatz sind (Corcho und Gómez-Pérez, 2001, S. 131 ff.). Um dieses Mapping zu erstellen, können Ontology-Matching-Verfahren eingesetzt werden (Euzenat und Shvaiko, 2007, S. 13 f.). Abbildung 3.6 zeigt eine mögliche Systemarchitektur.

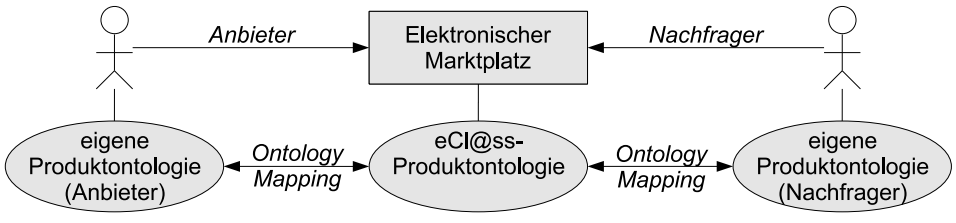



**Abbildung 3.6.:** Anwendung von Ontology Mappings für elektronische Marktplätze (in Anlehnung an Schreder u. a., 2007, S. 3)

Mit einer solchen Architektur ist der Nutzer in der Lage, auch detaillierte Suchanfragen zu formulieren und darauf sinnvolle Antworten zu bekommen. In einer weiteren Ausbaustufe

können die semantischen Auszeichnungen auch dazu genutzt werden, automatisierten Agenten den weiteren Geschäftsablauf, z.B. die Verhandlung mit den Anbietern und den Vertragsabschluss, zu übertragen (Dustdar u. a., 2006, S. 4 f.). Um eine dynamische Nutzung zu erlauben, können die Agenten auch in die Lage versetzt werden, bei Bedarf selbst ein Matching mit den Ontologien der Geschäftspartner durchzuführen (Malucelli u. a., 2005, S. 36). Dadurch entfällt die Notwendigkeit, das Matching im Vorfeld für alle potentiellen Verhandlungspartner durchzuführen und die entsprechenden Mappings vorzuhalten.

## 3.3. Verfahren

Um Ontology Mappings mit einem Matching-System automatisiert zu finden, existieren verschiedene Ansätze, die meist auf Heuristiken basieren (Noy, 2004, S. 66). Grob unterscheiden lassen sich schema- und instanzbasierte Verfahren (de Bruijn u. a., 2006, S. 100). Erstere verwenden nur die Ontologien selbst, letztere arbeiten mit Dokumenten, die mit diesen Ontologien annotiert wurden. Da in vielen Anwendungsfällen keine annotierten Dokumente vorhanden sind, lassen sich instanzbasierte Verfahren nicht immer anwenden. Daher werden in dieser Arbeit nur schemabasierte Verfahren betrachtet.

Ein dem schemabasierten Ontology Matching verwandtes, älteres Problem ist das Zusammenführen von Datenbanken, die Schemaintegration. Viele Verfahren, die im Bereich der Datenbank-Schemaintegration entwickelt wurden, finden sich daher im Ontology Matching wieder (Kalfoglou und Schorlemmer, 2005, S. 20f).

Nach Shvaiko und Euzenat können schembasierte Matching-Verfahren wie in Abbildung 3.7 klassifiziert werden. Diese Klassifikation basiert auf der Taxonomie von Rahm und Bernstein (Rahm und Bernstein, 2001, S. 373).

Die in der Übersicht dargestellten Verfahren werden in Matching-Werkzeugen oft zu hybriden und zusammengesetzten Matchern kombiniert (Rahm und Bernstein, 2001, S. 337). Diese stellen aber im engeren Sinne keine eigenen Verfahren dar.

Shvaiko und Euzenat teilen schemabasierte Verfahren in elementbasierte und strukturbasierte Verfahren, wobei diese jeweils noch einmal in interne und externe Verfahren unterteilt werden. Bei internen Verfahren werden nur die Ontologien selbst als Eingabedaten verarbeitet; bei externen Verfahren werden weitere externe Ressourcen in den Match-Operator mit einbezogen (vgl. Abbildung 3.2)[1].

### 3.3.1. Elementbasierte Verfahren

Elementbasierte Verfahren führen paarweise Vergleiche einzelner Elemente von Ontologien durch. Dabei gibt es feingranulare Verfahren, die als Elemente einzelne Entitäten von Ontologien auffassen, und grobgranulare Verfahren, die als Elemente größere Strukturen von Ontologien verwenden (Rahm und Bernstein, 2001, S. 338f.), wenngleich die meisten element-

[1] Shvaiko und Euzenat bezeichnen interne Verfahren als „syntactic techniques“ (Shvaiko und Euzenat, 2005, S. 155). Da der Begriff aber mit „external techniques“ ein Gegensatzpaar bildet, habe ich die Übersetzung „interne Verfahren“ gewählt, die zudem verdeutlicht, dass nur Information aus den Ontologien selbst, also interne Information, genutzt wird.

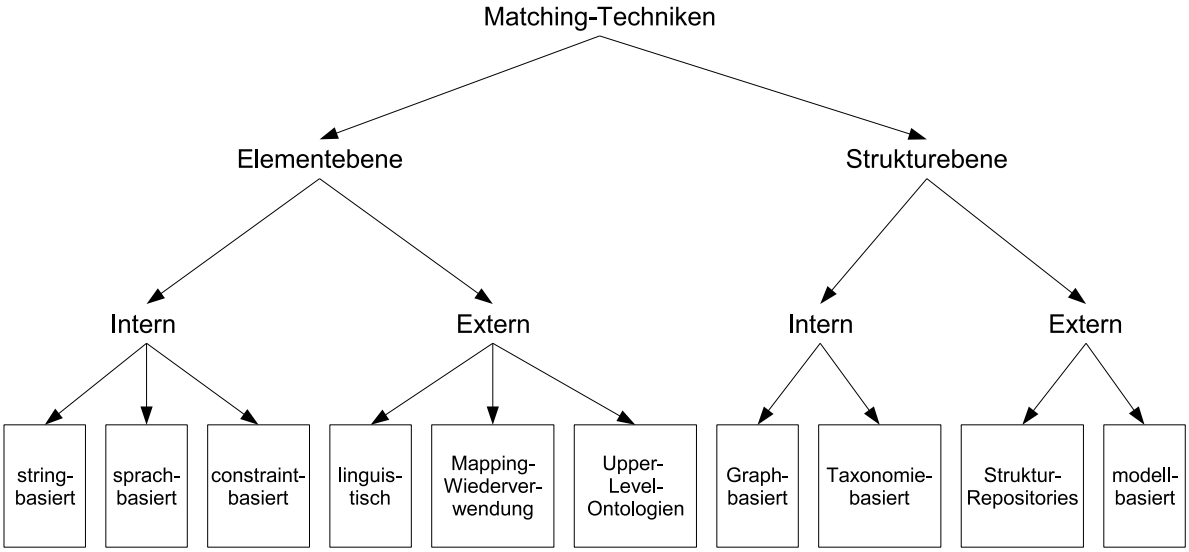



**Abbildung 3.7.:** Klassifikation von Matching-Verfahren (nach Shvaiko und Euzenat, 2005, S. 154)

basierten Verfahren auf der feingranularen Ebene arbeiten. Die Unterscheidung von internen und externen Verfahren heißt bei Giunchiglia und Shvaiko (2003, S. 275) schwache und starke semantische Verfahren.

Bei elementbasierten Verfahren wird in erster Linie der Name eines Elements verarbeitet[2]. Dabei wird jedes Element isoliert von anderen Elementen betrachtet, strukturelle Relationen wie Subtyp-Supertyp-Beziehungen werden nicht berücksichtigt. Interne elementbasierte Verfahren sind (Shvaiko und Euzenat, 2005, S.156f.):

*Stringbasierte Verfahren:* Stringbasierte Verfahren führen Vergleiche der Bezeichner und Beschreibungen von Elementen aus. Dabei wird ein Ähnlichkeitsmaß für die Bezeichner berechnet, das dann meist direkt als Stärkemaß der Referenz verwendet wird (siehe Formel 3.2 auf Seite 42). Einfache Ähnlichkeitsmaße vergleichen z.B. die Länge gemeinsamer Prä- und Suffixe (Shvaiko und Euzenat, 2005, S. 156). Komplexere Verfahren messen die *edit distance*, d.h., die Anzahl der Bearbeitungsschritte, die benötigt werden, um einen Bezeichner in den anderen zu überführen, oder vergleichen n-Gramme, d.h., Substrings fester Länge (Euzenat und Shvaiko, 2007, S. 77 ff.). Eine Übersicht von Ähnlichkeitsmaßen für stringbasierte Verfahren findet sich bei Cohen u. a. (2003).

*Sprachbasierte Verfahren:* Eine Erweiterung von stringbasierten Verfahren stellen sprachbasierte Verfahren dar. Sie führen zusätzliche Verarbeitungsschritte durch. Dazu zählen das Zerlegen der zu vergleichenden Bezeichner in einzelne sprachliche Token, das Zu-

[2]Es gibt unterschiedliche Methoden, den Namen eines Elements zu vergeben, z.B. als URI des Elements, in einem separaten `rdfs:label`-Element oder in einem proprietären Element, z.B. `product:name` (vgl. Kapitel 3.1.1). Verwenden zwei Ontologien unterschiedliche Methoden, insbesondere proprietäre Namenslemente, so arbeiten elementbasierte Verfahren in der Regel eher schlecht.

rückführen von Token auf ihre Grundform, das heißt, das Entfernen von Flexionsendungen (Giunchiglia u. a., 2004, S. 68), das Eliminieren irrelevanter Token, sogenannter *stop words* (z.B Artikel und Konjunktionen) (Shvaiko und Euzenat, 2005, S. 157), und das Ersetzen von diakritischen Zeichen, z.B. Umlauten und Zeichen mit Akzenten – `München` wird zu `Muenchen`, `Montréal` zu `Montreal` usw. (Euzenat und Shvaiko, 2007, S. 76 ff.). Auf diese vorverarbeiteten Strings können dann stringbasierte Verfahren angewandt werden (Cross, 2003, S. 137). Obwohl die Kenntnis über Flexionsendungen und die Unterscheidung in relevante und irrelevante sprachliche Token genau genommen externes Wissen ist, werden diese Verfahren von Shvaiko und Euzenat zu den internen Verfahren gezählt.

Zu den sprachbasierten Verfahren sind auch phonetische Vergleiche, z.B. nach dem *Soundex*-Verfahren (siehe Holmes und McCabe, 2002), zu zählen, die die Ähnlichkeit von zwei Bezeichnern anhand deren Aussprache messen (Giunchiglia und Shvaiko, 2003, S. 275).

*Constraint-basierte Verfahren: Constraints* sind Eigenschaften von Elementen in Ontologien, die über deren Namen hinausgehen. Dazu zählen Datentypen und deren Wertebereiche – haben zwei Elemente den gleichen Wertebereich, so sind sie mit einer gewissen Wahrscheinlichkeit als gleich anzusehen. Dasselbe gilt für mengenwertige Elemente, für die gleiche oder ähnliche Kardinalitätsbeschränkungen gelten (Shvaiko und Euzenat, 2005, S. 157). Constraintbasierte Verfahren werden meist mit anderen Verfahren kombiniert (Cross, 2003, S. 137f.), da sie allein zu wenig aussagekräftig sind: dass z.B. zwei Elemente `Hausnummer` und `Geburtsjahr` numerischen Typs sind, macht ein Mapping zwischen diesen Elementen noch nicht sinnvoll. Hätte man allerdings mit einem stringbasierten Verfahren festgestellt, dass `Hausnr.` und `Hausnummer` mit einer gewissen Wahrscheinlichkeit verwandt sind, so würde ein constraintbasiertes Matching diese Aussage untermauern und insgesamt wahrscheinlicher machen (vgl. Euzenat und Shvaiko, 2007, S. 93).[3]

Externe elementbasierte Verfahren vergleichen ebenfalls einzelne Elemente, beziehen aber externe Quellen in diesen Vergleich mit ein. Zu externen elementbasierten Verfahren zählen:

*Linguistische Verfahren:* Linguistische Verfahren nutzen Thesauri und Wörterbücher, um Ähnlichkeiten zwischen Elementen zu finden. Dies können sowohl allgemeine wie auch domänenspezifische Fachthesauri sein (Shvaiko und Euzenat, 2005, S. 157f.). Gebräuchliche einsprachige allgemeine Thesauri sind WordNet (Wordnet, 2007) für Englisch und GermaNet für Deutsch (Hamp und Feldwig, 1997). Ein bekannter mehrsprachiger Thesaurus ist EuroWordNet, der die Sprachen Holländisch, Spanisch, Italienisch, Englisch, Französisch, Deutsch, Tschechisch und Estnisch umfasst (Vossen, 1999, S. 6).

[3] Bei Euzenat u. a. (2008, S. 184) werden Constraint-basierte Verfahren unter den strukturbasierten Verfahren eingeordnet. Tatsächlich stellen sie einen Grenzfall dar: wenn eine Ontologie etwa die Aussage enthält, dass `Hans` eine Instanz vom Typ `Maurer` ist, dann kann das entweder als Eigenschaft des Konzeptes `Hans` oder als Verbindung zwischen den beiden Konzepten `Hans` und `Maurer` betrachtet werden. Je nach Sichtweise würde man ein Matching-Verfahren, das diese Information nutzt, dann zu den element- oder zu den stukturbasierten Verfahren zählen.

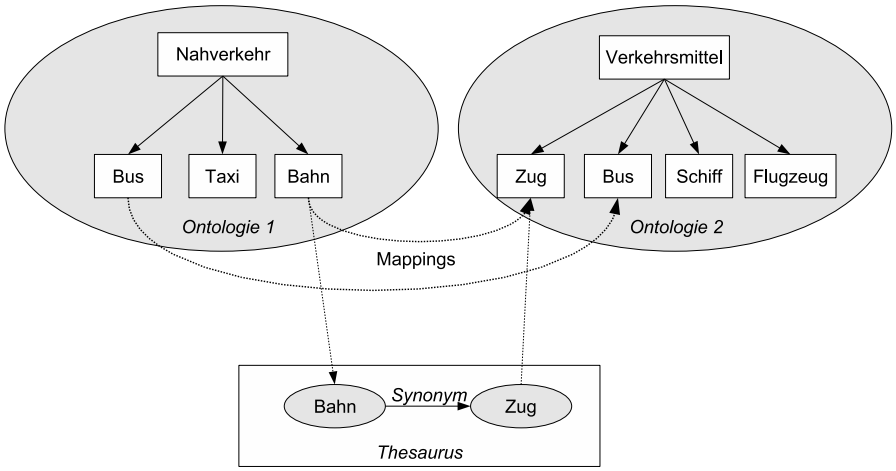



**Abbildung 3.8.:** Externes Matching mit Hilfe eines Thesaurus

Beispiele für fachspezifische Thesauri sind MeSH (United States National Library of Medicine, 2007), ein Thesaurus für medizinische Fachbegriffe, und der Getty Thesaurus of Geographic Names, der mehr als eine Million geographische Bezeichnungen enthält (Getty Research Institute, 2006). Mit Hilfe dieses Thesaurus lässt sich z.B. ein Mapping zwischen `Finnland` und `Suomi` finden, das mit einem internen Verfahren nicht gefunden werden kann.

Giunchiglia und Shvaiko (2003, S. 10ff.) unterscheiden die linguistischen Verfahren zwischen sinn- und glossenbasierten Verfahren: sinnbasierte Verfahren nutzen direkte Verbindungen in Thesauri, wie in Abbildung 3.8 dargestellt[4]. Glossenbasierte Verfahren verwenden dagegen die Erklärungen einzelner Begriffe und messen deren Ähnlichkeit, indem z.B. der Anteil gemeinsamer Wörter in zwei Erklärungen verglichen wird.

*Upper-Level-Ontologien:* Shvaiko und Euzenat beschreiben die Nutzung von Upper-Level-Ontologien als mögliche Technik des Ontology Matching. Upper-Level-Ontologien sind Ontologien wie SUMO und OpenCyc, die Sammlungen von Allgemeinwissen enthalten (siehe Kapitel 2.2.1). Auch die oben genannten Thesauri können als Upper-Level-Ontologien im weitesten Sinne aufgefasst werden. Sie werden meist als Ontologien bezeichnet, auch wenn es die Meinung gibt, dass es sich dabei um lexikalische Datenbanken, aber nicht um Ontologien handele (vgl. Lüngen und Storrer, 2006, S. 4). Systeme, die neben Thesauri noch andere Upper-Level-Ontologien nutzen, existieren derzeit jedoch nicht (Shvaiko und Euzenat, 2005, S. 158f.).

*Mapping-Wiederverwendung:* Wenn man ein Matching von zwei Ontologien $O_1$ und $O_2$ durchführt und bereits Mappings zwischen $O_1$ und $O_3$ einerseits und $O_2$ und $O_3$ andererseits vorliegen, so kann es hilfreich sein, die gefundenen Mappings wiederzuverwenden (Shvai-

[4]Diese Abbildung demonstriert noch einmal den Unterschied zwischen internem und externem Matching: die Äquivalenz der beiden `Bus` benannten Elemente wird von einem internen stringbasierten Matcher gefunden; um die Äquivalenz von `Zug` und `Bahn` zu finden, wird dagegen ein Thesaurus benötigt.

ko und Euzenat, 2005, S. 158). Diese Mappings können aus vorangegangenen Matching-Prozessen stammen, oder sie existieren, weil beide Ontologien Begriffe einer gemeinsamen Top-Level-Ontologie verwenden (Noy, 2004, S. 65). Abbildung 3.9 verdeutlicht den Prozess: das Mapping-Element von `Bahn` in Ontologie 1 zu `Bahn` in Ontologie 2 wird über die beiden Beziehungen zum dem Element `Zug` in Ontologie 3 gefunden.

Dieser Ansatz ist für das Matching von Ontologien aus der gleichen Domäne sinnvoll, kann jedoch beim Matching von Ontologien verschiedener Domänen zu Schwierigkeiten führen (Rahm und Bernstein, 2001, S. 342).

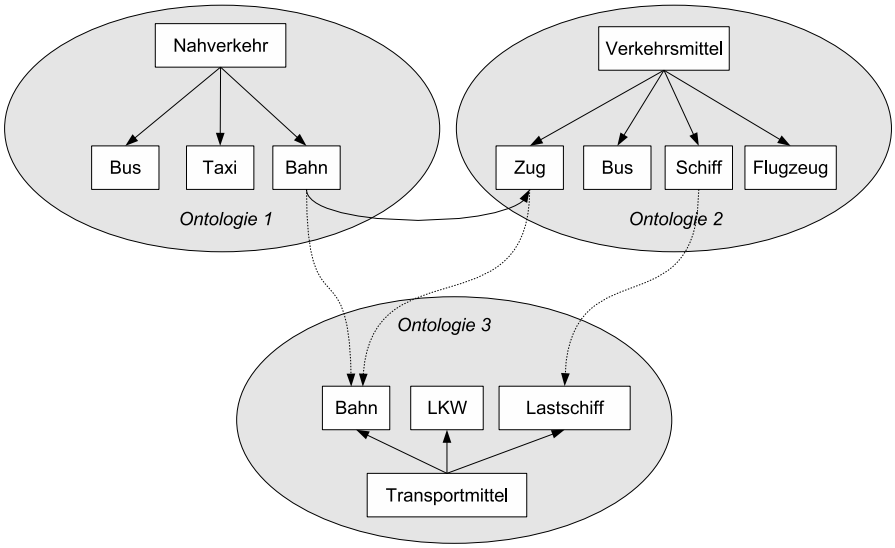



**Abbildung 3.9.:** Matching durch Wiederverwenden von Mappings

### 3.3.2. Strukturbasierte Verfahren

Im Gegensatz zu elementbasierten Verfahren arbeiten strukturbasierte Verfahren nicht mit einzelnen Elementen der Ontologien, sondern mit Gruppen von Elementen, die durch Beziehungen verbunden sind. Auch hier lassen sich interne und externe Verfahren unterscheiden. Interne strukturbasierte Verfahren können nicht allein arbeiten, sondern benötigen eine Menge von vorliegenden Mapping-Elementen, sogenannten Anker-Paaren (vgl. Noy und Musen, 2001, S. 63), die manuell eingegeben oder durch die oben beschriebenen Verfahren gewonnen werden – letztere Methode wird auch als *Boostrapping* bezeichnet (Ehrig und Staab, 2004a, S. 686). Diese Anker-Paare können zum einen auf ihre Konsistenz geprüft, zum anderen genutzt werden, um weitere Mappings abzuleiten (Euzenat und Shvaiko, 2007, S. 115). Zu den internen strukturbasierten Verfahren zählen:

*Graphbasierte Verfahren:* Bei graphbasierten Verfahren werden Ontologien als Graphen betrachtet (vgl. Kapitel 2.3.1). Die grundlegende Idee graphbasierter Verfahren ist, dass, wenn eine Ähnlichkeit zwischen zwei Elementen in verschiedenen Ontologien besteht, dann auch eine Ähnlichkeit zwischen deren Nachbarn bestehen kann (Shvaiko und Euzenat, 2005, S. 159, und Noy, 2004, S. 67). Dazu wird z.B. der Anteil gemeinsamer benachbarter Konzepte ermittelt (vgl. Maedche und Staab, 2002, S. 255ff). Man unterscheidet

zwischen strukturbasierten Verfahren, die nur *Local Context*, d.h., unmittelbare Nachbarknoten verwenden, und solchen, die auch *Non-Local Context*, d.h. weiter entfernte Knoten, mit einbeziehen (Noy und Musen, 2001, S. 63). Im Beispiel in Abbildung 3.8 könnte man so schließen, dass die Konzepte `Nahverkehr` und `Verkehrsmittel` zu einem gewissen Grad gleich sind, da beide über zwei gemeinsame Nachfolgerknoten (`Bus` sowie `Zug`/`Bahn`) verfügen.

*Taxonomiebasierte Verfahren:* Einen Spezialfall von graphbasierten Verfahren stellen taxonomiebasierte Verfahren dar. Hier werden nur Taxonomierelationen in den Graph aufgenommen, also Superklasse-Subklasse-Beziehungen. Dadurch entsteht eine hierarchische Struktur, die bestimmten Verfahren erlaubt, auf Mappings zu schließen. So kann man etwa Entsprechungen von Elementen auf den Pfaden zwischen Anker-Paaren annehmen (vgl. Noy und Musen, 2001, S. 70ff). Abbildung 3.10 illustriert das Prinzip: Aufgrund der beiden Anker-Mappings zwischen `Verkehrsmittel` und `Verkehrsmittel` sowie zwischen `Straßenbahn` und `Tram` kann darauf geschlossen werden, dass auch zwischen `Bahn` und `Schienenverkehr` eine Beziehung besteht.

Einen ähnlichen Spezialfall stellen sogenannte *mereologiebasierte Verfahren* dar. Der Begriff *Mereologie* bezeichnet Teil-Ganzes-Beziehungen. Analog zu taxonomiebasierten Verfahren wird bei mereologiebasierten Verfahren anhand von Anker-Mappings geschlossen, dass, wenn zwei Klassen ähnliche Teile haben, auch die Klassen als Ganzes ähnlich sind (Euzenat und Shvaiko, 2007, S. 103).

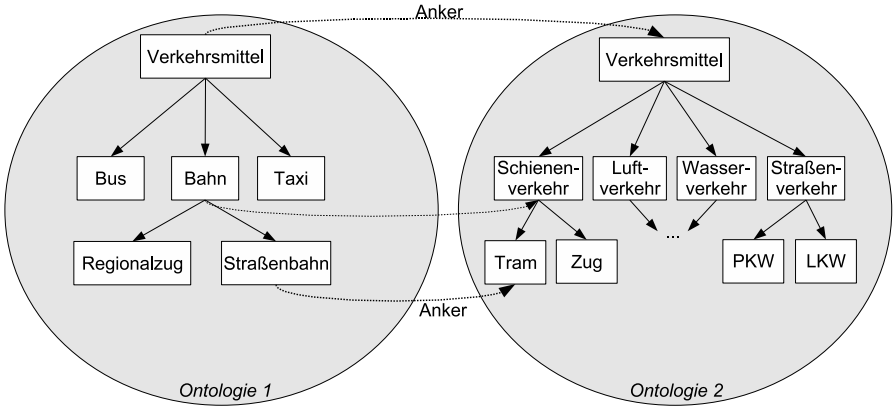



**Abbildung 3.10.:** Taxonomiebasiertes Matching

Externe strukturbasierte Verfahren schließen weitere Informationsquellen ein. Zu externen strukturbasierten Verfahren zählen:

*Struktur-Repositories:* Struktur-Repositories sind Sammlungen von Teilontologien. Mit diesen können Teilstrukturen der Ontologien, die ein Matcher bearbeitet, verglichen werden, um einen Näherungswert für deren Ähnlichkeit festzustellen. Dies entspricht dem oben dargestellten Ansatz der Mapping-Wiederverwendung, mit dem Unterschied, dass keine einzelnen Elemente, sondern Teilstrukturen einer Ontologie verglichen werden. Dabei werden nicht unmittelbar Mappings gefunden, die gewonnene Information kann jedoch

genutzt werden, um zu entscheiden, ob sich eine detailliertere Untersuchung der Teilstrukturen auf Mappings lohnt (Shvaiko und Euzenat, 2005, S. 160).

*Modellbasierte Verfahren:* Shvaiko und Euzenat nennen modellbasierte Verfahren als weitere mögliche Ausprägung externer strukturbasierter Verfahren. Bei diesen Verfahren werden die Ontologien und die bereits bekannten Mappings in einer logischen Beschreibungssprache (etwa in propositionaler Logik) beschrieben und dann von einem Beweiser bearbeitet, um weitere Entsprechungen zu finden (vgl. auch Meilicke u. a., 2006, S. 61 ff.).

Die in diesem Kapitel dargestellten Ansätze werden in verschiedenen Werkzeugen implementiert, wobei meist mehrere Verfahren in einem Werkzeug implementiert sind. Einige Beispiele für diese Werkzeuge werden im nächsten Kapitel dargestellt.

## 3.4. Werkzeuge

Ontology-Matching-Werkzeuge lassen sich grundsätzlich in automatisierte und semi-automatisierte Werkzeuge unterscheiden (siehe z.B. Kalfoglou und Schorlemmer, 2005, S. 28). Automatisierte Werkzeuge wenden einen Match-Operator auf ein Paar von Ontologien an und erzeugen damit ein Mapping. Bei semi-automatisierten Werkzeugen muss ein menschlicher Nutzer dagegen in den Prozess der Mapping-Generation eingreifen und einzelne Mappings bestätigen oder ablehnen, woraufhin in der nächsten Iteration weitere Mappings generiert werden.

Semi-automatisierte Werkzeuge erzielen in der Regel Ergebnisse besserer Qualität, da menschliches Fachwissen einbezogen werden kann, sind aber insbesondere bei großen Ontologien nicht praktikabel, da hier die Zahl der benötigten menschlichen Eingriffe zu hoch wäre. Auch in bestimmten Anwendungsszenarien wie im Falle autonomer Agenten, in denen kein menschliches Eingreifen vorgesehen ist, sind semi-automatisierte Werkzeuge nicht sinnvoll einsetzbar (Kalfoglou und Schorlemmer, 2005, S. 28, Noy, 2004, S. 66). Allerdings lassen sich auch aus automatisierten Werkzeugen leicht semi-automatisierte konstruieren, indem mehrere Iterationen des automatisierten Verfahrens durchlaufen werden, zwischen denen ein Nutzer das gefundene Mapping kontrollieren und ggf. korrigieren kann. Die korrigierten Mapping-Elemente dienen dann als Eingabe für die nächste Iteration (Euzenat u. a., 2004, S. 60).

In dieser Arbeit werden fast ausschließlich automatisierte Werkzeuge betrachtet. Da es sehr viele Matching-Werkzeuge gibt (siehe z.B. Kalfoglou u. a., 2005, S. 12 ff. und Euzenat und Shvaiko, 2007, S. 186 ff. für eine umfassende, wenn auch wahrscheinlich nicht vollständige Auflistung), waren für diese Arbeit weitere Einschränkungen nötig.

In der Auswahl sind nur Werkzeuge enthalten, die zum einen als Testversionen im Internet beziehbar sind, und die zum anderen Ontologien im RDF-S- und OWL-Format als Eingabe verarbeiten können, da es sich dabei um das mit Abstand am weitesten verbreitete Ontologieformat handelt (Cardoso, 2007, S. 85). Von diesen wurden nur die derzeit am weitesten verbreiteten Werkzeuge FOAM, INRIA, COMA++ und CROSI betrachtet.

Das Werkzeug PROMPT ist nicht voll automatisiert, aber eines der ältesten und bekanntesten Werkzeuge (Ehrig und Staab, 2004b, S. 11) und wurde daher in die Auswahl mit aufgenommen. Darüber hinaus wurde das Werkzeug Falcon betrachtet, das zwar relativ neu und daher nicht so weit verbreitet ist wie die eben genannten, aber vielversprechende Ansätze zum

Erreichen von Skalierbarkeit enthält. Alle getesteten Werkzeuge sind, zumindest als Testversion, im Internet frei verfügbar.

### 3.4.1. PROMPT und Anchor-PROMPT

Das Werkzeug PROMPT[5] wurde und wird am Institut für medizinische Informatik in Stanford entwickelt. Es dient zum Matching und Merging von Ontologien, wobei der Fokus auf dem Merging von Ontologien liegt. Abbildung 3.11 zeigt einen Screenshot der Anwendung.

Der PROMPT-Algorithmus sieht vor, dass dem Nutzer Vorschläge präsentiert werden, die er annehmen oder ablehnen und durch eigene Vorschläge ersetzen kann. Diese Vorschläge werden dann durch das Werkzeug umgesetzt und führen zu neuen Vorschlägen. Entstehen durch einen Vorschlag Konflikte, so muss der Nutzer ebenfalls entscheiden, wie diese zu lösen sind. Der Prozess endet, wenn ein komplettes Mapping gefunden bzw. die Ontologien komplett zusammengeführt sind. Für die initiale Menge von Vorschlägen kann ein beliebiger elementbasierter Vergleichsoperator angewendet werden (Noy und Musen, 2000, S. 452).

Die neuen Vorschläge werden nach bestimmten Regeln aus den Änderungsvorgaben abgeleitet. Werden z.B. zwei Klassen aufeinander abgebildet, so wird versucht, deren Superklassen und Eigenschaften ebenfalls aufeinander abzubilden, sofern ein elementbasierter Vergleichsoperator eine gewisse Ähnlichkeit feststellt (Noy und Musen, 2000, S. 452).

PROMPT ist als Plug-In für Protégé implementiert (siehe Kapitel 2.5.2.4). Der Nutzer wird bei seinen Entscheidungen unterstützt, indem das Werkzeug begründet, warum die Änderungen vorgeschlagen wurden, und versucht, bei großen Ontologien stets Änderungen an benachbarten Elementen nacheinander vorzuschlagen, um die Fokusverschiebungen minimal zu halten (Noy und Musen, 2000, S. 453).

Anchor-PROMPT ist ein strukturbasierter Matching-Algorithmus. Eine Menge von initialen Mapping-Elementen (sogenannten Anker-Paare) wird benutzt, um weitere mögliche Mapping-Elemente zu finden. Anchor-PROMPT betrachtet dazu die Elemente, die sich in den beiden Ontologien auf Pfaden zwischen Ankern befinden, und nimmt eine Ähnlichkeit zwischen diesen Elementen an, die umso höher ausfällt, je größer der Pfad zwischen den Ankern ist, auf dem sich diese Elemente befinden (siehe Abbildung 3.10 auf Seite 56). Anchor-PROMPT kann auch als eigenständiges Matching-Werkzeug genutzt werden, wobei die Anker-Paare manuell gesetzt oder mit einem beliebigen elementbasierten Matcher gewonnen werden (Noy und Musen, 2001, S. 63 ff., und Noy und Musen, 2003, S. 1011 ff.).

PROMPT und Anchor-PROMPT sind Teile der sogenannten PROMPT-Suite, einer Sammlung von Werkzeugen, die zum Arbeiten mit verschiedenen Ontologien entwickelt wurden. Neben den genannten umfasst die Sammlung die Werkzeuge PROMPTDiff, das die Verwaltung verschiedener Versionen von Ontologien ermöglicht, und PROMPTFactor, das die Erstellung von Teilontologien zu bestimmten Themen ermöglicht (Noy und Musen, 2003, S. 1026 ff.).

[5] PROMPT wird in den Publikationen der Entwickler meist in Großbuchstaben geschrieben, obwohl sich dort keine Auflösung des Akronyms findet.

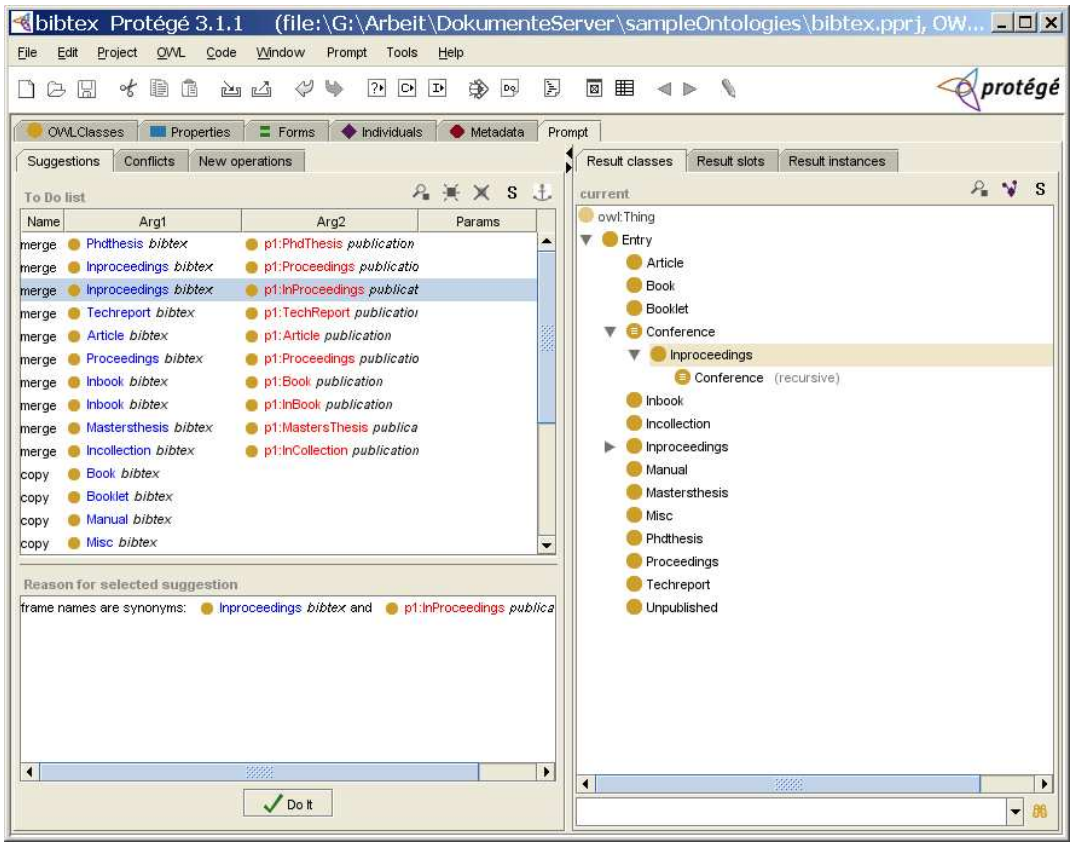



**Abbildung 3.11.:** Screenshot des Matching-Werkzeugs PROMPT

### 3.4.2. FOAM

Das Werkzeug FOAM (**F**ramework for **O**ntology **A**lignment and **M**apping) wurde am Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) der Universität Karlsruhe entwickelt. Es kann als Kommandozeilenwerkzeug, über ein Java-API und über eine webbasierte Nutzerschnittstelle verwendet werden (Ehrig, 2007b, S. 146f.). FOAM kann mit Ontologien in OWL Lite und OWL DL umgehen, mit Ontologien in OWL Full dagegen nicht. Die Ontologien werden dabei mit dem Framework KAON2 (siehe Kapitel 2.5.2.3) verarbeitet. In der Literatur werden die in FOAM implementierten Algorithmen auch unter den Namen *NOM* (Naïve Ontology Mapping) (Ehrig und Staab, 2004a, S. 689) und *QOM* (Quick Ontology Mapping) (Ehrig und Staab, 2004a, S. 690) geführt.

FOAM produziert einfache Mappings nach Definition 3.1, wobei nur Gleichheitsrelationen betrachtet werden (Ehrig, 2007b, S. 19). Weiterhin ist ein Mapping bei FOAM stets bijektiv, d.h., jedes Element einer Ontologie wird auf maximal ein anderes Element abgebildet (Ehrig und Staab, 2004a, S. 689).

Der Matching-Prozess bei FOAM läuft in fünf Phasen ab, die in Iterationen wiederholt werden können, wie in Abbildung 3.12 dargestellt ist (vgl. Ehrig, 2007b, S. 62 ff.):

1. *Elementerstellung* bezeichnet das Abbilden der Eingabeontologien $O_1$ und $O_2$ auf eine Menge von Elementen, die zum Matching herangezogen werden. Dies können in einfachen Fällen die Bezeichner von Klassen, Eigenschaften etc. in den Ontologien sein, oder es

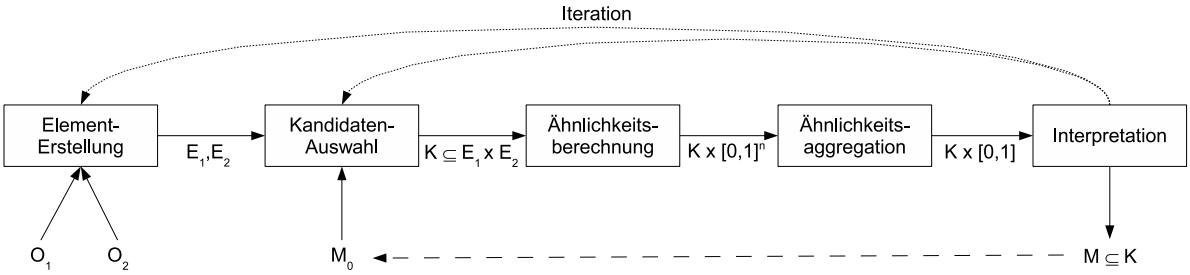



**Abbildung 3.12.:** Matching-Prozess bei FOAM (in Erweiterung von Ehrig, 2007b, S. 62)

werden nur Untermengen von diesen Bezeichnern verwendet. Elemente können aber auch mehr Informationen, z.B. über Super- und Subklassen und Instanzen, mit einbeziehen. Das Ergebnis sind zwei Mengen $E_1$ und $E_2$. Diese Phase wird üblicherweise nur einmal durchgeführt und bei weiteren Iterationen übersprungen.

2. *Kandidatenauswahl* ist die Auswahl von Paaren von Elementen, die verglichen werden sollen. Im einfachsten (und ineffizientesten) Fall werden alle Elemente paarweise verglichen. In FOAM wird das Eingabemapping $M_0$ (oder ein Mapping $M$ aus einer vorangegangenen Iteration) verwendet, um geeignete Kandidatenpaare auszuwählen. Implementierte Strategien sind u.a. der Vergleich von Elementen, die ähnlich benannt sind wie die, für die bereits Mappings gefunden wurden, oder von Elementen, die in der Nachbarschaft von den Elementen liegen, für die bereits Mappings existieren (Ehrig und Staab, 2004a, S. 691). Das Ergebnis ist eine Menge von Kandidaten $K$.

3. *Ähnlichkeitsberechnung* bestimmt für die im vorangegangenen Schritt ermittelten Kandidaten einen Ähnlichkeitswert. Dabei können verschiedene Techniken, wie in Kapitel 3.3 beschrieben, parallel genutzt werden. In FOAM werden hier mehrere verschiedene Techniken eingesetzt, wie String-Vergleich der Bezeichner, Nutzung von WordNet, Vergleich benachbarter Einheiten in der Taxonomie, Vergleich von Einheiten, zu denen eine Relation besteht, usw. (Ehrig, 2007b, S. 28 ff., S. 68 f., u. S. 148). Für jede dieser Techniken wird ein Ähnlichkeitswert ermittelt, der üblicherweise im Intervall $[0, 1]$ liegt.

4. *Ähnlichkeitsaggregation* bildet die verschiedenen Ähnlichkeitswerte, die im vorangegangenen Schritt ermittelt wurden, auf einen einzigen Wert ab. Dabei werden die einzelnen Ähnlichkeitswerte gewöhnlich in einer gewichteten Summe zusammengefasst. FOAM verwendet dafür Gewichte, die aus einer Trainingsmenge von Ontologiepaaren und Referenz-Mappings gelernt wurden (Ehrig und Staab, 2004a, S. 689). Außerdem können die einzelnen Ähnlichkeitswerte noch durch eine Funktion vorverarbeitet werden. Bei FOAM wird z.B. eine sogenannte *Sigmoid-Funktion* auf die Ähnlichkeitswerte angewendet, die starke Ähnlichkeiten weiter verstärkt und den Einfluss schwacher Ähnlichkeiten reduziert (Ehrig, 2007b, S. 71f.).

5. *Interpretation* bezeichnet die Auswahl eines Mappings aus den mit aggregierten Ähnlichkeitswerten versehenen Paaren von Entitäten. Bei semi-automatisierten Verfahren wird diese Auswahl in der Regel von den Nutzern getroffen. Für automatisierte Verfahren ist

der einfachste Weg die Nutzung einer unteren Schranke für den Ähnlichkeitswert, wobei alle Paare, die einen Ähnlichkeitswert oberhalb dieser Schranke haben, das Mapping bilden. Darüber hinaus forciert FOAM ein bijektives Mapping und verwirft daher, wenn eine Entität in mehreren Paaren enthalten ist, alle Paare bis auf das, das den höchsten Ähnlichkeitswert aufweist (Ehrig, 2007b, S. 72f.). Außerdem kann FOAM auch als semi-automatisiertes Werkzeug betrieben werden und präsentiert in diesen Fällen einige kritische Paare von Entitäten (diejenigen, bei denen die Ähnlichkeit zweifelhaft ist, deren Ähnlichkeit also nahe bei der unteren Schranke liegt) dem Nutzer zur manuellen Evaluation (Ehrig und Sure, 2005, S. 72). Das Ergebnis ist die Menge von Mappings $M$, die ausgegeben oder in einer weiteren Iteration verwendet wird.

Die maximale Anzahl der Iterationen kann bei FOAM in einer Konfigurationseinstellung festgesetzt werden (Ehrig, 2007b, S. 223). Obwohl Konvergenz zwar in den meisten Fällen eintritt, dies aber nicht notwendig der Fall sein muss, ist es nicht sinnvoll, diese als Abbruchkriterium zu wählen (Ehrig, 2007b, S. 75).

Die meisten Matching-Werkzeuge verwenden ähnliche Phasen wie FOAM, wobei einzelne Phasen wegfallen können. So ist beispielsweise der Schritt der Ähnlichkeitsaggregation nicht notwendig, wenn nur ein Ähnlichkeitsmaß verwendet wird (Ehrig und Staab, 2004a, S. 689).

### 3.4.3. INRIA (OWL Lite Alignment)

OLA (**O**ntology **L**ite **A**lignment) ist eine Entwicklung der Universität von Montréal und des Institut National de Recherche en Informatique et en Automatique (INRIA) Rhône-Alpes. Der zugrunde liegende Matching-Mechanismus wird meist mit *INRIA* bezeichnet (Euzenat u. a., 2005, S. 99).

Mit INRIA lassen sich Ontologien in OWL Lite sowie, eingeschränkt, in OWL DL aufeinander beziehen; Ontologien in OWL Full werden dagegen nicht unterstützt. Es werden stets nur einfache, homogene Mappings generiert (Euzenat u. a., 2004, S. 59 ff.).

Zunächst werden die Ontologien mit OWL API (siehe Kapitel 2.5.2.1) in einen gerichteten Graphen überführt. Auf diesem wird dann das eigentliche Mapping durchgeführt (Euzenat u. a., 2004, S. 60 ff.). Die einzelnen Elemente werden mit Hilfe von drei vordefinierten elementbasierten Matching-Techniken verglichen: Test auf Namensgleichheit mit Hilfe von WordNet, Edit-Distanz und Test auf enthaltene Substrings. Außerdem können diese Tets mit der strukturbasierten Technik, Eigenschaften von Klassen zu vergleichen, zu einem zusammengesetzten Match-Operator kombiniert werden. Das Java-API ermöglicht das Hinzufügen weiterer Techniken (Euzenat, 2004, S. 706).

Anhand des gerichteten Graphen werden Ähnlichkeiten zwischen benachbarten Elementen berücksichtigt – zwei Elemente, die ähnliche benachbarte Elemente besitzen, werden insgesamt als ähnlicher betrachtet als Elemente, die keine ähnlichen Nachbarn besitzen. Aus diesen Bedingungen wird dann ein Gleichungssystem aufgestellt, dessen Lösung die paarweisen Ähnlichkeitswerte zwischen den Elementen der Ontologien sind.

Aus diesen Informationen – den elementbasierten Vergleichen und der Berücksichtigung der Ähnlichkeiten benachbarter Elemente – wird ein gewichteter Ähnlichkeitswert für jedes Paar von Elementen berechnet. Die optimalen Gewichte können je nach Art der Eingabeontologien

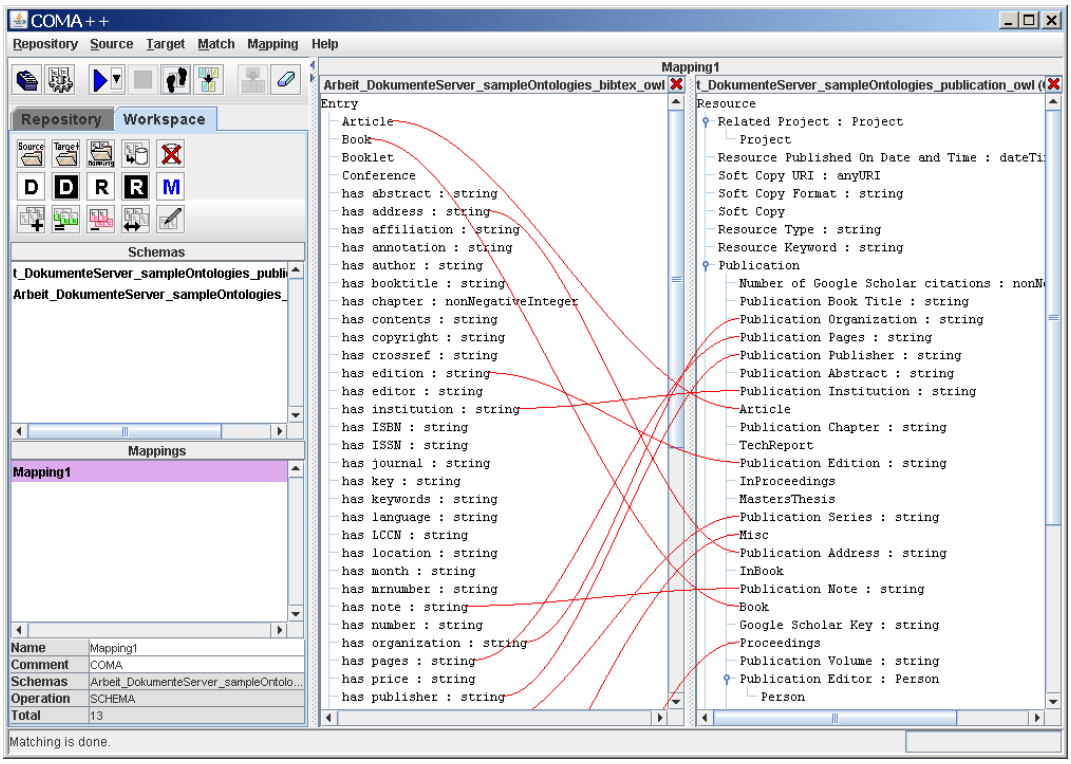



**Abbildung 3.13.:** Screenshot des Matching-Werkzeugs COMA++

variieren. Es ist auch möglich, aus einer Menge von Ontologiepaaren und Referenzmappings optimale Gewichte zu lernen (Euzenat u. a., 2004, S. 63 ff.).

Die Ausgabe kann mit Hilfe verschiedener Renderer in verschiedenen Formaten erfolgen. Unterstützt werden RDF, OWL, C-OWL und SWRL (siehe Kapitel 3.1.4). Außerdem ist es möglich, ein XSLT-Stylesheet zu generieren, mit dem Daten, die in einer Ontologie ausgedrückt sind, in die Begriffe der anderen Ontologie überführt werden können (Euzenat, 2004, S. 707). Dieses kann zum Query- und Answer-Rewriting (siehe Kapitel 3.2.1) verwendet werden.

### 3.4.4. COMA++

COMA++ (**Co**mbining **Ma**tch Algorithms) ist ein Werkzeug, das am Fachbereich Informatik der Universität Leipzig entwickelt wird. Es kann XML-Schema-Dateien und Ontologien im OWL-Format als Eingabedateien verarbeiten, die intern in ein eigenes Format, das auf gerichteten, azyklischen Graphen basiert, umgewandelt werden. Ursprünglich wurde COMA für das Matching von XML- und Datenbankschemas entwickelt (Do und Rahm, 2002, S. 610), die Erweiterung COMA++ unterstützt auch das Matching von Ontologien. Darüber hinaus können auch Datenbank-Schemas über eine ODBC-Schnittstelle (Geiger, 1995) importiert werden. Die Bedienung erfolgt über eine eigene Nutzerschnittstelle, die in Abbildung 3.13 dargestellt ist (Aumueller u. a., 2005, S. 906).

COMA++ verwendet ein Repository, in dem die importierten Schemas und Ontologien in einer Datenbank gespeichert werden. Außerdem werden dort Mappings, externe Informationen wie Synonymlisten und Lexika sowie Meta- und Konfigurationsdaten der einzelnen Matcher

gespeichert (Do und Rahm, 2007, S. 858 f.). Die gespeicherten Mappings zwischen Fragmenten werden genutzt, um das Matching ähnlicher Fragmente zu erleichtern (Rahm u. a., 2004, S. 30).

Bei COMA++ liegt ein besonderer Fokus auf dem Matching großer Schemas und Ontologien. Dazu werden diese zunächst in Fragmente zerlegt und die Ähnlichkeit der Fragmente bestimmt, ein detailliertes Matching wird nur auf ähnlichen Fragmenten durchgeführt (Rahm u. a., 2004, S. 30).

Das Ergebnis von COMA++ sind Mappings nach Definition 3.2, wobei nur die Gleichheitsrelation berücksichtigt wird (Do und Rahm, 2007, S. 857 ff.). Auch heterogene Mappings, die z.B. Klassen und Properties aufeinander abbilden, sind möglich (Massmann u. a., 2006, S. 108).

Das Matching der Fragmentpaare erfolgt in ähnlichen Schritten wie bei FOAM, wobei verschiedene element- und strukturbasierte Match-Operatoren und verschiedene Aggregationsstrategien zum Einsatz kommen können. Auch ein instanzbasierter Matcher ist in COMA++ enthalten (Massmann u. a., 2006, S. 108). Abbildung 3.14 zeigt den Matching-Prozess bei COMA++:

1. Zunächst werden die Eingabeontologien in Fragmente zerlegt.
2. Die Fragmente werden paarweise auf Ähnlichkeit untersucht und diejenigen Paare herausgesucht, die eine Mindestähnlichkeit aufweisen.
3. Diese Paare werden dann nacheinander nach demselben Mechanismus verarbeitet: Zunächst werden Elemente aus beiden Fragmenten erstellt, zwischen denen eine Ähnlichkeit bestehen kann. Diese werden dann paarweise mit verschiedenen Verfahren verglichen und die Ähnlichkeitswerte aggregiert. Alle Paare von Elementen, die einen bestimmten Ähnlichkeitswert übersteigen, werden als Teilergebnis zurückgegeben.
4. Das Gesamtergebnis wird als Vereinigungsmenge aller Teilergebnisse zurückgeliefert.

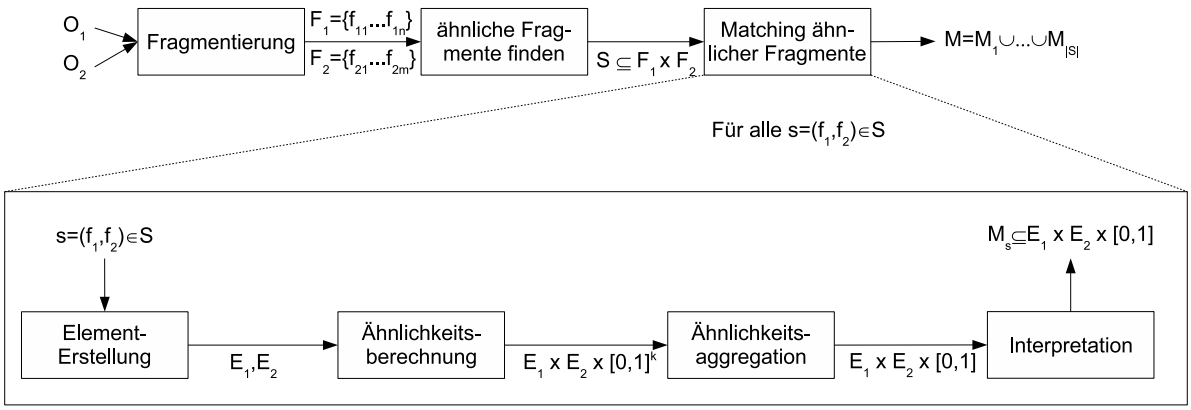



**Abbildung 3.14.:** Matching-Prozess bei COMA++ (in Anlehnung an Do und Rahm, 2007, S. 872)

### 3.4.5. CROSI Mapping System

Das CROSI (**C**apturing, **R**epresenting, and **O**perationalizing **S**emantic **I**nteroperability) **M**apping **S**ystem (kurz CMS) ist ein Werkzeug, das von der University of Southampton und

den Hewlett Packard Laboratories entwickelt wurde. Es kann über eine Kommandozeilenschnittstelle, ein JAVA-API und eine webbasierte Nutzerschnittstelle angesprochen werden. CMS basiert auf demselben Phasen-Modell wie FOAM (Kalfoglou u. a., 2005, S. 21 ff.).

CMS kann Ontologien im OWL- und RDFS-Format verarbeiten, die mit Hilfe von JENA (siehe Kapitel 2.5.2.2) eingelesen werden. Die Ausgabe kann u.a. im OWL-Format erfolgen.

Das Kernstück von CMS ist eine Sammlung von verschiedenen individuellen Match-Operatoren, die, mit Gewichten versehen, kombiniert werden – dabei kommen u.a. einfache String-Vergleiche, Synonymprüfung mit WordNet und strukturbasierte Vergleiche zum Einsatz. Auch externe Matching-Systeme können als Operatoren eingesetzt werden; in der Standard-Version von CMS sind INRIA und FOAM enthalten (Kalfoglou u. a., 2005, S. 23 f.). Damit kann CMS auch als Meta-Matching-System verwendet werden, das die Ergebnisse mehrerer Matching-Systeme kombiniert. Die Gewichte, mit denen die einzelnen Operatoren aggregiert werden, müssen vom Nutzer manuell gesetzt werden, ebenso wie die untere Schranke, die zur Produktion des Ergebnisses im letzten Schritt genutzt wird (Kalfoglou u. a., 2005, S. 37).

CMS produziert Mappings nach Definition 3.1, wobei nur die Gleichheitsrelation verwendet wird (Kalfoglou und Hu, 2005, S. 1). Werden externe Matching-Systeme eingesetzt, die Mappings mit mehr Informationen produzieren, z.B. andere Relationstypen, so muss diese Mehrinformation unter Umständen verworfen werden, um die Ergebnisse aggregieren zu können (Kalfoglou u. a., 2005, S. 37).

### 3.4.6. Falcon-AO

Das Werkzeug Falcon-AO (Technologies for **f**inding, **a**ligning and **l**earning ontologies, and **c**apturing knowledge by an **on**tology-driven application; tool for **a**ligning **o**ntologies) wird an der School of Computer Science and Engineering der Southeast University in Nanjing, China, entwickelt. Es verarbeitet Ontologien in OWL Lite und OWL DL (Hu u. a., 2006a, S. 124).

Ein zentraler Bestandteil von Falcon-AO ist die Partitionierung von Ontologien. Zunächst werden die Ontologien $O_1$ und $O_2$ in kleinere Subontologien $O_{11} \dots O_{1n}$ und $O_{21} \dots O_{2m}$ partitioniert. Diese werden dann paarweise mit Hilfe von drei Match-Operatoren verarbeitet (Hu u. a., 2006b, S. 72 ff.). Im einzelnen kommen ein string- und ein strukturbasierter Match-Operator zum Einsatz, sowie ein Vergleich sogenannter virtueller Dokumente, die aus gewichteten Mengen von Begriffen bestehen, die in der Nähe einer Entität in einer Ontologie stehen (Hu und Qu, 2006, S. 302).

Für die Aggregation der drei Ähnlichkeitsmaße der Kandidaten ist in Falcon-AO ein dynamischer Ansatz implementiert, der zunächst die linguistische und die strukturelle Vergleichbarkeit zwischen beiden Subontologien misst: die linguistische Vergleichbarkeit ist das Verhältnis von Kandidaten und Entitäten in den Subontologien; die strukturelle Vergleichbarkeit untersucht, wie viele gemeinsame Sprachkonstrukte in beiden Subontologien verwendet werden. Je nachdem, wie diese beiden Werte ausfallen, werden die Gewichte für die Aggregation gesetzt (Hu u. a., 2006a, S. 126). Abbildung 3.15 zeigt den Gesamtprozess.

Die Vorverarbeitung durch Partitionierung verfolgt zwei Zwecke: einerseits kann durch das anschließende Matching festgestellt werden, welche größeren Teile der Ontologien zueinander in Beziehung stehen. Letzteres wird von den Autoren als *Block-Matching* bezeichnet: Mit dieser Methode lassen sich Teilontologien anstelle einzelner Entitäten aufeinander abbilden (Hu und

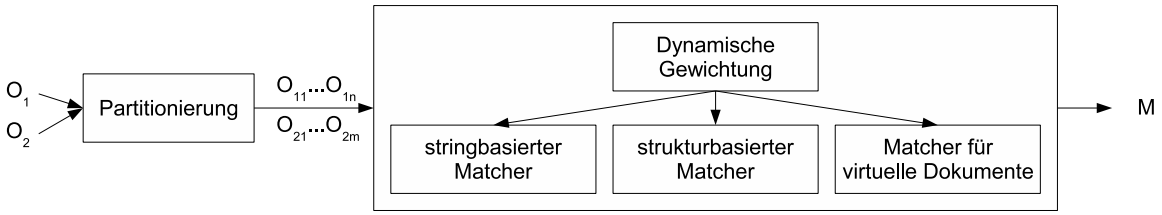



**Abbildung 3.15.:** Matching-Prozess bei Falcon (nach Hu u. a., 2006a, S. 125)

Qu, 2006, S. 300). Andererseits ist das Partionieren auch ein möglicher Ansatz, mit größeren Ontologien umzugehen. Werden diese zunächst partioniert, muss der eigentliche Matching-Algorithmus nur auf kleineren Ontologien arbeiten (Hu u. a., 2006b, S. 74).

## 3.5. Qualitätsmaße für Matching-Werkzeuge

Um die Qualität der Ergebnisse von Matching-Werkzeugen zu messen, werden in der Regel Paare von Ontologien verwendet, die zuvor von menschlichen Experten in Beziehung gesetzt wurden. Die Arbeit der Experten dient als sogenanntes *Referenz-Mapping*, das dann mit der Ausgabe des Matching-Werkzeugs verglichen wird. Für diesen Vergleich werden üblicherweise drei Maße verwendet: *Precision*, *Recall* und *F-Measure* (vgl. Do u. a., 2002, S. 224 ff.), die aus dem *Information Retrieval* stammen (Baeza-Yates und Ribeiro-Neto, 1999, S. 73 ff.).

Vier Grundbegriffe, die bei der Definition dieser Maße eine Rolle spielen, sind *true positives*, *true negatives*, *false positives* und *false negatives*. Diese sind in Abbildung 3.16 dargestellt. $O_1 \times O_2 \times T$ bezeichnet die Menge aller möglichen Abbildungen, das Kreuzprodukt aller Elemente beider Ontologien $O_1$ und $O_2$ sowie aller möglichen Beziehungstypen $T$ (hier werden nur einfache Mappings berücksichtigt). $M$ ist die Menge der von einem Werkzeug gefundenen Mappings, und $R$ ist das Referenz-Mapping.

*True positives:* Als *true positives* werden die Mapping-Elemente bezeichnet, die von einem Matching-Werkzeug gefunden wurden und auch korrekt sind. Die Menge der true positives ist definiert als $M \cap R$.

*True negatives:* Als *true negatives* werden die Mapping-Elemente bezeichnet, die weder korrekt sind noch von einem Matching-Werkzeug gefunden wurden. Die Menge der true negatives ist definiert als $(O_1 \times O_2 \times T) \setminus (M \cup R)$. Diese Kennzahl wird aber in der Analyse von Ontology Mappings eher selten verwendet.

*False positives:* Als *false positives* werden die Mapping-Elemente bezeichnet, die fälschlicherweise von einem Matching-Werkzeug gefunden werden, obwohl sie nicht korrekt sind. Die Menge der False positives ist definiert als $M \setminus R$.

*False negatives:* Als *false negatives* werden die Mapping-Elemente bezeichnet, die zwar korrekt sind, aber von einem Matching-Werkzeug nicht gefunden werden. Die Menge der false negatives ist definiert als $R \setminus M$.

Im Idealfall gilt $M = R$, das heißt, das gefundene Mapping entspricht exakt dem Referenz-Mapping, es enthält also nur true positives. In der Praxis tritt dieser Fall jedoch so gut wie nie

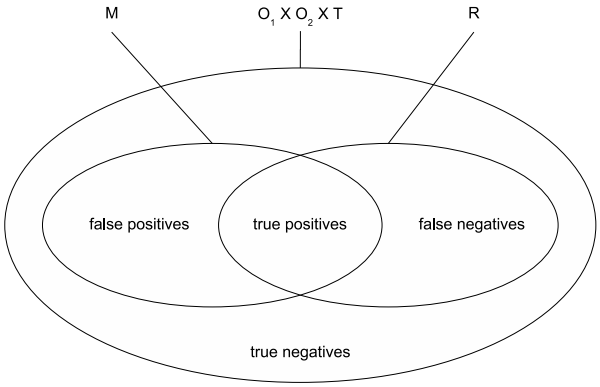



**Abbildung 3.16.:** True positives, true negatives, false positives und false negatives bei einem Mapping $M$ und einem Referenzmapping $R$ (nach Euzenat und Shvaiko, 2007, S. 206)

ein, weswegen man anhand dieser Begriffe verschiedene Qualitätsmaße definiert, die aussagen, bis zu welchem Grad man sich diesem Ideal angenähert hat.

### 3.5.1. Precision

Precision misst den Anteil korrekter Mapping-Elemente in einem von einem Matching-Werkzeug gefundenen Mapping. Precision ist definiert als

$$P(M,R) := \frac{|M \cap R|}{|M|} \in [0,1] \tag{3.4}$$

Ein hoher Precision-Wert sagt, dass ein Mapping-Element, das von einem Matching-Werkzeug gefunden wurde, mit hoher Wahrscheinlichkeit korrekt ist. Ein Precision-Wert von 0 bedeutet, dass keines der gefundenen Mapping-Elemente korrekt ist, ein Precision-Wert von 1, dass alle gefundenen Mapping-Elemente korrekt sind. Precision steht damit für einen niedrigen Anteil an false positives, ohne eine Aussage über false negatives zu treffen.

Da zur Berechnung der Precision nur überprüft werden muss, ob ein gefundenes Mapping-Element korrekt ist, ist es nicht notwendig, a priori ein komplettes Referenz-Mapping zu erstellen (Euzenat und Shvaiko, 2007, S. 206).

### 3.5.2. Recall

Recall misst den Anteil gefundener korrekter Mapping-Elemente am Referenz-Mapping. Recall ist definiert als

$$R(M,R) := \frac{|M \cap R|}{|R|} \in [0,1] \tag{3.5}$$

Ein hoher Recall-Wert sagt aus, dass ein großer Teil der Elemente des Referenz-Mappings gefunden werden. Bei einem Wert von 1 werden alle Elemente des Referenz-Mappings gefunden, bei einem Wert von 0 keines. Recall steht damit für einen niedrigen Anteil an false negatives, ohne eine Aussage über false positives zu treffen. Recall wird im Data Mining auch als *true positive rate* bezeichnet (Witten, 2005, S. 172).

### 3.5.3. F-Measure

Ein sehr pessimistisches Verfahren erzielt meist einen hohen Precision-Wert auf Kosten vieler false negatives und damit einen niedrigen Recall-Wert, während ein optimistisches Verfahren einen hohen Recall-Wert auf Kosten vieler false positives und damit einen niedrigen Precision-Wert erzielt (Ehrig, 2007b, S. 83). Daher ist es sinnvoll, zur Qualitätsmessung ein weiteres Maß einzuführen, das beide Messwerte vereint. Van Rijsbergen definiert dazu das Maß F-Measure (van Rijsbergen, 1979, S. 134[6]) als:

$$F_\alpha(M,R) := \frac{P(M,R)\cdot R(M,R)}{(1-\alpha)\cdot P(M,R)+\alpha\cdot R(M,R)} \in [0,1] \tag{3.6}$$

Der Parameter $\alpha$ legt dabei das Gewicht von Precision und Recall fest. Ein Wert nahe 0 verschiebt das Gewicht zu Gunsten des Recall-Wertes, ein Wert nahe 1 zu Gunsten des Precision-Wertes; es gilt $F_0(M,R) = R(M,R)$ und $F_1(M,R) = P(M,R)$. Ein häufig verwendeter Wert ist $\alpha = 0.5$ (Witten, 2005, S. 172), für den das F-Measure dem harmonischen Mittel von Precision und Recall entspricht und sich vereinfacht schreiben lässt als

$$F_{0.5}(M,R) = \frac{2\cdot P(M,R)\cdot R(M,R)}{P(M,R)+R(M,R)} = \frac{2\cdot|M\cap R|}{|M|+|R|} \in [0,1] \tag{3.7}$$

Die Evaluation der vorgestellten Werkzeuge mit diesen Qualitätsmaßen liegt außerhalb des Fokus dieser Arbeit. Für eine detaillierte Darstellung sei z.B. auf Euzenat u. a. (2006) verwiesen.

## 3.6. Skalierbarkeit

Das im vorangegangenen Kapitel erläuterte Vorgehen, mit dem üblicherweise die Qualität von Matching-Werkzeugen geprüft wird, erklärt, warum meist nur mit kleinen Ontologien getestet wird: menschliche Experten mit dem Erstellen eines kompletten Referenz-Mappings für sehr große Ontologien zu beauftragen, ist oft zu teuer und aufwändig, und die Betrachtung des Precision-Wertes allein ist nicht aussagekräftig genug. Daher werden in den meisten Untersuchungen der Qualität von Werkzeugen nur kleine Ontologien verwendet (Do und Rahm, 2007, S. 857). Allerdings gibt es Foschungsarbeiten, in denen die Qualität von Matching-Werkzeugen an einem partiellen Referenz-Mapping von großen Ontologien gemessen wurde (z. B. Zhang u. a., 2004, S. 102 ff.).

Außerdem sind Skalierbarkeit und Effizienz über lange Zeit keine vorrangigen Themen in der Entwicklung von Ontology-Matching-Werkzeugen gewesen; der Fokus lag vielmehr auf der Verbesserung der Ergebnisqualität (vgl. Ehrig und Staab, 2004a, S. 683), teilweise auch für künstliche, akademisch möglicherweise interessante, aber praxisferne Beispiele, wie z.B. Ontologien ohne Bezeichner und ohne Strukturinformationen (Ehrig und Sure, 2005, S. 73). Erst in der letzten Zeit widmen sich die Entwickler auch diesen Themen.

[6] Van Rijsbergen erklärt nicht, wofür das *F* in *F-Measure* steht, und auch in der späteren Literatur finden sich dazu keine Hinweise.

### 3.6.1. Skalierbarkeitsprobleme

In den Bereichen Medizin und Biowissenschaft sowie im E-Business werden oft große Ontologien eingesetzt (siehe Shadbolt u. a., 2006, S. 96 und Hepp, 2006, S. 2). Daher wurden, um die Skalierbarkeit von Ontology-Matching-Werkzeugen zu testen, zwei Paare von großen OWL-Ontologien aus diesen Bereichen eingesetzt: der NCI Thesaurus und die Gene Ontology aus der Medizin und die beiden Produktontologien eCl@ss und UNSPSC aus dem Bereich e-Business. Die Testdaten und die Konfiguration der Testumgebung sind in Anhang A beschrieben.

Keines der getesteten Werkzeuge konnte das Paar der größeren Testontologien aufeinander abbilden; die genauen Testergebnisse sind in Tabelle 3.1 dargestellt. Besonders überraschend ist, dass auch die Werkzeuge Falcon-AO und COMA++, die beide explizit zum Matching großer Ontologien angelegt sind (vgl. Do und Rahm, 2007, S. 857 ff. und Hu u. a., 2006b, S. 72 ff.), die sehr großen Ontologien ebenfalls nicht verarbeiten konnten. Der Grund der `NullPointerException` bei FOAM ist unbekannt. Die vom INRIA-Align-System gemeldete Systembeschränkung ist eine Beschränkung des von diesem System genutzten Parsers zum Einlesen der OWL-Dateien.

| **Werkzeug** | **GO + NCI** | **eCl@ss + UNSPSC** |
|---|---|---|
| FOAM | Bricht nach 17h mit einer `NullPointerException` ab | Bricht nach 6h mit einer `NullPointerException` ab |
| INRIA | Bricht nach 95h mit einem `OutOfMemoryError` ab | meldet nach wenigen Sekunden, dass die Ontologien aufgrund einer Systembeschränkung nicht geladen werden können |
| PROMPT | Protégé bricht beim Versuch, die Quellontologie zu laden, nach wenigen Minuten mit einem `OutOfMemoryError` ab | |
| COMA++ | Bricht nach 16h mit einem `OutOfMemoryError` ab | findet kein Mapping |
| CROSI (einfacher String-Vergleich) | Versuch wurde nach vier Wochen ergebnislos abgebrochen | Versuch wurde nach vier Wochen ergebnislos abgebrochen |
| Falcon-AO | Bricht nach 6h mit einem `OutOfMemoryError` ab | findet kein Mapping |

**Tabelle 3.1.:** Testergebnisse der einzelnen Matching-Werkzeuge

Der Test zeigt, dass die Mehrzahl der Werkzeuge Probleme mit der Größe des Hauptspeichers hat. Dies liegt daran, wie diese Werkzeuge meist implementiert werden. Quelltextbeispiel 3.7 zeigt eine solche prototypische Implementierung. Hier gibt es im wesentlichen zwei Probleme:

- Zum einen werden bei den meisten Werkzeugen die Ontologien im ersten Schritt komplett als Objektmodelle in den Hauptspeicher geladen, was diesen selbst bei einer vergleichsweise großzügigen Speicherzuweisung von 1GB oft bereits füllt (Kalfoglou u. a., 2005, S. 35).

- Zum anderen werden auch bis zum Ende des Matching-Prozesses sämtliche Mapping-Elemente im Hauptspeicher gehalten, um ganz zum Schluss zurückgegeben zu werden – da bei großen Ontologien derselben Domäne auch viele solcher Elemente existieren können, führt dies ebenfalls zu einer starken Speicherauslastung.

Aufgrund dieser Implementierungsansätze wird oft bereits sehr viel Hauptspeicher genutzt, ohne dass das eigentliche Matching schon berücksichtigt wäre. Dessen Implementierung kann jedoch mit zahlreichen temporären Variablen und dergleichen zusätzlich viel Hauptspeicher verbrauchen.

```
List<MappingElement> matchOntologies(URI onto1, URI onto2) {
  Ontology ontology1 = loadOntologyIntoMemory(onto1);
  Ontology ontology2 = loadOntologyIntoMemory(onto2);

  List<MappingElement> lstMapping = new List<MappingElement>();

  // some preprocessing
  // ...

  // find mappings - this can be pretty complex in reality!
  while(moreMappingsAreFound) {
    lstMapping.add(new MappingElement(...));
  }

  // some postprocessing
  // ...

  return lstMapping;
}
```

**Quelltextbeispiel 3.7:** Pseudocode-Darstellung eines Matching-Prozesses

Auch die Zeitkomplexität ist beim Matching großer Ontologien ein Problem, wenngleich es sich in den Ergebnissen dieses Versuchs nur im Fall des CROSI Mapping Systems niederschlägt. Da die meisten Matching-Werkzeuge jedes Element einer Ontologie mindestens einmal mit jedem Element einer anderen Ontologie vergleichen, beträgt der Zeitaufwand in der Regel mindestens $O\left(n^2\right)$, je nach Implementierung des Vergleichsoperators. Bei strukturbasierten Verfahren liegt dieser Wert in der Regel noch höher (Ehrig und Staab, 2004a, S. 692 f.). Der Partionierungsalgorithmus, der im Werkzeug Falcon-AO verwendet wird, hat eine Laufzeitkomplexität von $O\left(n^2\right)$ (Hu und Qu, 2006, S. 76).

### 3.6.2. Skalierbarkeitstechniken

In einigen der vorgestellten Werkzeuge finden sich bereits Ansätze zum Erreichen einer gewissen Skalierbarkeit. Die Partitionierung von Ontologien, wie sie bei COMA++ und Falcon-AO durchgeführt wird, erlaubt, das eigentliche Matching-Problem auf kleineren Teilontologien durchzuführen. Allerdings wird bei dieser Technik das Skalierbarkeitsproblem nur auf einen anderen Schritt, nämlich das Fragmentieren selbst, verlagert. Am Fragmentieren der großen Eingabeontologien scheiterte im Versuch insbesondere das Werkzeug Falcon-AO.

Die Wahl geeigneter Speicherstrukturen ist für die Skalierbarkeit ebenso essentiell. Bis auf COMA++ versuchen die meisten Werkzeuge, die Ontologien, wie in Quelltextbeispiel 3.7 skizziert, komplett in den Hauptspeicher zu laden. COMA++ dagegen speichert die Ontologien in einer relationalen Datenbank, auch, um den Hauptspeicher zu entlasten (Do und Rahm, 2007, S. 859). Ein solcher Ansatz erscheint eher geeignet, um ein skalierbares Matching-Werkzeug zu entwickeln. Dabei sollte besonderes Augenmerk auf eine effiziente Persistenz-Lösung gelegt werden (vgl. Kapitel 2.5.3).

Die Laufzeitkomplexität von $O\left(n^2\right)$, die sich ergibt, wenn man alle Elemente beider Ontologien paarweise miteinander vergleicht, lässt sich nach Ehrig und Staab durch die Beschränkung auf den Vergleich geeigneter Elemente reduzieren. Dazu können die Elemente in sortierten Listen gehalten werden, aus denen nur Elemente an äquivalenten Positionen miteinander verglichen werden. Damit lässt sich die Laufzeitkomplexität auf $O\left(n \cdot \log n\right)$ verringern, ohne dass die Qualität des Mappings nennenswert leidet (Ehrig und Staab, 2004a, S. 692 f.).

Kombiniert man diesen Ansatz mit der Fragmentierung, so ist es außerdem möglich, nicht alle Fragmente paarweise miteinander zu vergleichen, wie z.B. bei Falcon-AO implementiert(Hu u. a., 2006b, S. 72 f.), sondern nur Fragmente, die eine gewisse Ähnlichkeit aufweisen. Dabei muss der Test auf Ähnlichkeit von Fragmenten weniger aufwändig sein als das komplette Matching von Fragmenten, um einen Skalierbarkeitszugewinn zu erreichen. Dieser Ansatz wird von COMA++ implementiert (Do und Rahm, 2007, S. 871 f.).

Im folgenden Teil wird ein Prototyp vorgestellt, der einige dieser Ideen aufgreift, um damit ein skalierbares Ontology-Matching-System zu entwickeln.

# 4. Entwicklung eines Prototypen

Im Rahmen dieser Masterarbeit wurde ein Prototyp entwickelt, der in der Lage ist, auch Matchings sehr großer Ontologien durchzuführen. Design und Implementierung des Prototypen sowie eine Evaluation der Matching-Ergebnisse werden in diesem Kapitel dargestellt.

## 4.1. Implementierung

Im vorherigen Kapitel wurden Techniken skizziert, mit denen skalierbare Ontology-Matching-Systeme entwickelt werden können. Mit Blick auf diese Techniken wurde die Architektur des Prototypen-Systems entworfen.

### 4.1.1. Grundlegende Überlegungen

Auch wenn die im letzten Kapitel untersuchten Werkzeuge nicht skalierbar waren, erscheint zumindest der Ansatz der Partitionierung von Ontologien, mit Hilfe dessen das Matching-Problem in kleinere Teilprobleme zerlegt werden kann, als besonders vielversprechend. Die Partitionierung von Ontologien soll daher einen Kernbaustein des Systems bilden. Da es unterschiedliche mögliche Mechanismen zur Partitionierung von Ontologien gibt, soll unter verschiedenen dieser Mechanismen ausgewählt werden können.

Das Partitionieren der Ontologien muss, um Skalierbarkeit zu gewährleisten, außerhalb des Hauptspeichers geschehen. Daher sollten die Eingabeontologien zunächst in eine relationale Datenbank geladen werden, die einen effizienten Zugriff auf die Ontologieelemente erlaubt. Die Partitionen werden ebenfalls in der relationalen Datenbank abgelegt, die Partitionierung kann daher in Form von Operationen auf der Datenbank implementiert werden, es ist folglich nicht nötig, die Ontologien hierzu komplett im Hauptspeicher zu halten.

Das Matching wird dann paarweise auf den Partitionen ausgeführt. Diese haben moderate Größen und stellen daher, im Gegensatz zu den Eingabeontologien, keine besonderen Anforderungen an die eingesetzten Matching-Werkzeuge. Durch dieses Design können beliebige Werkzeuge zum Matching der Partitionen wiederverwendet werden.

Die von den Matching-Werkzeugen produzierten Teilergebnisse sollen nach der Ausführung jedes paarweisen Matchings wahlweise in einer Datei oder in einer Datenbank gespeichert werden. So kann verhindert werden, dass größere Mengen von Mapping-Elementen im Hauptspeicher gehalten werden müssen.

Prinzipiell wäre es auch möglich gewesen, bestehende Matching-Algorithmen so neu zu implementieren, dass sie auf einer persistent gespeicherten Ontologie arbeiten und so darauf verzichten können, die Gesamtontologie in den Speicher zu laden. Dieser Ansatz hätte den Vorteil, dass die zusätzlichen Schritte der Partitionierung der Ontologien und der Nachbear-

beitung der gefundenen Mappings, auf die noch eingegangen wird, entfallen könnten. Dagegen bietet der in dieser Arbeit umgesetzte Ansatz mehrere Vorteile:

- Bestehende Matching-Werkzeuge können, ebenso wie deren zukünftige Weiterentwicklungen, ohne größeren Implementierungsaufwand wiederverwendet werden. Dasselbe gilt für bestehende Werkzeuge zur Partitionierung.

- Die modulare Architektur erlaubt die Kombination unterschiedlicher Matching- und Partitionierungswerkzeuge. Damit kann mit Hilfe des Systems untersucht werden, welche Partitionierungstechniken für das Matching-Problem gut einsetzbar sind, und wie gut bestimmte Matching- und Partitionierungswerkzeuge in Kombination funktionieren.

- Enthält eine Ontologie einzelne (z.B. fehlerhafte oder widersprüchliche) Aussagen, mit denen ein Matching-Werkzeug nicht umgehen kann[1], so werden bei einem partionenbasierten Ansatz nur diejenigen Partitionen verworfen, in denen diese Statements enthalten sind. Es besteht so die Chance, zumindest ein teilweises Mapping zu finden.

Darüber hinaus bietet die Zerlegung des Matching-Problems in kleinere Teilprobleme einen weiteren Vorteil: die Teilprobleme sind voneinander unabhängig und können verteilt auf mehreren Rechnern gelöst werden, um den Gesamtprozess zu beschleunigen. Die Umsetzung dieses Ansatzes wird in Kapitel 4.4.1 dargestellt.

### 4.1.2. Systemarchitektur

Die Steuerungsklasse des Prototypen ist die Klasse `LargeOntologyMatcher`. Bei ihrem Aufruf werden die beiden Eingabeontologien, für die ein Matching ausgeführt werden soll, als Parameter übergeben. Diese Klasse initialisiert anhand von Angaben aus einer Konfigurationsdatei die weiteren benötigten Komponenten und startet den Matching-Prozess der Eingabeontologien.

Für jede der Grundfunktionalitäten des Systems – Partitionierung der Ontologien, Matching der Partionen, Filtern der Ergebnisse usw. – gibt es abstrakte Basisklassen. Ein komplettes Klassendiagramm ist in Abbildung 4.1 dargestellt[2].

Die Ontologien und Partitionen werden mit Hilfe eines Persistenzdienstes gespeichert. Der Persistenzdienst wird in der Klasse `AbstractOntologyStorageFacade` gekapselt; im System ist eine konkrete Implementierung für den Jena-Datenbank-Persistenzdienst (siehe Kapitel 2.5.3.1) enthalten.

Die Klasse `AbstractOntologyPartioner` übernimmt das Partitionieren einer Ontologie. Sie hat Zugriff auf den Persistenzdienst, um große Ontologien außerhalb des Hauptspeichers partitionieren zu können, und legt für jede Partition eine neue persistente Ontologie an. Im System sind drei verschiedene Partitionierungsalgorithmen implementiert: der einfache

[1] Insbesondere bei FOAM zeigte sich während der Entwicklung, dass hin und wieder bestimmte Ontologien oder Ontologiepartitionen von dem Werkzeug gar nicht geladen werden.

[2] Die grau markierten Klassen stellen Erweiterungen dar, die zur Optimierung des Systems dienen. Sie werden in Abschnitt 4.4 erläutert.

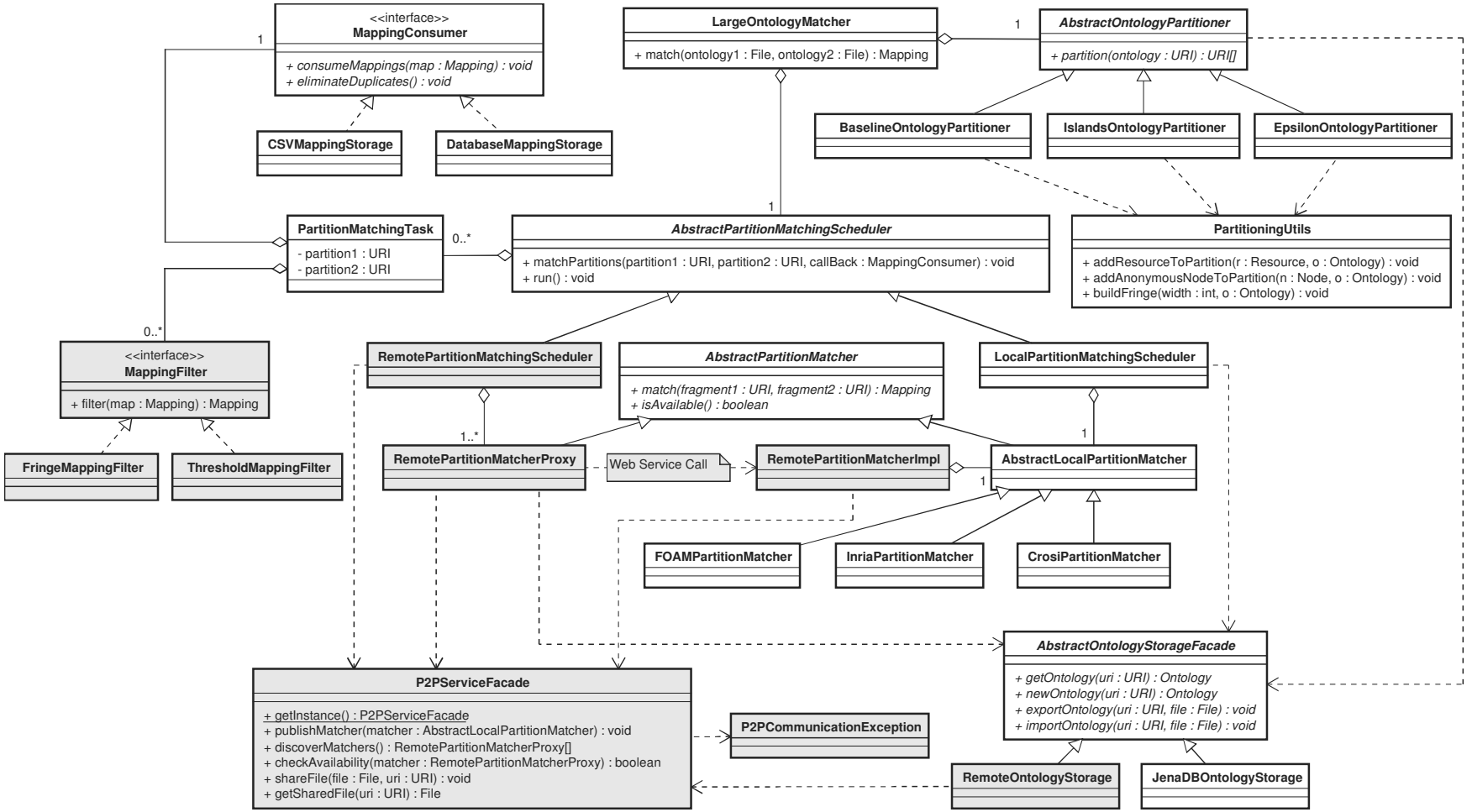



**Abbildung 4.1.:** Klassendiagramm

Baseline-Algorithmus, der Islands-Algorithmus und der $\varepsilon$-Connections-Algorithmus. Die Klasse `PartitioningUtils` enthält darüber hinaus Funktionen, die von allen Partitionierungsalgorithmen benötigt werden.

Die Paare von Partitionen, für die ein Matching erstellt werden soll, werden von der Klasse `AbstractPartitionMatchingScheduler` verwaltet; jedes dieser Paare wird durch ein Objekt der Klasse `PartitionMatchingTask` repräsentiert.

Das Matching der Partitionen erfolgt mit einer Implementierung der Klasse `AbstractPartitionMatcher`. Hierin wird jeweils der Aufruf eines bestehenden Matching-Werkzeugs gekapselt. Es existieren Implementierungen für die Nutzung FOAM, INRIA und CROSI CMS (siehe Kapitel 3.4). Vor dem Aufruf werden die Partitionen zunächst in je eine temporäre Ontologiedatei exportiert, weil die meisten Matching-Werkzeuge nur Dateien, aber keine in Datenbanken gespeicherten Ontologien als Eingaben verarbeiten.

Die Weiterverabeitung der Ergebnisse erfolgt, sobald ein Teilergebnis für ein Paar von Partitionen vorliegt. Jedes Teilergebnis wird zur Speicherung an eine Klasse, die das Interface `MappingConsumer` implementiert, übergeben. Es stehen Implementierungen für die Speicherung in CSV-Dateien und in einer Datenbanktabelle zur Verfügung. Nachdem alle Teilergebnisse vorliegen, werden aus den gespeicherten Mapping-Elementen Duplikate eliminiert. Abbildung 4.2 zeigt das Zusammenspiel der Klassen als Sequenzdiagramm.

## 4.2. Ablauf des Matching-Prozesses

Aus dem Zusammenspiel der Klassen ist ersichtlich, dass das System in mehreren Schritten abläuft. Nach dem initialen Laden der Ontologien werden diese zunächst partitioniert. Die einzelnen Partitionen werden paarweise einem Matching-Werkzeug übergeben, das daraus ein Teilmapping erzeugt. Die Teilmappings werden zwischengespeichert und, wenn alle Teilmappings vorliegen, von Duplikaten bereinigt, bevor das resultierende Gesamtmapping dem Nutzer zur Verfügung gestellt wird.

### 4.2.1. Einlesen der Ontologien

Zunächst müssen die Ontologien, die in der Regel in Form von OWL-Dateien vorliegen, in das System eingelesen werden. Im Prototypen werden sie in den datenbankbasierten Speicher geladen. Hierbei können optimierte Verfahren zum Laden großer Ontologien eingesetzt werden (siehe Kapitel 2.5.3.1). Nach dem Einlesen der Ontologien können diese mit Hilfe von Datenbankoperationen verarbeitet werden, ohne dass sie komplett in den Hauptspeicher geladen werden müssen.

### 4.2.2. Partitionierung der Ontologien

Im nächsten Schritt werden die Ontologien in einzelne Partitionen zerlegt. Diese Partitionen sind wiederum Ontologien. Fasst man eine Ontologie nach der einfachen Definition 2.1 auf Seite

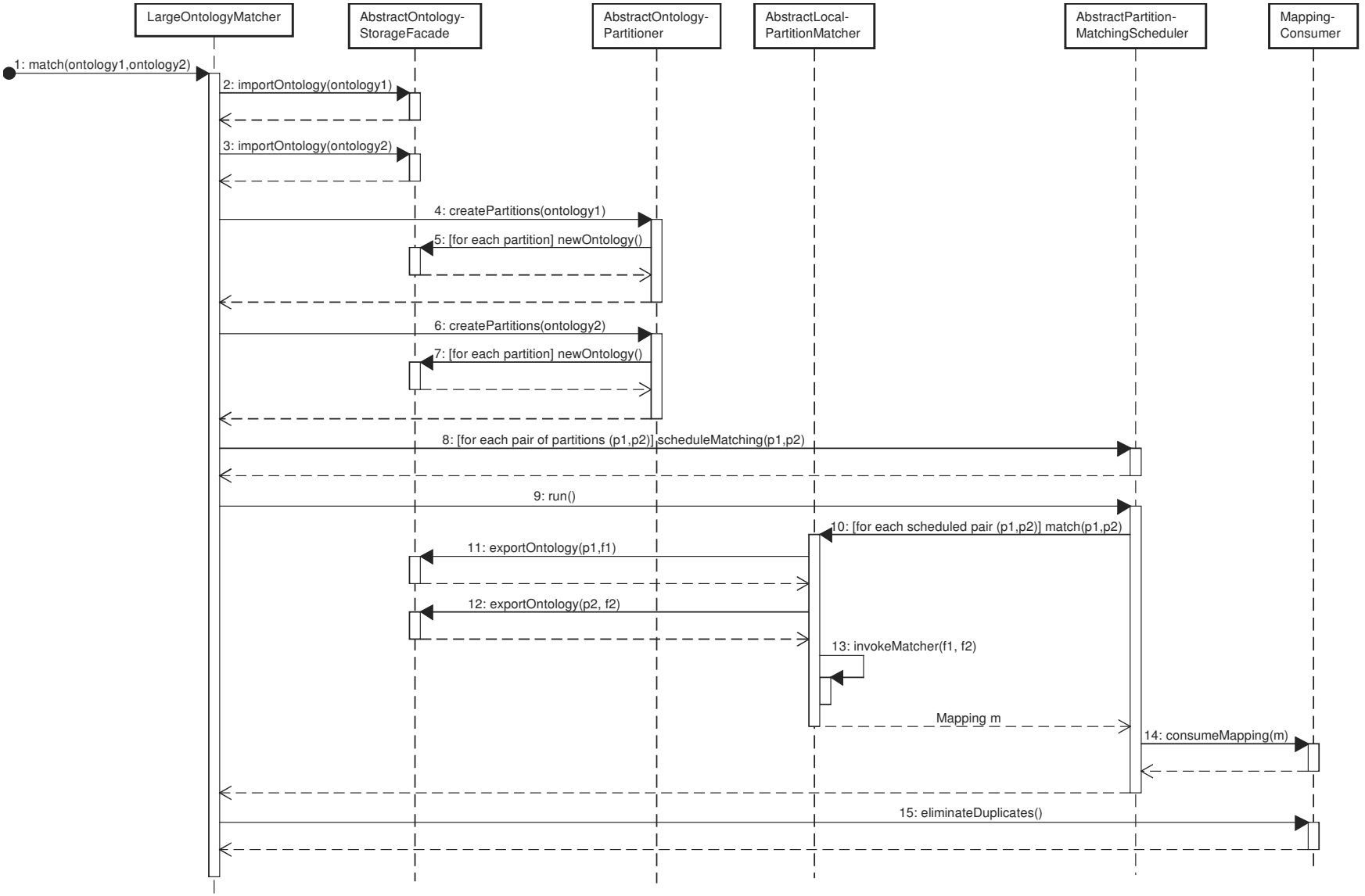



**Abbildung 4.2.:** Ablauf des Matching-Prozesses als Sequenzdiagramm

10 auf, so lässt sich eine Menge $P$ von Partitionen nach Stuckenschmidt und Klein (2004a) wie folgt definieren:

$$P := \{P_1, ..., P_n\} \text{ mit } P_i := \langle S_i, A_i \rangle, S_i \subseteq S, A_i \subseteq A, S_i \neq \emptyset \vee A_i \neq \emptyset. \tag{4.1}$$

Stuckenschmidt und Klein definieren außerdem, dass eine Menge von Partitionen *abdeckend* ist, wenn

$$\bigcup_{1 \leq i \leq n} S_i = S \text{ und } \bigcup_{1 \leq i \leq n} A_i = A \tag{4.2}$$

gilt, und dass die Partitionen *disjunkt* sind, wenn

$$i \neq j \Rightarrow S_i \cap S_j = \emptyset \wedge A_i \cap A_j = \emptyset \tag{4.3}$$

erfüllt ist.

Diese Definitionen lassen sich auch auf die komplexeren Ontologiedefinitionen in Kapitel 2.2.2 übertragen.

Für das Ontology-Matching-Problem ist es sinnvoll, dass die erzeugten Partitionen abdeckend sind, damit alle möglichen Mapping-Elemente gefunden werden. Disjunkte Partitionen sind in den meisten Fällen nicht möglich, wenn man gleichzeitig Abdeckung fordert, da beim Partitionieren eines komplett zusammenhängenden Graphen Knoten verdoppelt werden müssen, da wegen des Abdeckungskriteriums keine Kanten verworfen werden sollen. Ein weniger striktes Kriterium sieht vor, die *Redundanz*, also die Menge der verdoppelten Knoten und Kanten, möglichst gering zu halten (Schlicht und Stuckenschmidt, 2006, S. 5).

#### 4.2.2.1. Der Baseline-Algorithmus

Ein einfacher und naiver Weg, eine Ontologie zu partitionieren, ist in der Klasse `BaselineOntologyPartitioner` implementiert und wird im Folgenden als *Baseline-Algorithmus* bezeichnet, da er als Grundgröße für die Messung der Verbesserungsmöglichkeiten, die ausgefeiltere Partitionierungsmechanismen bieten, dient.

Der Baseline-Algorithmus basiert auf der Tripel-Darstellung von Ontologien im RDF-Format (siehe Kapitel 2.3.1). Er iteriert über alle Resourcen, die als Subjekte von Aussagen in einer Ontologie auftreten, und fügt stets alle Aussagen über diese Subjekte in eine Partition ein, bis eine Partition eine bestimmte Größe überschritten hat. Die Funktionsweise des Baseline-Algorithmus ist in Quelltextbeispiel 4.1 skizziert.

Wird ein anonymer Knoten (siehe Kapitel 2.3.1) als Objekt einer Aussage angetroffen, so werden alle Aussagen, die diesen Knoten als Subjekt oder Objekt besitzen, mit in die Partition eingefügt. Auf diese Weise wird dafür gesorgt, dass komplexe Aussagen als Einheiten erhalten bleiben. Würde man diesen Sonderfall nicht behandeln, so entstünden inhaltlich andere Aussagen (vgl. Horrocks u. a., 2003, S. 12). Außerdem werden für jedes Subjekt und jedes Objekt die Typ-Informationen redundant in die Partitionen kopiert, da in OWL DL jedes Konzept einen Typ besitzen muss und einige Matching-Werkzeuge OWL-DL-konforme Ontologien als Eingabe benötigen.

```
Ontology[] partition-baseline-N(Ontology ontology) {
  Ontology[] partitions;
  currentPartition = new Ontology();
  partitions.add(currentPartition);
  for(each subject x in ontology) {
    if(x is not an anonymous node) {
      for(each statement s that has x as subject) {
        add s to currentPartition
        if(s.object is an anonymous node)
          add all statements connected to s.object to currentPartition
      }
      add type statements for each object added to currentPartition
    }
    if(currentPartition.size > N) {
      currentPartition = new Ontology();
      partitions.add(currentPartition);
   }
  }
  return partitions;
}
```

**Quelltextbeispiel 4.1:** Der Baseline-Partitionierungs-Algorithmus in Pseudocode

Da die Tripel in keiner bestimmten Reihenfolge sortiert sind, ist auch ihre Aufteilung auf die Partitionen zufällig[3]. Aussagen über ein bestimmtes Subjekt sind zwar stets in derselben Partition enthalten, aber Aussagen, die eine Resource als Objekt enthalten, können über verschiedene Partitionen verteilt sein.

Wird der Baseline-Algorithmus komplett in Form von Datenbankoperationen implementiert, so ist sein Speicherverhalten optimal: es wird stets nur ein Tripel und eine Liste der bisherigen Partitionen im Speicher gehalten, wobei in dieser Liste nur Bezeichner der Partitionen, nicht die Partionen selbst enthalten sind. Bei $n$ Aussagen in einer Ontologie ist die Speicherkomplexität folglich $O\,(n/N)$.

Der Baseline-Algorithmus erzeugt eine abdeckende Partitionierung der Eingabeontologie.

#### 4.2.2.2. Der Islands-Algorithmus

Während der Baseline-Algorithmus nur einzelne Aussagen betrachtet, gibt es auch fortgeschrittene Algorithmen, die Ontologien nach Sinneinheiten zerlegen. Einer davon ist der von Stuckenschmidt und Klein (2004b) beschriebene *Islands-Algorithmus*, der auch mit großen Ontologien bereits erfolgreich getestet wurde. In dieser Arbeit wurde die Implementierung des Werkzeugs *PATO* (Stuckenschmidt und Klein, 2005)[4] eingesetzt.

[3] Eine bestimmte Sortierung der Tripel – etwa nach Themenbereichen – kann dabei durchaus auch zu einer weitgehend thematischen Partitionierung führen. Dies wäre jedoch ein zufälliger Effekt.

[4] Einige Formeln wurden gegenüber dem Artikel von Stuckenschmidt und Klein (2004b) leicht modifiziert. Nicht alle Beispiele im Artikel lassen sich mit den darin enthaltenen Formeln nachvollziehen, wohl aber mit den modifzierten, die in dieser Arbeit angegeben sind. Die Abweichungen von den Original-Formeln sind jeweils gekennzeichnet.

Die Grundidee des Algorithmus besteht darin, sogenannte *Line Islands* zu finden. Dabei wird die T-Box einer Ontologie als Graph betrachtet (vgl. Kapitel 2.3.1): Die Klassen bilden die Knoten, die Verbindungen zwischen den Klassen mit Prädikaten die Kanten. Anschaulich entspricht ein Line Island einer Teilontologie, die stärker in sich zusammenhängt, als sie mit allen anderen Elementen verbunden ist.

Formal werden Line Islands wie folgt definiert: zunächst wird eine Abhängigkeit $\omega$ zwischen zwei Klassenen $c_i$ und $c_k$ definiert als[5]

$$\omega\left(c_i, c_j\right) := \frac{a_{ij}}{\sum_k \left(a_{ik} + a_{ki}\right)} \tag{4.4}$$

Dabei ist $a_{ij}$ das Gewicht, das einer Kante vorab zugewiesen ist. Bei Stuckenschmidt und Klein (2004b) und in dieser Arbeit wird dafür stets 1 angenommen, es ist aber prinzipiell denkbar, verschiedene Relationstypen unterschiedlich zu gewichten. Eine sinnvolle Anwendung einer solchen Gewichtung wäre, Kanten, die Klassenäquivalenzen repräsentieren, hoch zu gewichten, damit äquivalente Klassen stets in derselben Partition liegen.

Bei konstanten Kantengewichten von 1 ist die Abhängigkeit zwischen zwei Klassen $c_i$ und $c_j$ der Kehrwert der Anzahl der mit $c_i$ verbundenen Klassen. Abbildung 4.3 zeigt den Graphen einer kleinen Beispielontologie, in Abbildung 4.4 sind die entsprechenden Abhängigkeiten eingetragen.

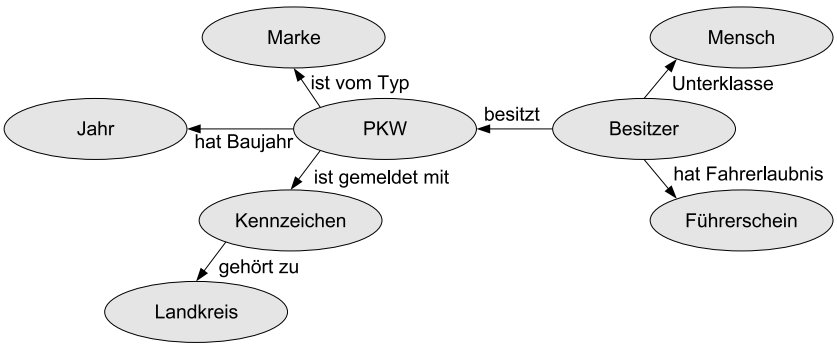



**Abbildung 4.3.:** Eine Beispielontologie, die partitioniert werden soll

Mit diesen Gewichten wird der Abhängigkeitsgraph $O$ definiert als $G := \langle V, E, \omega \rangle$, wobei $V$ die Menge der Klassen und $E$ die Menge der Kanten zwischen diesen ist. Ein Line Island ist dann nach Stuckenschmidt und Klein (2004b) definiert als Untermenge $I \subseteq V$, für die gilt:

1. $I$ induziert einen zusammenhängenden, ungerichteten Subgraphen von $G$.
2. Es gibt einen gewichteten Graphen $T = \langle I, E_T, \omega_T \rangle$, für den gilt:
   a) $T$ ist in $G$ eingebettet

[5] Bei Stuckenschmidt und Klein (2004b) steht im Zähler $a_{ij} + a_{ji}$, wobei sowohl $a_{ij}$ als auch $a_{ji}$ gleich 1 gesetzt wurden, was offensichtlich ein Widerspruch ist. Aus diesem Grund erscheint die obige Formel korrekter und sinnvoller.

b) $T$ ist ein maximaler Spannbaum des von $I$ induzierten Subgraphen

c) Es gilt folgende Gleichung[6]:

$$\max_{\{(v,w)\in E_T|(v\in I\wedge w\notin I)\vee(v\notin I\wedge w\in I)\}} \omega(v,w) < \min_{u,w\in I} \max(\omega(u,w),\omega(w,u)) \tag{4.5}$$

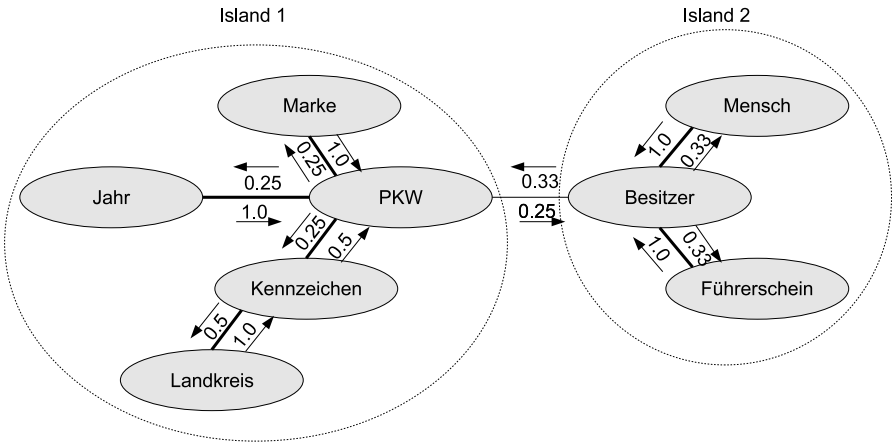



**Abbildung 4.4.:** Zwei Partitionen, die mit dem Islands-Algorithmus gefunden werden

Diese Definition trifft z.B. auf die beiden in Abbildung 4.4 markierten Inseln zu: Die minimale Abhängigkeit zwischen zwei Klassen in *Island 1* ist 0.5 und damit größer als die maximale Abhängigkeit einer Klasse der Insel zu einer anderen, nicht in der Insel enthaltenen Klasse (0.33). Dasselbe trifft auf *Island 2* zu.

Im Werkzeug *PATO* (Stuckenschmidt und Klein, 2005) wird die Suche nach Inseln wiederholt ausgeführt, wobei sukzessive versucht wird, größere Inseln in kleinere zu zerlegen. Anschließend werden zu kleine Inseln wieder zu größeren zusammengefasst und isolierte Knoten bestehenden Inseln zugewiesen. Details zur Implementierung von PATO finden sich in Stuckenschmidt und Klein (2004b, S. 294 ff.).

Die verwendete Implementierung des Islands-Algorithmus ist darauf ausgelegt, Light-Weight-Ontologies (vgl. Kapitel 2.2.1) und insbesondere deren Klassenhierarchie zu segmentieren (Stuckenschmidt und Klein, 2004b, S. 291). Daher ist die resultierende Partitionierung zwar abdeckend bezüglich der in der Ausgangsontologie enthaltenen Klassen, allerdings nicht bezüglich der in der Ausgangsontologie enthaltenen Prädikate und Instanzen. Wird z.B. die Ontologie aus Abbildung 4.3 wie in Abbildung 4.4 partitioniert, so ist das Prädikat `besitzt` anschließend in keiner der beiden Partitionen enthalten.

In der Implementierung im Prototypen wird das Ergebnis der PATO-Implementierung weiterverarbeitet, um zumindest einen großen Teil dieser Informationen zu rekonstruieren: so

[6] Auch diese Gleichung ist gegenüber dem Original-Paper von Stuckenschmidt und Klein (2004b) leicht modifiziert, in der nicht das Minimum aller Maxima beider Gewichte einer Kante, sondern das Minimum aller Kantengewichte in $T$ verwendet wird. Mit der Original-Gleichung lassen sich die Beispiele im Artikel nicht nachvollziehen, wohl aber mit der obigen, variierten Fassung.

werden alle Verbindungen innerhalb einer gefundenen Insel und alle Instanzen der in einer Insel enthaltenen Klassen in die entsprechende Partition übernommen.

#### 4.2.2.3. Der $\varepsilon$-Connections-Algorithmus

Ein weiterer fortgeschrittener Algorithmus zum Partitionieren von Ontologien ist der $\varepsilon$-Connections-Algorithmus. Er kann automatisiert eine Menge von Partitionen aus einer Ontologie erzeugen, die durch $\varepsilon$-Connections verbunden sind (siehe Kapitel 3.1.4.5). Für die Integration wurde die Implementierung des Algorithmus in dem Open-Source-Werkzeug *SWOOP* (Kalyanpur u. a., 2006) eingesetzt (siehe auch Kapitel 2.5.1.1) und in der Klasse `EpsilonOntologyPartitioner` gekapselt.

Im Gegensatz zum Islands-Algorithmus, der im Top-Down-Verfahren große zusammenhängende Blöcke in der Ontologie sucht und diese dann weiter unterteilt, arbeitet der $\varepsilon$-Connections-Algorithmus nach einem Bottom-Up-Verfahren: in jedem Schritt wird ein Element aus der Ausgangsontologie entfernt und in eine neu erzeugte Partition verschoben; dann werden weitere Elemente, die mit dem betrachteten Element in Verbindung stehen (das müssen nicht nur direkte Nachbarn sein), nachgezogen, sowie Verbindungen zu Elementen anderer Partitionen als $\varepsilon$-Connections, wie in Kapitel 3.1.4.5 definiert, hergestellt. Das Verfahren endet, wenn alle Elemente der Ausgangsontologie in eine Partition verschoben sind. Quelltextbeispiel 4.2 zeigt eine vereinfachte Version des Algorithmus, der vollständige Algorithmus ist in (Grau u. a., 2005, S. 7 ff.) nachzulesen.

Der $\varepsilon$-Connections-Algorithmus hat einige Schwächen: Große Ontologien werden nicht immer in hinreichend kleine Partitionen unterteilt, vielmehr gibt es Fälle, in denen als Ergebnis des Algorithmus nur die Ausgangsontologie als ein einzige große Partition zurückgegeben wird (Grau u. a., 2005, S. 14). In anderen Fällen werden sehr viele Partionen mit jeweils nur einem Konzept erzeugt (Schlicht und Stuckenschmidt, 2006, S. 9). Auch im Rahmen dieser Arbeit konnte dieses Verhalten beobachtet werden. Darüber hinaus brach der Algorithmus beim Partitionieren einiger Ontologien mit einem Speicherüberlauf ab, wobei nicht klar festzustellen war, ob dies auf den Algorithmus selbst oder auf einen Fehler der eingesetzten Implementierung zurückzuführen ist.

Die vom Algorithmus erzeugten Partitionen enthalten sowohl Aussagen der Original-Ontologien als auch $\varepsilon$-Connection-Aussagen, die die Partitionen zueinander in Beziehung setzen (siehe Kapitel 3.1.4.5). Da die $\varepsilon$-Connections eine Erweiterung des OWL-Standards sind, die von den meisten Matching-Werkzeugen nicht verarbeitet werden kann, werden die Verbindungen zwischen einzelnen Partitionen wieder entfernt, bevor die Partionen mit dem eingesetzten Matching-Werkzeug bearbeitet werden.

#### 4.2.2.4. Weitere Algorithmen

Darüber hinaus existieren weitere Algorithmen, mit denen Ontologien partitioniert werden, die aber aus unterschiedlichen Gründen nicht geeignet sind, um in dem implementierten System zum Einsatz zu kommen.

Guo und Heflin haben einen Algorithmus entwickelt, um die A-Box einer Ontologie effizient zu partitionieren (Guo und Heflin, 2006, S. 47 ff.). Da die im Rahmen dieser Arbeit betrachteten

```
Ontology[] epsilonPartitionOntology(Ontology ontology) {
  Ontology[] partitions;

  while(ontology.containsElements) {
    Ontology currentPartition = new Ontology();
    partitions.add(currentPartition);

    Element baseElement = ontology.getRandomElement();
    Element[] environment = ontology.getEnvironment(baseElement);

    currentPartition.add(baseElement);
    ontology.remove(baseElement);
    currentPartition.add(environment);
    ontology.remove(environment);

    for each(Property p connecting two elements in currentPartition) {
      currentPartition.add(p);
      ontology.remove(p);
    }
    for each(Property p connecting an element in currentPartition
                                to an element in ontology
                              or to an element in another partition) {
      currentPartition.addAsEpsilonConnection(p);
      ontology.remove(p);
    }
  }

  return partitions;
}
```

**Quelltextbeispiel 4.2:** Eine vereinfachte Version des $\varepsilon$-Connections-Algorithmus in Pseudocode

großen Ontologien jedoch oft große T-Boxen, aber keine oder nur wenige Instanzen enthalten, ist dieser Algorithmus für das Matching-Problem eher ungeeignet.

Außerdem existieren verschiedene Mechanismen, um einzelne Segmente aus einer Ontologie herauszulösen, die ein bestimmtes Konzept, z.B. eine Klasse, möglichst erschöpfend beschreiben. Sie erzeugen dazu eine Subontologie, die dieses Konzept und seine Umgebung enthält (Seidenberg und Rector, 2006, S. 19 ff.). Eine bekannte Implementierung ist das Werkzeug PROMPTFactor (siehe Kapitel 3.4.1). Allerdings lassen sich mit diesem Ansatz keine abdeckenden Partitionierungen, sondern nur die Umgebungen zu einzelnen Konzepten erzeugen. Zum Erstellen einer möglichst abdeckenden Partitionierung ist dieser Ansatz daher eher ungeeignet.

### 4.2.3. Matching von Partitionen

Für das Matching der Partitionen kann im implementierten Prototypen zwischen verschiedenen vorhandenen Matching-Werkzeugen gewählt werden. Derzeit sind die Werkzeuge FOAM, INRIA und CROSI CMS im System integriert, prinzipiell kann aber jedes beliebige Matching-Werkzeug angesprochen werden. Dabei sind alle automatisierten Arten von Matching-Verfahren denkbar. Um ein neues Matching-Werkzeug einzubinden, muss eine neue Subklasse von `AbstractLocalPartitionMatcher` implementiert werden.

Das gewählte Werkzeug wird für jedes Paar von Partitionen einmal aufgerufen und erzeugt auf diesen Partitionen ein Mapping als Zwischenergebnis.

Die Paare, für die ein Matching durchgeführt werden soll, werden von einer Subklasse von `AbstractPartitionMatchingScheduler` verwaltet. In der einfachsten Implementierung, der Klasse `LocalPartitionMatchingScheduler`, werden die Paare ohne besondere Reihenfolge an das gewählte Matching-Werkzeug übergeben. Um parallele Verarbeitung zu ermöglichen – während der Nachbearbeitung der Ergebnisse des Matchings eines Partitionenpaares kann bereits das nächste Paar an das Matching-Werkzeug übergeben werden – läuft der Scheduler in einem eigenen Thread.

### 4.2.4. Filtern und Speichern der Mappings

Nachdem alle Paare von Partitionen mit dem Matching-Werkzeug bearbeitet wurden, werden die Duplikate aus dem Gesamtergebnis gelöscht. Wendet man die beschriebene Technik des Partitionierens und Matchings der einzelnen Fragmente an, so kann das Problem auftreten, dass einige Mapping-Elemente doppelt gefunden werden. Dies ist darauf zurückzuführen, dass beim Partitionieren einzelne Knoten im RDF-Graphen der Ontologie verdoppelt werden – sofern die Ontologie nicht aus einzelnen, unverbundenen Komponenten besteht und alle Kanten in den Partitionen enthalten sein sollen, ist dies unvermeidbar.

Abbildung 4.5 zeigt das Problem: der Knoten `Bahn` aus der Ontologie taucht in beiden sowohl in Partition 1a als auch in Partition 1b auf. Wird nun ein Matching-Verfahren mit diesen beiden Partitionen ausgeführt, jeweils mit einer Partition der anderen Ontologie, die ein Gegenstück zum Konzept `Bahn` enthält, so wird die entsprechende Beziehung doppelt gefunden.

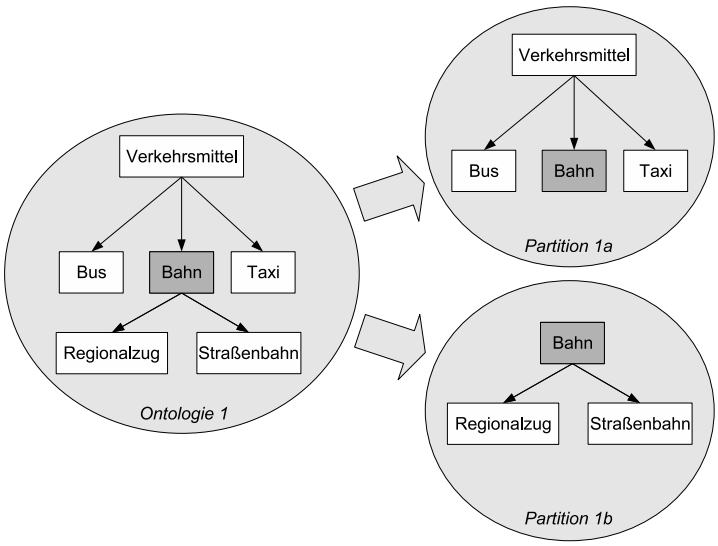



**Abbildung 4.5.:** Entstehen von Duplikaten durch Fragmentierung

Da das korrespondierende Konzept auch auf der anderen Seite in mehreren Partitionen enthalten sein kann, können Mapping-Elemente auch öfter als doppelt auftauchen. Außerdem ist es potentiell möglich, dass manche Beziehungen zwischen zwei Konzepten mit unterschiedlichen Beziehungstypen gefunden werden, je nachdem, wie das Konzept in den einzelnen Partitionen zu anderen Konzepten in Beziehung steht.

Dieses Problem macht es notwendig, nach dem Matching der einzelnen Partitionen Duplikate in der Menge der gefundenen Mapping-Elemente zu eliminieren. Als Duplikate werden dabei alle Beziehungen betrachtet, bei denen die beiden in Beziehung gesetzten Konzepte gleich sind, unabhängig von Beziehungstyp und Stärkemaß. Zur Auflösung der Duplikate werden folgende Regeln definiert[7]:

1. Als Stärkemaß der Beziehung wird das arithmetische Mittel aller Duplikate angenommen.
2. Existiert in der Menge der Duplikate ein Duplikat mit dem Beziehungstyp „gleich“, so wird dieser als Typ der Beziehung angenommen.
3. Existieren in der Menge der Duplikate sowohl der Beziehungstyp „allgemeiner“ als auch der Beziehungstyp „weniger allgemein“, so wird ebenfalls der Beziehungstyp „gleich“ angenommen.
4. Falls 2 und 3 nicht zutreffen: Existiert in der Menge der Beziehungstyp „allgemeiner“, so wird dieser als Beziehungstyp angenommen.
5. Falls 2 und 3 nicht zutreffen: Existiert in der Menge der Beziehungstyp „weniger allgemein“, so wird dieser als Beziehungstyp angenommen.
6. Falls 2–5 nicht zutreffen, so kann in der Menge der Duplikate nur noch der Beziehungstyp „disjunkt“ auftreten; dieser wird dann als Beziehungstyp angenommen.

Regel 3 gründet sich auf die Gesetzmäßigkeit, dass aus $A \subseteq B$ und $B \subseteq A$ folgt, dass $A = B$ gelten muss. Dies entspricht auch der Semantik der Subklassenbeziehung in RDF-S (vgl. Hitzler u. a., 2008, S. 72). Die anderen Regeln sind nicht motiviert, sondern stellen lediglich eine mögliche Form der Duplikatauflösung dar. Hier wären auch andere Formen denkbar: so könnte in Regel 1 z.B. auch das Maximum oder das Minimum aller Stärkemaße verwendet werden, und dem Beziehungstyp „disjunkt“ könnte Vorrang vor den Beziehungstypen „allgemeiner“ und „weniger allgemein“ gegeben werden. Theoretisch fundierte Untersuchungen dazu, wie diese Regeln am sinnvollsten zu formulieren sind, existieren derzeit nicht.

## 4.3. Ergebnisse

Der entwickelte Prototyp wurde unter zwei Aspekten getestet: der Skalierbarkeit und der Ergebnisqualität. Da derzeit für große Ontologien keine Referenzmappings vorliegen, musste die Ergebnisqualität anhand von Paaren kleinerer Ontologien untersucht werden.

### 4.3.1. Skalierbarkeit

Zum Test der Skalierbarkeit wurden dieselben Testontologien und dieselbe Testumgebung wie in den in Kapitel 3.6.1 beschriebenen Tests verwendet. Um exemplarisch die Skalierbarkeit des Systems zu demonstrieren, wurden das Baseline-Verfahren mit Partitionsgröße 5.000 und der INRIA-Matching-Algorithmus verwendet.

[7] Dabei wird davon ausgegangen, dass es, wie in Kapitel 3.1.4 definiert, die Beziehungstypen „äquivalent“, “allgemeiner“, „weniger allgemein“ und „disjunkt“ gibt.

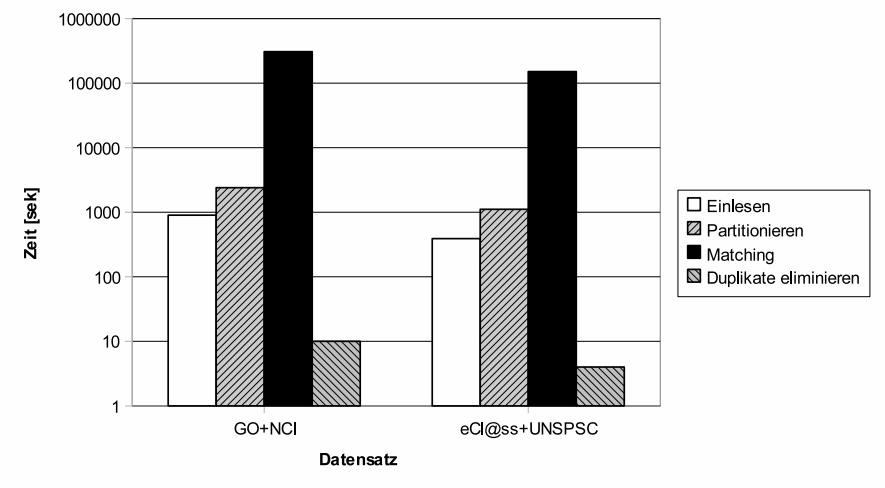



**Abbildung 4.6.:** Laufzeitverhalten des Prototypen bei großen Ontologien

Der Prototyp konnte die großen Ontologien mit dem zugewiesenen Hauptspeicher (1 MB) verarbeiten. Da die Speicherkomplexität prinzipiell nur von den Größen der einzelnen Partitionen abhängt (da nie mehr als zwei Partitionen gleichzeitig verarbeitet werden und nie Zwischenergebnisse von mehr als einem Paar von Partitionen im Speicher gehalten werden), ist davon auszugehen, dass auch noch größere Ontologien problemlos verarbeitet werden können.

Abbildung 4.6 zeigt, welcher Anteil der Gesamtlaufzeit auf die einzelnen Schritte des Systems entfällt. Da die Zeitachse logarithmisch aufgebaut ist, wird deutlich, dass der bei weitem größte Teil für das paarweise Matching der Partitionen aufgewendet wird[8].

### 4.3.2. Ergebnisqualität

Grundsätzlich gilt bei der Betrachtung der Ergebnisqualität, dass ein elementbasiertes Matching-Verfahren, sofern es mit einem abdeckenden Partitionierungsverfahren kombiniert wird, genauso gute Ergebnisse liefert wie auf den unpartitionierten Daten. Daher ist es insbesondere interessant, die Ergebnisqualität beim Einsatz strukturbasierter Matching-Verfahren zu untersuchen.

Die Ergebnisqualität des Prototypen wurde anhand von sechs Paaren von kleineren Beispielontologien getestet, für die ein Referenzmapping vorliegt. Diese Ontologien sind im Anhang A dargestellt.

Es wurden fünf Partitionierungsalgorithmen getestet: der Baseline-Algorithmus mit Partitionsgrößen von 250 und 500 Statements, der Islands-Algorithmus mit Größen von 50 und 100 Klassen sowie der $\varepsilon$-Connections-Algorithmus. Jeder der fünf Partitionierungs-Algorithmen wurde mit jeweils drei Matching-Systemen, dem INRIA-, dem FOAM- und dem CROSI-CMS-Matching-System, kombiniert.

[8]Die Laufzeitergebnisse sind nur als Näherungswerte zu verstehen; sie können bei Reproduktion der Versuche unterschiedlicher Systemkonfiguration schwanken. Dagegen sind die im folgenden geschilderten qualitativen Ergebnisse stets reproduzierbar.

Darüber hinaus wurden die Matching-Systeme auch auf den unpartitionierten Daten ausgeführt, um zu vergleichen, wie viel schlechter die Ergebnisse auf den partitionierten Daten sind. Der mittlere F-Measure-Wert beträgt bei INRIA 0.56, bei FOAM 0.84. Für das Matching-System CROSI CMS wurden drei strukturbasierte Algorithmen, `Structure`, `StructurePlus` und `HierarchyDisSim`, getestet. Der F-Measure-Wert für `HierarchyDisSim` liegt im Mittel bei 0.05, die Werte der anderen beiden Algorithmen noch deutlich darunter. Aufgrund dieser schlechten Ergebnisse wurden keine weiteren Tests mit dem Matching-System CROSI CMS durchgeführt; dieses Matching-System wird im folgenden nicht berücksichtigt.

Der $\varepsilon$-Connections-Algorithmus konnte nur bei zwei von sechs Testdatensätzen sinnvolle Ergebnisse liefern: bei zwei Datensätzen konnte jeweils eine Ontologie nicht partitioniert werden, bei zwei weiteren wurde nur jeweils eine Partition erzeugt. Aus diesem Grund wird der $\varepsilon$-Connections-Algorithmus bei den Ergebnissen und den Optimierungsansätzen nachfolgend nicht berücksichtigt.

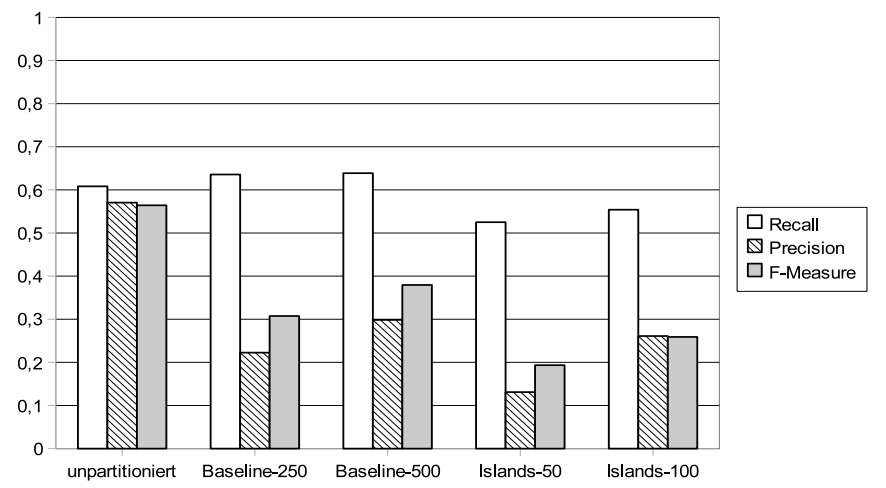



**Abbildung 4.7.:** Testergebnisse mit dem Matching-System INRIA

Abbildung 4.7 zeigt die Ergebnisse, die der Prototyp mit dem Matching-System INRIA und den verschiedenen Partitionierungsalgorithmen liefert, gemittelt über alle Testdatensätze[9]. Es werden jeweils Precision, Recall und F-Measure dargestellt.

Man kann erkennen, dass der Precision-Wert für die partitionierten Daten gegenüber dem Ergebnis auf den unpartitionierten Daten deutlich sinkt. Dagegen ist der Recall-Wert im Wesentlichen gleichbleibend; zum Teil wird sogar ein leicht verbesserter Recall-Wert auf den partitionierten Daten erzielt. Beide Effekte sind damit zu erklären, dass die Ergebnismenge bei unpartitionierten Daten deutlich größer ist als bei partitionierten. Dies führt einerseits zu einer hohen Anzahl an false positives, andererseits auch zu einer leicht erhöhten Anzahl an true positives. Der etwas verringerte Recall-Wert bei den beiden Varianten des Islands-Algorithmus ist damit zu erklären, dass der Islands-Algorithmus keine vollständige Partitionierung erzeugt; daher können einige Mapping-Elemente auf den partitionierten Daten nicht gefunden werden.

[9]Der Testfall People+Pets ist beim Algorithmus Islands-100 nicht berücksichtigt, da hier beide Ontologien aufgrund der geringen Größe der T-Boxen in je eine Partition zerlegt wurden.

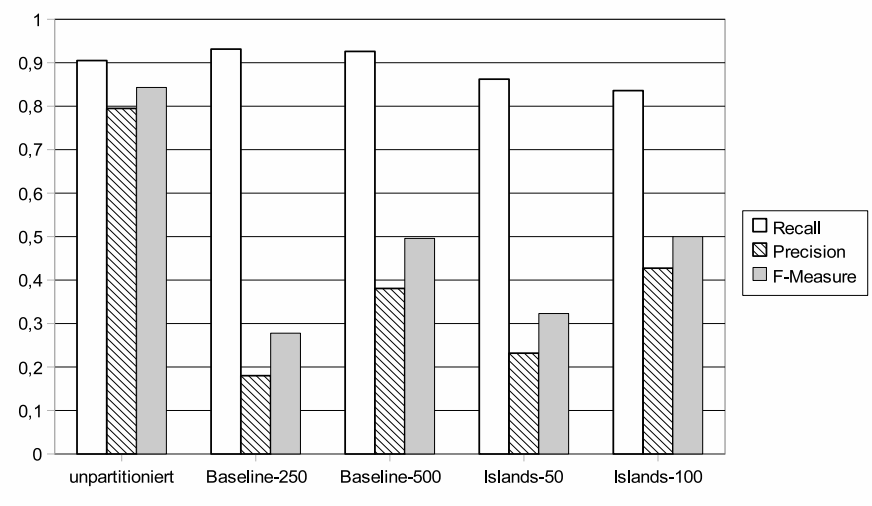



**Abbildung 4.8.:** Testergebnisse mit dem Matching-System FOAM

Abbildung 4.8 zeigt die entsprechenden Daten für das Matching-System FOAM. Hier ist ein ähnlicher Effekt bezüglich Recall und Precision zu beobachten. In beiden Fällen wird der jeweils höchste Recall-Wert mit dem Baseline-Verfahren erzielt, da dieses Verfahren eine abdeckende Partitionierung erzeugt. Den höchsten F-Measure-Wert erzielt man bei beiden Matching-Systemen mit dem Partitionierungs-Algorithmus Baseline-500, wobei bei FOAM mit Islands-100 ein vergleichbar gutes Ergebnis erzielt wird.

Im günstigsten Fall erreicht man mit INRIA im Mittel 68%, mit FOAM im Mittel 60% des F-Measure-Wertes für unpartitionierte Daten. Insgesamt sind die Ergebnisse des Matching-Systems FOAM – sowohl auf den partitionierten wie auch auf den unpartitionierten Daten – besser als die des Matching-Systems INRIA.

## 4.4. Optimierungen

Die Ergebnisse zeigen zwei Hauptprobleme: zum einen nimmt das paarweise Matching der Partitionen viel Zeit in Anspruch, zum anderen bleiben die erzielten Werte für Precision und F-Measure weit hinter denen zurück, die auf unpartitionierten Daten erreicht werden können. In diesem Abschnitt werden drei Optimierungen des Systems vorgestellt, mit denen diese Probleme gelöst oder zumindest verringert werden können. Der erste zielt auf eine Verbesserung des Laufzeitverhaltens, die anderen beiden auf eine Optimierung der Ergebnisqualität.

### 4.4.1. Verteiltes Matching

Im vorangegangenen Abschnitt wurde gezeigt, dass der größte Teil der Laufzeit für das paarweise Matching der Partitionen verbraucht wird. Durch Verteilung des Matching-Prozesses auf verschiedene Rechner lässt sich diese Zeit verringern.

#### 4.4.1.1. Motivation

Das Matching eines Paares von Partitionen ist ein Prozess, der unabhängig vom Matching eines anderen Paares durchgeführt werden kann. Daher ist es möglich, das Matching der Paare auch auf verschiedenen Rechnern durchzuführen. Ein möglicher Weg hierzu ist das Verteilen des Rechenprozesses in einem Peer-To-Peer(P2P)-Netzwerk. Dies hat den Vorteil, dass keine vorab bestimmten Client- und Serverrechner installiert werden müssen, vielmehr kann jeder Rechner als sogenannter *Peer* gleichzeitig die Rolle von Client und Server übernehmen. Jeder Peer kann Paare von Partitionen zum Matching an andere Rechner vergeben und selbst das Matching „fremder“ Paare für andere Peers durchführen. In einem P2P-Netzwerk können, auch über die Grenzen lokaler Netzwerke hinweg, verschiedenartige Rechner zusammengeschaltet werden, wie in Abbildung 4.9 dargestellt.

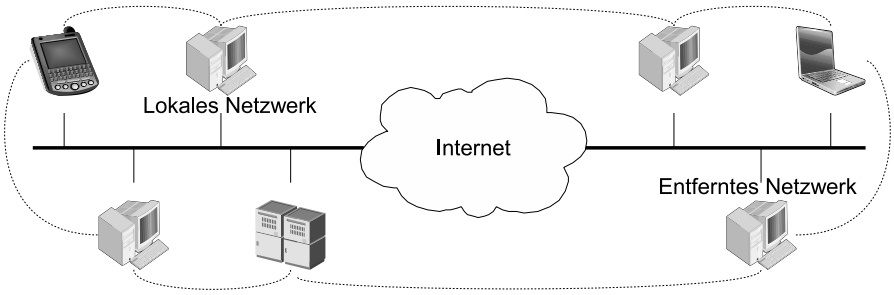



**Abbildung 4.9.:** P2P-Netz

Ein Framework, das die Entwicklung von P2P-basierten Anwendungen mit Java ermöglicht, ist *JXTA* (Sun Microsystems, 2007). Es enthält Funktionen zum Auffinden verbundener Rechner, zum Verschicken von Nachrichten an andere Rechner und vieles mehr. Auf der Basis von JXTA sind zahlreiche Dienste entwickelt worden, von denen zwei weitere in diesem Prototypen genutzt werden: Mit *JXTA SOAP* können Web Services über das P2P-Netz gesucht und aufgerufen werden (java.net, 2007b), und mit *JXTA CMS* können Dateien zwischen Peer-Rechnern ausgetauscht werden (java.net, 2007a).

#### 4.4.1.2. Systemerweiterung

Abbildung 4.10 zeigt, wie diese Dienste kombiniert werden. Der JXTA-CMS-Dienst wird genutzt, um einerseits die Partitionen über das Netz an diejenigen Peers zu verteilen, die das Matching der Partitionen durchführen, andererseits, um die gefundenen Teilmappings zwischen den Partitionen wieder an den Peer zurückzuschicken, der das Matching in Auftrag gegeben hat. Der JXTA-SOAP-Dienst wird verwendet, um entfernte Matcher zu starten.

Zur Implementierung wurden verschiedene Erweiterungen vorgenommen: die Klasse `P2PServiceAdapter` kapselt sämtliche Zugriffe über das JXTA-Framework und stellt dem Matching-System dessen Dienste zur Verfügung. Zu den Diensten gehören

- die Bereitstellung eines Matchers im P2P-Netz,
- die Entdeckung von Matchern im P2P-Netz,
- der Aufruf eines Matchers über das P2P-Netz,

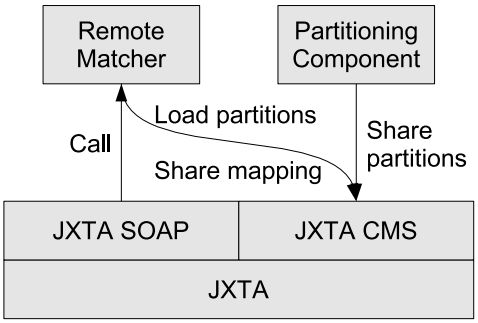



**Abbildung 4.10.:** Interaktion der Komponenten mit JXTA

- die Bereitstellung einer Datei im P2P-Netz und
- das Herunterladen einer Datei aus dem P2P-Netz.

Die Integration der entfernten Matcher erfolgt einerseits über eine spezielle Subklasse von `AbstractMatchingPartitionScheduler`, die neben den eigentlichen Aufgaben periodisch nach neuen Matchern sucht, und einer Subklasse von `AbstractPartitionMatcher`, die als Stellvertreter für ein entferntes Matching-System steht und den Aufruf als Web Service über das P2P-Netzwerk an die Klasse `P2PServiceAdapter` weitergibt.

Um über das P2P-Netz auf die Partitionen zuzugreifen, wird eine Subklasse von `AbstractOntologyStorageFacade` implementiert. Sie besitzt also dieselbe Schnittstelle wie ein lokaler Speicher für Partitionen. Diese Klasse fordert über die Klasse `P2PServiceAdapter` Partitionen über das P2P-Netz an und führt eine lokale Zwischenspeicherung durch.

Abbildung 4.11 zeigt den Prozess im Detail (Partitionierung und Filtern der Ergebnisse sind zur Vereinfachung nicht dargestellt). Von der Klasse `RemotePartitionMatcherProxy` können mehrere Objekte existieren, die jeweils für einen Peer stehen, und parallel unterschiedliche Paare von Partitionen verarbeiten; sobald ein Objekt frei wird, also das Teilergebnis des entfernten Matching-Systems zurück erhalten wurde, wird ihm vom Scheduler ein neues Paar von Partitionen zugeteilt.

#### 4.4.1.3. Ergebnisse

Die Verteilung in einem P2P-Netz wurde mit den beiden Paaren von großen Ontologien, die auch in Kapitel 3.6.1 verwendet wurden, getestet. Für jedes dieser Paare wurde ein Matching mit 4, 8 und 12 Peers durchgeführt, wobei stets ein zusätzlicher Peer genutzt wurde, der die Koordination übernimmt. Abbildung 4.12 zeigt das Laufzeitverhalten für das Ontologiepaar eCl@ss+UNSPSC, Abbildung 4.13 für das Paar NCI+GO[10]. Die Ontologien und die Konfiguration des Testsystems sind im Anhang A beschrieben.

Aus den Testergebnissen ist ersichtlich, dass durch Verdopplung der Anzahl der für den Matching-Prozess genutzten Peers eine Geschwindigkeitssteigerung um den Faktor 1,3 bis 1,4 erreicht werden kann. Dass aus einer Verdopplung der Anzahl der Peers keine Verdopplung der

[10] Auch hier gilt, dass die Testergebnisse lediglich als Näherungswerte, nicht als exakt reproduzierbare Werte zu verstehen sind.

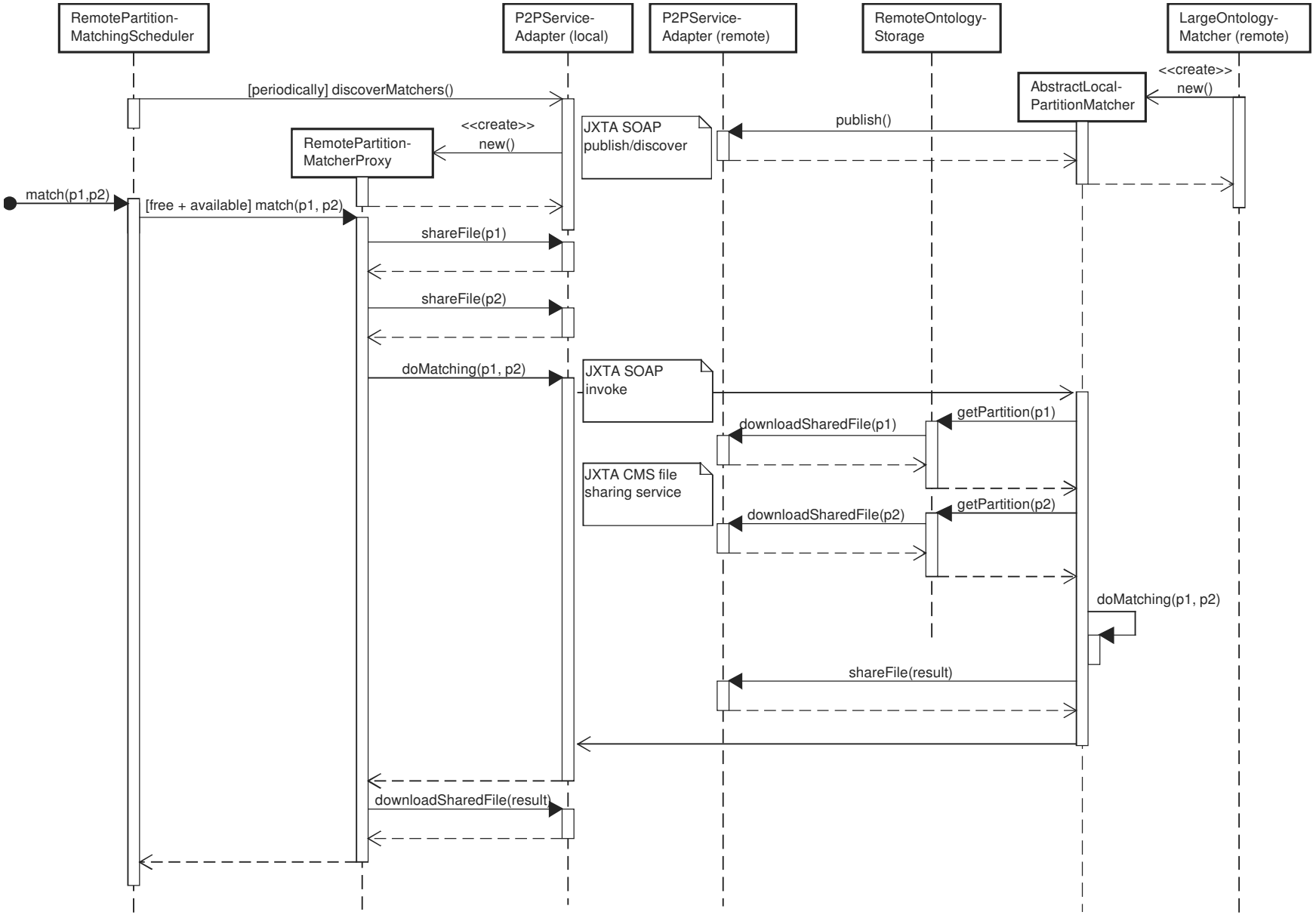



**Abbildung 4.11.:** Ablauf eines verteilten Matching-Prozesses

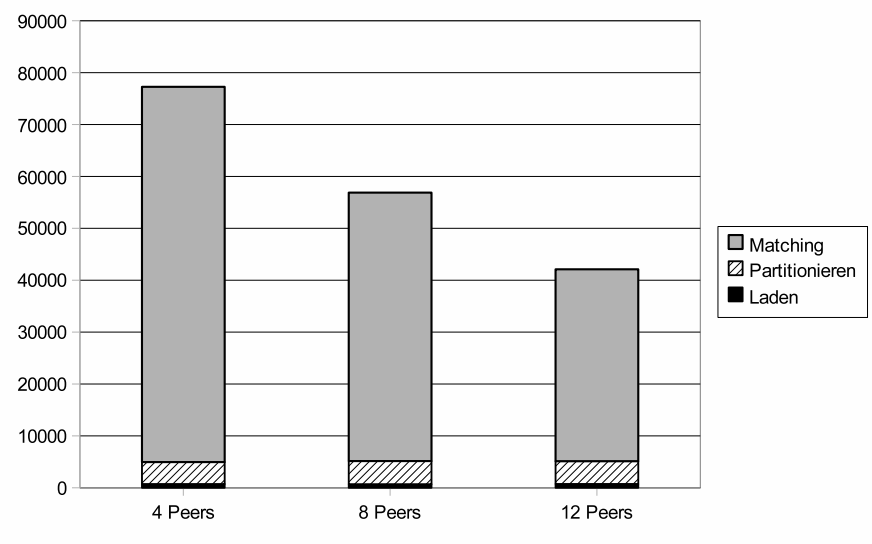



**Abbildung 4.12.:** Laufzeit des Matchings im P2P-Netz, Datensatz eCl@ss+UNSPSC

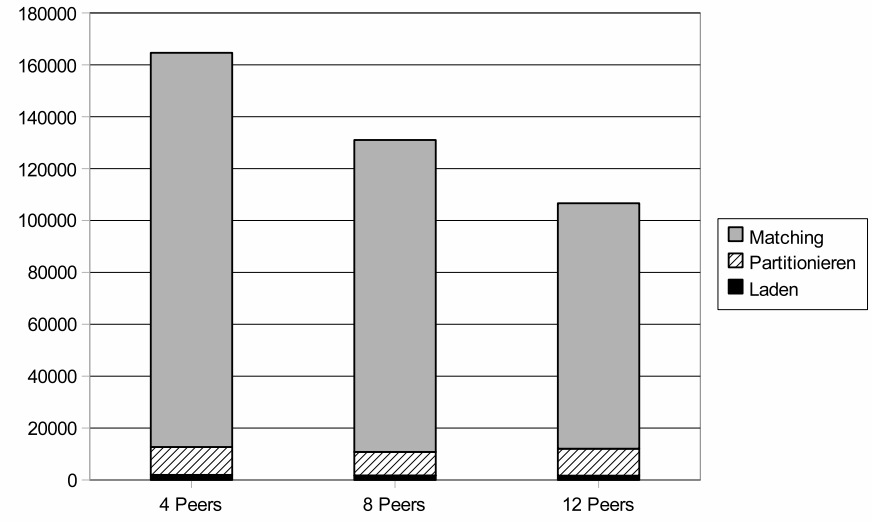



**Abbildung 4.13.:** Laufzeit des Matchings im P2P-Netz, Datensatz NCI+GO

Geschwindigkeitssteigerung folgt, liegt daran, dass zusätzlicher Aufwand für die Übertragung der Partitionen und Teilergebnisse über das P2P-Netz betrieben werden muss. Außerdem kann eine bestimmte Zeit verstreichen, bis sich alle Peers miteinander vernetzt haben – in den durchgeführten Versuchen vergingen teils bis zu acht Stunden, bevor ein Peer vom Master genutzt werden konnte. Dies bedeutet, dass bis zu diesem Zeitpunkt mit weniger Peers gearbeitet wird, und dass folglich insgesamt bei Verdopplung der Peer-Anzahl keine Halbierung der Rechenzeit erreicht werden kann.

Neben der bisweilen großen Zeitspanne bis zur kompletten Vernetzung aller Peers haben sich während der Tests weitere Probleme mit der Stabilität der eingesetzten JXTA-Implementierung gezeigt: so weisen z.B. sowohl die SOAP- als auch die File-Sharing-Erweiterung nur eine geringe Fehlertoleranz bei Nichterreichbarkeit einzelner Peers auf. Da diese Nichterreichbarkeit

jedoch in einem P2P-Netz ein realistisches Szenario ist, etwa weil Nutzer ihre Rechner herunterfahren oder weil Netzwerkverbindungen ausfallen, mussten hier zusätzliche Maßnahmen getroffen werden, um ein stabiles System zu gewährleisten.

### 4.4.2. Verwendung überlappender Partitionen

Die Verwendung überlappender Partitionen dient zur Verbesserung der Ergebnisqualität. Dabei werden bei der Partitionierung bewusst Redundanzen erzeugt, indem bestimmte Aussagen der Ausgangsontologien in mehrere Partitionen kopiert werden.

#### 4.4.2.1. Motivation

Die Vorhersage, ob ein Element zu einem anderen in Beziehung steht, ist einem Matching-Werkzeug umso besser möglich, je mehr Information über dieses Element zur Verfügung steht. Strukturbasierte Werkzeuge (siehe Kapitel 3.3.2) beziehen dabei Informationen über die Nachbarelemente mit ein. Gerade am Rand einer Partition steht diese Information jedoch nur eingeschränkt zur Verfügung.

Um diesen Nachteil auszugleichen, kann man Partitionen auch überlappend erzeugen. Dazu wird ein Rand um jede Partition gezogen, also alle Elemente, mit denen ein Element der Partition in Beziehung steht, mit in die Partition aufgenommen. Führt man diesen Schritt einmal durch, so erhält man einen Rand der Breite 1, beim nächsten Mal einen Rand der Breite 2, und so weiter. Abbildung 4.14 zeigt zwei überlappende Partitionen, jeweils mit Rand der Breite 1 und 2.

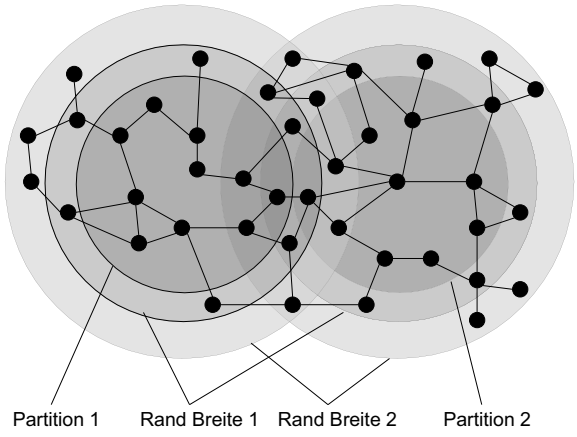



**Abbildung 4.14.:** Zwei überlappende Partitionen

Ist $\delta$ der maximale Verzweigungsgrad der Ontologie (also die maximale Anzahl von Verbindungen, die ein Element mit einem anderen eingeht), $N$ die Größe einer Partition, dann hat eine Partition mit einem Rand der Breite $b$ die Größe $O\left(\delta^b \cdot N\right)$, das heißt, die Größe der Partition wächst im ungünstigsten Fall exponentiell mit der Breite des Randes (auch wenn dieses Wachstum gedämpft wird, je stärker sich die Partition an die Gesamtontologie annähert). Daher sind nur kleine Ränder praktikabel.

#### 4.4.2.2. Systemerweiterung

Zur Implementierung dieses Ansatzes sind zwei Schritte notwendig: zunächst wird eine Methode benötigt, die einen Rand um eine Partition erzeugt. Der Algorithmus in Pseudocode ist in Quelltextbeispiel 4.3 gezeigt. Wie auch bei der Implementierung des Baseline-Algorithmus werden anonyme Knoten gesondert behandelt, und es werden Typ-Statements für jedes Konzept in die Partition kopiert. Zur Markierung eines Elements als Randelement wird ein `owl:AnnotationProperty` verwendet. Zur Erzeugung von Rändern der Breite $b$ wird die Methode $b$-mal aufgerufen.

```
void buildFringe(Ontology partition) {
  for each concept x in partition {

    for each statement s with s.subject==x {
      if(partition does not contain s) {
        add s to partition;
        if(s.subject is an anonymous node)
          add all statements connected to s.subject to partition
        add type statements for all subjects and objects added to partition
        mark all added concepts as fringe elements
    }

    for each statement s with s.object==x {
      if(partition does not contain s) {
        add s to partition;
        if(s.subject is an anonymous node)
          add all statements connected to s.subject to partition
        add type statements for all subjects and objects added to partition
        mark all added concepts as fringe elements
    }

  }
}
```

**Quelltextbeispiel 4.3:** Erzeugen eines Randes

Um das Ergebnis des Matchings nicht zu verfälschen, werden die Markierungen der Randelemente vor dem Matching der Partitionen entfernt; anderenfalls könnte ein strukturbasiertes Matching-Werkzeug zwei Elemente, die keine inhaltliche Beziehung haben, aufgrund des gemeinsamen `AnnotationProperty` als ähnlich einstufen.

Nach dem Matching wird ein zusätzlicher Filterungsschritt durchgeführt: alle Mapping-Elemente zwischen Konzepten, von denen mindestens eines als Randelement gekennzeichnet ist, müssen aus der Ergebnismenge entfernt werden. Hierzu wird eine neue Klasse implementiert, die `AbstractMappingFilter` erweitert. Würde man diesen Schritt auslassen, so erhielte man zwar für Elemente aus dem Kern der Partition bessere Ergebnisse, jedoch für diejenigen auf dem Rand schlechtere. Das liegt daran, dass die Randelemente der Partition hinzugefügt wurden, um das Mapping der ursprünglichen Elemente der Partition verlässlicher zu machen.

Werden diese erweiterten Partitionen an ein Matching-Werkzeug übergeben, so finden diese meist auch Beziehungen zwischen den Randelementen. Aus demselben Grund, wie sie hinzugefügt wurden, sind diese Beziehungen zwischen den Randelementen wenig aussagekräftig: über sie liegt zu wenig Information über benachbarte Elemente vor, da sie eben auf dem Rand

der Partition liegen. Da bei einer ursprünglich abdeckenden Partitionierung die Randelemente auch mindestens einmal verarbeitet werden, wenn sie im Kern einer Partition liegen, können die Beziehungen zwischen Randelementen verworfen werden, ohne damit den Recall-Wert zu verschlechtern.

#### 4.4.2.3. Ergebnisse

Im Rahmen dieser Arbeit wurde nur die Verwendung eines Randes der Breite 1 getestet. Der Grund hierfür ist neben dem oben erörterten Komplexitätsproblem die vergleichsweise geringe Größe der Testontologien: für einen größeren Rand würden sich die Partitionen sehr schnell den Originalontologien annähern, damit würde das Testergebnis zu stark verzerrt.

Abbildung 4.15 zeigt, über alle Testfälle gemittelt, das Verhältnis zwischen Partitionen ohne Rand und Partitionen mit Rand der Breite 1. Dabei wird deutlich, dass bei dem Baseline-Verfahren das Verhältnis zwischen Rand und eigentlicher Partition deutlich größer ist als bei den anderen Verfahren. Dies ist darauf zurückzuführen, dass ein gut arbeitendes Partitionierungsverfahren solche Partitionen erzeugt, bei dem zusammenhängende Konzepte möglichst in derselben Partition liegen. Daher ist bei einem guten Partitionierungsverfahren die Wahrscheinlichkeit, dass der Nachbar eines Konzeptes in derselben Partition liegt und nicht in den Rand aufgenommen werden muss, höher als bei einem schlechten Partitionierungsverfahren.

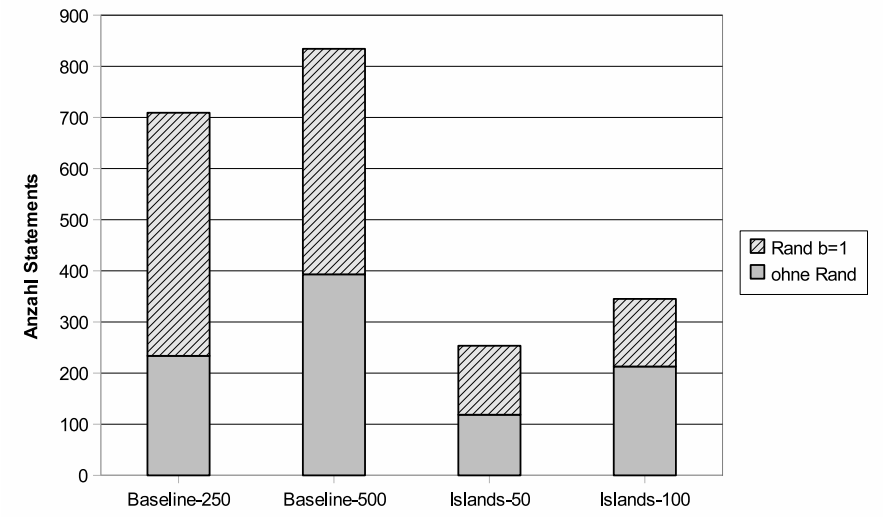



**Abbildung 4.15.:** Partitionsgrößen mit und ohne Rand

Die Partitionsgröße ist ein unmittelbarer Indikator für die Speicherkomplexität des gesamten Verfahrens: je größer die Partitionen, desto mehr Speicher benötigt das Matching-Verfahren, um sie zu verarbeiten. Auch die gesamte Laufzeitkomplexität hängt von der Partitionsgröße ab: Je größer die Überlappung der Partitionen, desto höher ist die Laufzeit, da das paarweise Matching der Partitionen entsprechend aufwändiger wird. Dies ist darauf zurückzuführen, dass jedes Element, das in mehreren Partitionen enthalten ist, auch in mehreren Matching-Prozessen berücksichtigt werden muss.

Abbildung 4.16 zeigt am Beispiel des Matching-Werkzeugs FOAM, wie sich die Hinzunahme des Randes auf die Laufzeit auswirkt. Es ist bei allen Testfällen zu beobachten, dass die

Laufzeit bei Hinzunahme des Randes etwa um den Faktor 2 bis 4 ansteigt – die Größe des Faktors scheint jedoch nicht nur von der Partitionsgröße abzuhängen. Dieser Faktor kann auch beim Matching mit dem Werkzeug INRIA festgestellt werden, dort allerdings ist die größte relative Zunahme bei dem Partitionierungsalgorithmus Baseline-500 und die geringste bei dem Partitionierungsalgorithmus Islands-100 festzustellen.

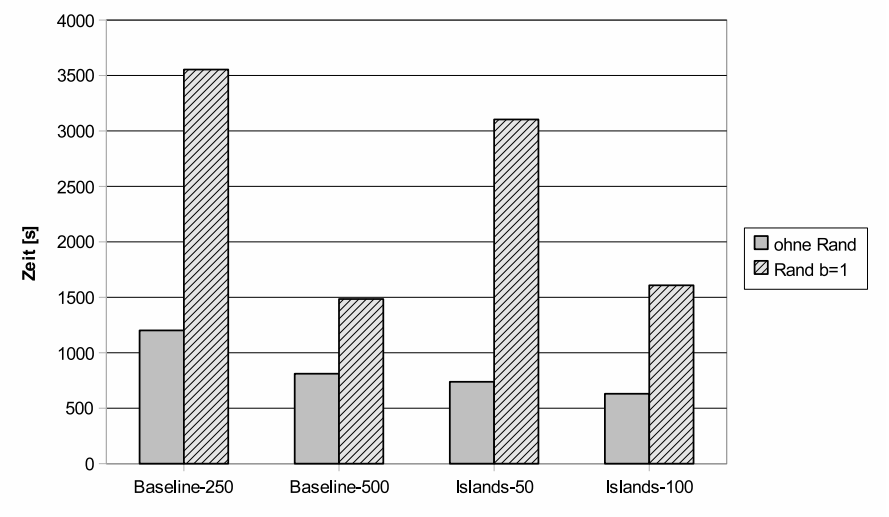



**Abbildung 4.16.:** Laufzeit des Verfahrens mit und ohne Rand

Die Ergebnisqualität kann durch die Verwendung überlappender Partitionen stark zunehmen. Abbildung 4.17 zeigt, wiederum gemittelt über alle Testfälle, die Ergebnisse mit und ohne Rand für das Matching-Werkzeug INRIA, Abbildung 4.18 die entsprechenden Ergebnisse für das Matching-Werkzeug FOAM. Gegenüber der Version ohne Rand ergibt sich in allen Fällen ein deutlich höherer Wert für Precision und F-Measure bei weitgehend gleichbleibendem Wert für Recall.

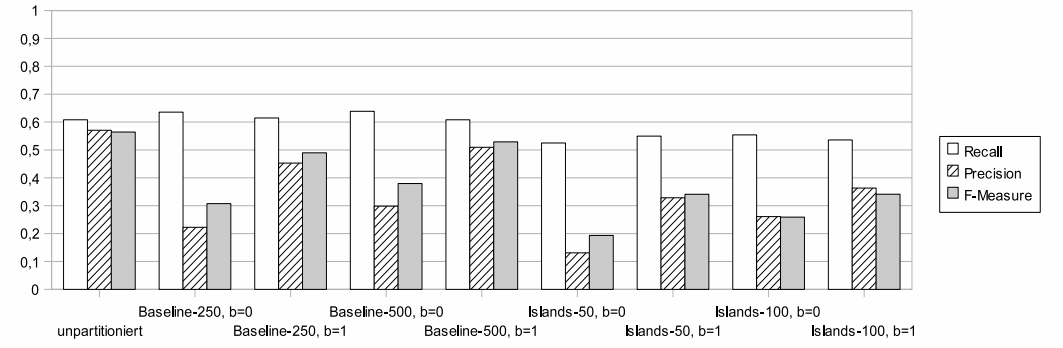



**Abbildung 4.17.:** Ergebnisqualität mit und ohne Rand, Matching-System INRIA

Mit dem Matching-System INRIA erreicht man im besten Fall, mit dem Partitionierungs-Algorithmus Baseline-500, im Mittel 93.8% des Wertes für F-Measure, der auf den unpartitionierten Daten erzielt wird. Mit dem Matching-System FOAM werden im Mittel 92.3% des Wertes für F-Measure auf den unpartitionierten Daten erreicht.

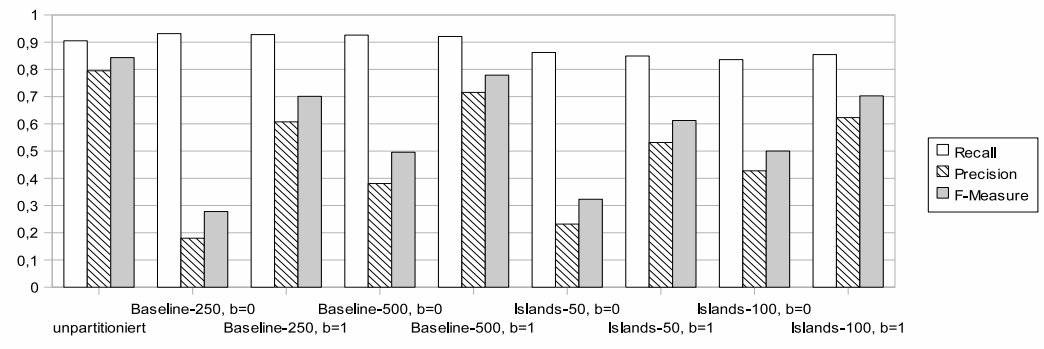



**Abbildung 4.18.:** Ergebnisqualität mit und ohne Rand, Matching-System FOAM

### 4.4.3. Filtern der Ergebnisse

Die Ergebnisse zeigen, dass der schlechte Precision-Wert auf eine große Ergebnismenge im Vergleich zu unpartitionierten Daten und damit auf eine hohe Anzahl an false positives zurückzuführen ist. Daher ist ein weiterer möglicher Ansatz zur Optimierung der Ergebnisqualität der Einsatz eines Filters auf den Ergebnissen.

#### 4.4.3.1. Motivation

Die untersuchten Matching-Werkzeuge liefern für jedes gefundene Mapping-Element ein Konfidenz-Maß (vgl. Kapitel 3.1.4.1). Betrachtet man den mittleren Wert für dieses Maß, jeweils für true positives und false positives, so kann man, wie in Tabelle 4.1 gezeigt, feststellen, dass der Wert für true positives signifikant höher ist als für false positives.

| | unpartitioniert | | Baseline-250 | | Baseline-500 | | Islands-50 | | Islands-100 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *c(tp)* | *c(fp)* | *c(tp)* | *c(fp)* | *c(tp)* | *c(fp)* | *c(tp)* | *c(fp)* | *c(tp)* | *c(fp)* |
| INRIA | 0.979 | 0.746 | 0.971 | 0.691 | 0.977 | 0.712 | 0.973 | 0.651 | 0.979 | 0.739 |
| FOAM | 0.985 | 0.470 | 0.978 | 0.188 | 0.981 | 0.254 | 0.975 | 0.166 | 0.986 | 0.309 |

**Tabelle 4.1.:** Mittlerer Konfidenzwert für true positives und false positives

Diese Beobachtung legt die Vermutung nahe, dass durch das Aussortieren von Mapping-Elementen mit geringem Konfidenzwert das Gesamtergebnis besser wird. Daher wird ein Filter hinzugefügt, der nur Mapping-Elemente mit einem Konfidenzwert über einer vordefinierten Schranke $\tau$ durchlässt.

#### 4.4.3.2. Systemerweiterung

Der Filter wird in der Klasse `ThresholdMappingFilter` umgesetzt, die das Interface `MappingFilter` implementiert. Jedes Teilergebnis wird zusätzlich mit diesem Filter verarbeitet, bevor es gespeichert wird. Weitere Änderungen sind nicht erforderlich.

#### 4.4.3.3. Ergebnisse

Die Ergebnisqualität hängt entscheidend von der Wahl des Wertes für $\tau$ ab. Daher wurden für jeden Partitionierungsalgorithmus und jedes Matching-System alle Werte zwischen 0 und 1 mit einer Intervallbreite von 0.01 getestet und damit, gemittelt über alle Testfälle, optimale Werte für $\tau$ ermittelt. Dabei bezieht sich „optimal“ stets auf den Wert für F-Measure[11]. Die ermittelten optimalen Werte für $\tau$ sind, zusammen mit dem erzielten F-Measure, in Tabelle 4.2 dargestellt. Die Prozentwerte in Klammern bezeichnen dabei das Verhältnis des F-Measure-Wertes zum F-Measure-Wert auf den unpartitionierten Daten.

| | INRIA | | | FOAM | | |
|---|---|---|---|---|---|---|
| | $\tau_{opt}$ | F-Measure | | $\tau_{opt}$ | F-Measure | |
| unpartioniert | 0.908 | 0.698 | | 0.696 | 0.895 | |
| Baseline-250 | 0.950 | 0.690 | (98.9%) | 0.983 | 0.804 | (89.8%) |
| Baseline-500 | 0.950 | 0.691 | (99.0%) | 0.948 | 0.845 | (94.4%) |
| Islands-50 | 0.952 | 0.649 | (93.0%) | 0.973 | 0.854 | (95.4%) |
| Islands-100 | 0.906 | 0.657 | (94.1%) | 0.906 | 0.847 | (94.6%) |

**Tabelle 4.2.:** Optimale Werte für den Filter-Grenzwert $\tau$

Abbildung 4.19 zeigt exemplarisch das Verhalten für den Partitionierungsalgorithmus Baseline-500 und das Matching-System INRIA, gemittelt über alle Testfälle: auf der x-Achse ist der Wert für $\tau$ abgetragen, auf der y-Achse die Werte für Recall, Precision und F-Measure bei Einsatz eines Filters mit diesem Grenzwert.

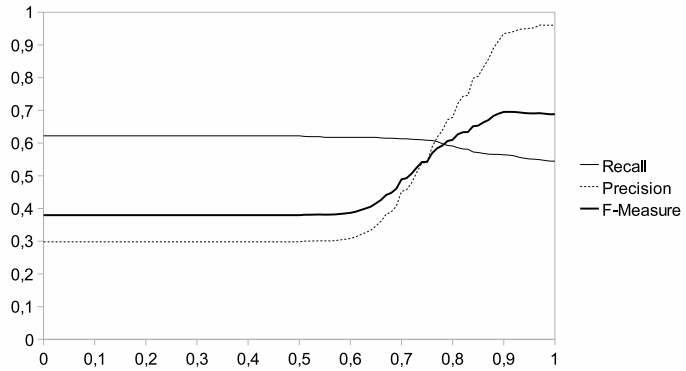



**Abbildung 4.19.:** Recall, Precision und F-Measure in Abhängigkeit von $\tau$, am Beispiel von Baseline-500 und INRIA, gemittelt über alle Testfälle

Es ist zu beobachten, dass ein deutlicher Qualitätszugewinn erzielt werden kann: Die Ergebnisse für $\tau = 0$ entsprechen den Ergebnissen ohne Anwendung eines Filters; gegenüber diesen wird durch den Filter eine deutliche Verbesserung des F-Measure-Wertes erzielt. Da dieser Wert über dem Wert für F-Measure auf den unpartitionierten Daten liegt, ist es sinnvoll, auch für diese einen Filter einzusetzen, um die Ergebnisse vergleichbar zu machen. Daher wurde auch für das Matching der unpartitionierten Daten analog ein optimaler Wert für $\tau$ ermittelt.

[11] Trivialerweise liegt der optimale Wert für $\tau$ bezüglich Recall bei 0, bezüglich Precision bei 1.

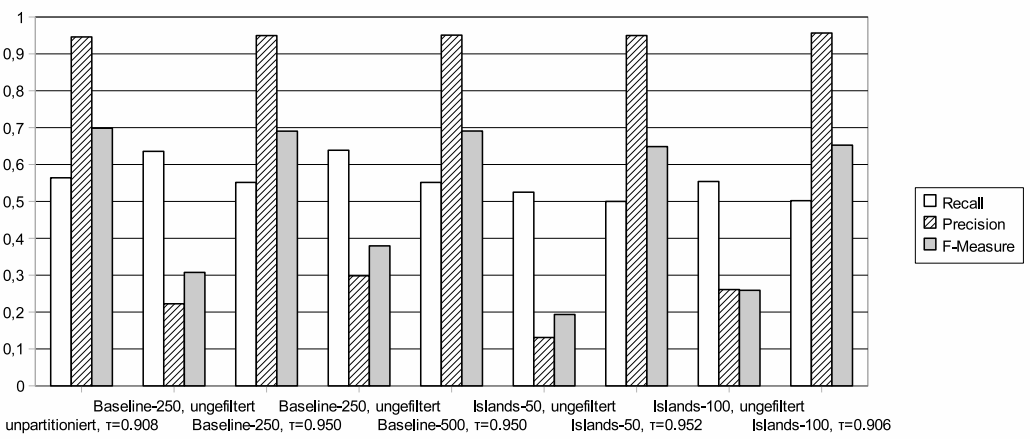



**Abbildung 4.20.:** Ergebnisqualität mit und ohne Filter, Matching-System INRIA

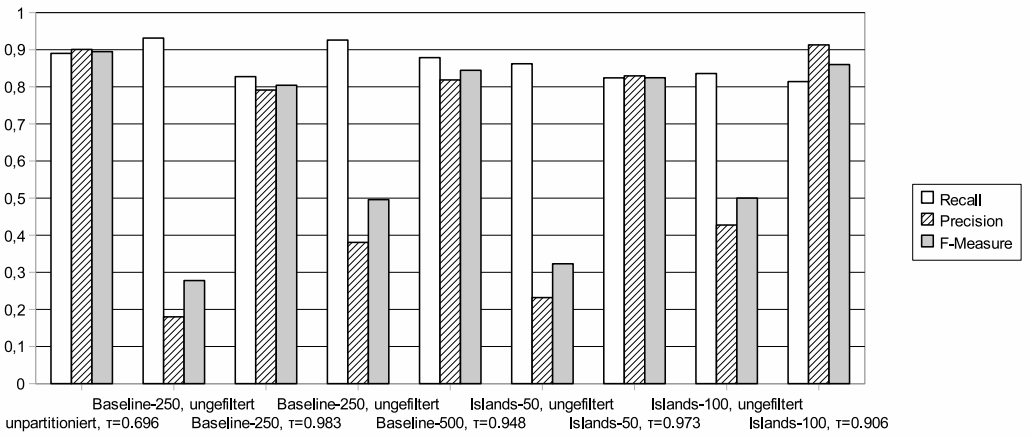



**Abbildung 4.21.:** Ergebnisqualität mit und ohne Filter, Matching-System FOAM

Abbildung 4.20 zeigt jeweils die ungefilterten und die optimal gefilterten Ergebnisse auf den partitionierten Daten im Verhältnis zu den (ebenfalls optimal gefilterten) Ergebnissen auf den unpartitionierten Daten mit dem Matching-System INRIA, Abbildung 4.21 die entsprechenden Ergebnisse mit dem Matching-System FOAM.

Es kann beobachtet werden, dass der ohne Filter auftretende Effekt des stark verringerten Precision-Wertes durch den Einsatz des Filters nahezu vollständig ausgeglichen werden kann: der Precision-Wert ist bei den gefilterten Ergebnissen über alle Partitionierungsverfahren hinweg nahezu konstant. Bei INRIA werden im besten Fall ca. 99,5%, bei FOAM ca. 96,1% des F-Measure-Wertes auf unpartitionierten Daten erreicht.

Darüber hinaus ist zu beobachten, dass die Unterschiede zwischen Baseline-250 und Baseline-500 sowie zwischen Islands-50 und Islands-100 eher gering sind. Dies zeigt, dass die Partitionsgröße bei Einsatz eines Filters nur von nachrangiger Bedeutung ist.

Anders als die Nutzung überlappender Partitionen erzeugt das Filtern der Ergebnisse kaum zusätzliche Laufzeiten; der Aufwand ist linear in der Anzahl der Zwischenergebnisse und im Vergleich zum eigentlichen Matching vernachlässigbar gering. Die Speicherkomplexität erhöht

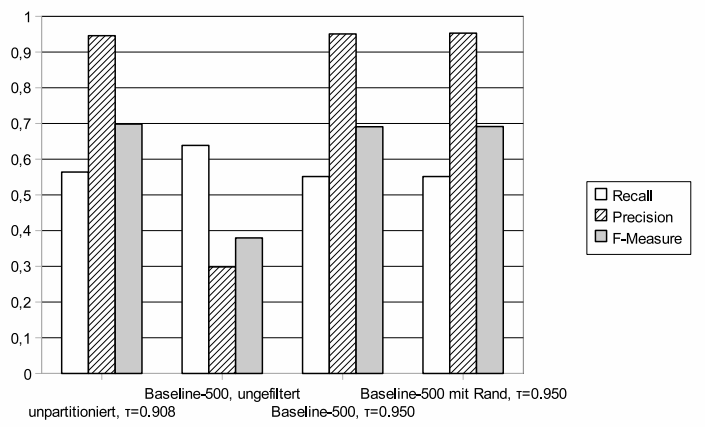



**Abbildung 4.22.:** Kombination von Rand und Filter, Matching-System INRIA

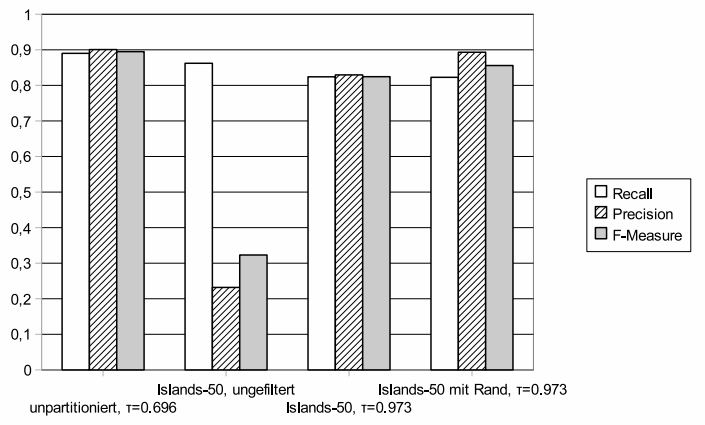



**Abbildung 4.23.:** Kombination von Rand und Filter, Matching-System FOAM

sich ebenfalls nicht, da für das Filtern keine zusätzlichen temporären Variablen o.ä. benötigt werden.

### 4.4.4. Kombination von überlappenden Partitionen und Filtern

Dass sowohl mit überlappenden Partitionen als auch mit Filtern der Mapping-Elemente eine qualitatitve Verbesserung erreicht werden kann, legt die Hypothese nahe, dass durch Kombination beider Methoden eine noch höhere Ergebnisqualität erzielt werden kann.

Abbildung 4.22 zeigt Recall, Precision und F-Measure in der Kombination von überlappenden Partitionen und einem Filter für das Matching-System INRIA, aus Gründen der Übersichtlichkeit nur für den besten Fall der Partitionierung mit Baseline-500, Abbildung 4.23 die entsprechenden Daten für das Matching-System FOAM.

Es ist nahezu kein Unterschied zwischen überlappenden und nicht überlappenden Partitionen erkennbar. Tatsächlich verbessert sich das Ergebnis auf gefilterten Daten durch Hinzunahme eines Randes der Breite eins bei FOAM etwa drei Prozentpunkte, bei INRIA nur um etwa einen Prozentpunkt verbessert.

## 4.5. Zusammenfassung

In diesem Kapitel wurde ein Prototyp vorgestellt, mit dem bestehende, nicht-skalierbare Matching-Systeme auch auf große Ontologien angewendet werden können.

Für die Partitionierung wurden unterschiedliche Algorithmen eingesetzt. Dabei hat sich herausgestellt, dass die Wahl eines Algorithmus, der, wie der Baseline-Algorithmus, eine abdeckende Partitionierung erzeugt, auf jeden Fall vorzuziehen ist, selbst wenn er keine sinnvoll zusammenhängenden Einheiten als Partitionen erzeugt.

Es wurden verschiedene Ansätze zur Optimierung der Ergebnisqualität diskutiert. Da das Filtern der Ergebnisse, im Gegensatz zur Benutzung überlappender Partitionen, zum einen die besseren Ergebnisse, zum anderen nur wenig zusätzliche Laufzeit- und Speicherkomplexität erzeugt, ist dieser Ansatz vorzuziehen. Die Kombination beider Ansätze bringt nur eine minimale Verbesserung der Ergebnisqualität; daher kann der Optimierungsansatz der überlappenden Partitionen in den meisten Fällen verworfen werden.

Als Optimum hat sich die Nutzung des Baseline-Verfahrens in Kombination mit dem Matching-Werkzeug FOAM und einem grenzwertbasierten Filter herausgestellt.

# 5. Ausblick

Ontology Matching ist ein zentrales Problem in vielen Anwendungen, in denen Ontologien zum Einsatz kommen. Mit dem entwickelten Prototyp wurde gezeigt, dass sich bestehende, nicht-skalierbare Matching-Werkzeuge mit nur geringem Qualitätsverlust auch auf große Ontologien anwenden lassen, und dass dieses Verfahren auch auf Rechnern mit moderater Hardwareausstattung lauffähig ist.

## 5.1. Ausbaumöglichkeiten des Prototypen

Der in Kapitel 4 vorgestellte Prototyp kann auf verschiedene Arten weiterentwickelt werden. Je nach Anwendungsfall und spezifischen Anforderungen sind dabei unterschiedliche Erweiterungen denkbar.

Die Ergebnisse, die von Matching-Werkzeugen geliefert werden, sind meist nicht perfekt. Daher ist es in vielen Fällen nötig, die Ergebnisse manuell nachzubearbeiten. Ein möglicher Weg der Nachbearbeitung, der den menschlichen Aufwand minimal hält, führt über implizite Bewertungen: wenn ein Nutzer ein Mapping-Element verwendet, wird dieses automatisch gut bewertet; andere Mapping-Elemente, die in diesem Zusammenhang ebenfalls möglich gewesen wären, aber vom Nutzer nicht verwendet wurden, erhalten eine schlechte Bewertung. Auf diese Weise ist es möglich, längerfristig korrekte und inkorrekte Ergebnisse zu trennen (Paulheim u. a., 2007, S. 47 ff.).

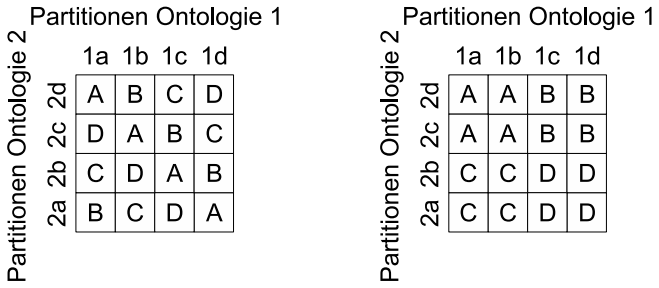
Mochol u. a. (2006, S. 37 ff.) haben gezeigt, dass, abhängig von bestimmten Charakteristika von Ontologien, manche Matching-Werkzeuge ein besseres Ergebnis als andere liefern, und dass es nicht *das* beste Matching-Werkzeug für alle Ontologien gibt. Das von ihnen vorgestellte System schlägt dynamisch ein passendes Matching-Werkzeug vor. Da der in dieser Arbeit vorgestellte Prototyp bereits die Möglichkeit vorsieht, unterschiedliche Matching-Werkzeuge zu nutzen, könnte die Kombination mit diesem Ansatz die Ergebnisqualität noch weiter verbessern. Außerdem wäre zu untersuchen, ob eine ähnliche Abhängigkeit auch für die Wahl des Partitionierungsverfahrens existiert.

Bislang wurden nur Matching-Werkzeuge eingesetzt, die einfache Mappings liefern. Dennoch ist es ohne weiteres denkbar, mit dem in dieser Arbeit vorgestellten Ansatz auch Werkzeuge zu nutzen, die komplexe Mappings erstellen. Interessant wäre hierbei insbesondere die Frage, ob der Baseline-Algorithmus auch bei diesen Werkzeugen die besten Ergebnisse liefert, da zum Auffinden von komplexen Mappings alle durch ein Mapping-Element in Beziehung gesetzten Elemente in derselben Partition enthalten sein müssen.

Auf die Optimierung der Laufzeit beim Matching großer Ontologien wurde in dieser Arbeit nur ein geringer Fokus gelegt. Dennoch kann auch die Geschwindigkeit, in der ein Mapping gefunden wird, relevant sein (Ehrig, 2007b, S. 39). Neben dem Einsatz möglichst schneller

Matching-Verfahren für das paarweise Matching der Partitionen und der Verteilung des Rechenprozesses ist es auch denkbar, zunächst mit einem schnellen Verfahren abzuschätzen, welche Paare von Partitionen überhaupt gemeinsame Konzepte enthalten, und diese dann bevorzugt zu verarbeiten, um schneller zu einem brauchbaren Teilergebnis zu kommen. Hier ist ein zweistufiger Ansatz denkbar: die Abschätzung erfolgt mit einem schnellen, aber qualitativ schlechteren Matching-Verfahren, das eigentliche Matching der Partitionen mit einem qualitativ guten, aber langsameren Verfahren.

Auch für die Verteilung im P2P-Netz lassen sich Laufzeitoptimierungen erzielen. Wenn ein Client $n$ Partitionen matcht, müssen im schlechtesten Fall $2n$, im besten Fall dagegen nur $2\sqrt{n}$ Partitionen übertragen werden. Abbildung 5.1 zeigt diesen Zusammenhang anhand von zwei Ontologien mit je vier Partitionen und vier Clients A, B, C und D. Im ersten Fall müssen alle Partitionen an alle Clients verteilt werden, im zweiten Fall nur die Hälfte aller Partitionen. Auch allgemeine Mechanismen zur Optimierung verteilten Rechnens, wie die Zuteilung besonders komplexer Operationen an schnellere Rechner und das Einführen von Zwischenknoten, die eine weitere Verteilung von Operationen durchführen, sind denkbar (siehe z.B. Verbeke u. a., 2002). Außerdem sollte untersucht werden, ob mit anderen Frameworks zum verteilten Rechnen in P2P-Netzen höhrere Geschwindigkeitszuwächse erreicht werden können.




(a) schlechtester Fall (b) optimaler Fall

**Abbildung 5.1.:** Unterschiedliche Verteilung von Partitionen

Sind sensible Daten, zum Beispiel Unternehmenswissen, in den Ontologien enthalten, so ist der P2P-Ansatz zunächst nicht geeignet, da auf diese Weise Außenstehende Zugriff auf Teile der Daten erhalten können, sofern das P2P-Netz über Unternehmensgrenzen hinaus ausgedehnt wird. Mitra u. a. (2005) haben einen Ansatz entwickelt, bei dem Ontologien verschlüsselt, die verschlüsselten Ontologien von einem Matching-System verarbeitet und das gefundene Mapping anschließend wieder entschlüsselt wird. Mit Hilfe dieses Ansatzes wäre ein verteiltes Matching auch von sensiblen Daten denkbar.

## 5.2. Verwandte Probleme

In dieser Arbeit wurde ein partitionenbasiertes Matching-Verfahren vorgestellt. Es wurden bereits einige solcher Verfahren vorgestellt (siehe z.B. Wang u. a., 2006, S. 99 ff. oder Hu u. a., 2006b, S. 72 ff.), allerdings nutzen alle diese Ansätze eigene Matching-Algorithmen, die Wiederverwendung bestehender Matching-Systeme wurde bislang nicht untersucht. Außerdem wurde

in Kapitel 3.6.1 gezeigt, dass auch die existierenden Werkzeuge, die einen partitionenbasierten Ansatz verfolgen, nicht beliebig skalierbar sind.

Der in dieser Arbeit vorgestellte Prototyp erstellt ein komplettes Mapping der Eingabeontologien. Ein anderer Ansatz im Umgang mit großen Ontologien ist das gezielte Suchen nach Mapping-Elementen, ohne dabei die gesamten Ontologien zueinander in Beziehung zu setzen. Solche Verfahren wurden von Besana u. a. (2005) und Lopez u. a. (2006) vorgestellt.

Skalierbarkeit ist auch in anderen Forschungsgebieten des Semantic Web ein wichtiges Thema. So ist zum Beispiel die Speicherung und vor allem der effiziente Zugriff auf gespeicherte Ontologien, d.h., die performante Beantwortung von Abfragen auf großen Ontologien, ein essentielles Problem bei der Entwicklung skalierbarer Persistenzsysteme für Ontologien (Liu und Hu, 2005).

Auch Reasoning mit großen Ontologien ist eine Herausforderung (Zhang u. a., 2006, S. 209 f., sowie Wandelt und Möller, 2007). Ähnlich wie die in dieser Arbeit untersuchten Matching-Systeme sind auch Reasoning-Systeme meist nicht skalierbar. Auch für die Skalierung von Reasoning-Systemen gibt es erste Ansätze, die auf Partitionierung der Ontologien basieren (Guo und Heflin, 2006, S. 47 ff.).

Werden große Ontologien entwickelt, so müssen diese für Entwickler und Benutzer visualisiert und navigierbar gemacht werden. Werkzeuge, die eine für den Nutzer einfache Betrachtung und Bearbeitung großer Ontologien ermöglichen, sind daher ebenfalls ein Forschungsgegenstand in der Semantic-Web-Forschung (Tzitzikas und Hainaut, 2006, S. 99 ff.). Einen umfangreichen Überblick über Verfahren und Werkzeuge geben Katifori u. a. (2007). In diesem Zusammenhang sind auch skalierbare Visualisierungsmöglichkeiten für Mappings zwischen Ontologien zu erforschen (Euzenat und Shvaiko, 2007, S. 273).

## 5.3. Fazit

Skalierbarkeit wird in zukünftigen Semantic-Web-Anwendungen eine größere Rolle spielen als bisher. Der entwickelte Prototyp ist ein Baustein für Systeme, die mit großen Ontologien umgehen müssen. Es wurde gezeigt, dass mit den entwickelten Methoden ein Matching großer Ontologien möglich ist, und dass die Ergebnisse, gemessen an den Möglichkeiten derzeit existierender Ontology-Matching-Werkzeuge, von sehr hoher Qualität sind.

# A. Testsystem und Testdaten

Die Tests, die in dieser Arbeit zitiert wurden, wurden in einer Testumgebung durchgeführt, die wie folgt konfiguriert ist:

| | |
|---|---|
| Prozessor | Intel XEON 3.2 GHz |
| Speicher (RAM) | 1 GB |
| Betriebssystem | Microsoft Windows Server 2003 SP 1 |
| Java-Version | 1.5 |
| Zugesicherter Speicher für Java | 1 GB |
| MySQL-Version | 4.0.24-nt |

**Tabelle A.1.:** Konfiguration der Testumgebung

Für die Untersuchung des verteilten Rechnens im P2P-Netz wurde folgende Rechnerkonfiguration eingesetzt:

| | |
|---|---|
| Prozessor | Intel Pentium 2.0 GHz |
| Speicher (RAM) | 1 GB |
| Betriebssystem | Microsoft Windows XP |
| Java-Version | 1.5 |
| Zugesicherter Speicher für Java | 512 MB |
| MySQL-Version | 5.0.51a |
| Netzwerkverbindung | 100 MBit LAN |

**Tabelle A.2.:** Konfiguration der Testumgebung für das P2P-Netz

In dieser Arbeit wurden verschiedene Testdatensätze verwendet, um die vorgestellten Systeme und die eigene Entwicklung zu testen. Für den Test der Skalierbarkeit wurden die in Tabelle A.3 aufgeführten großen Ontologien eingesetzt.

Um die Skalierbarkeit der Systeme zu testen, wurde versucht, ein Mapping zwischen den beiden ersten und den beiden letzten Paare zu finden.

Zur inhaltlichen Analyse des implementierten Systems wurden verschiedene Paare von Ontologien, für die ein Referenzmapping existiert, verwendet. Diese sind in Tabelle A.4 dargestellt.

Die ersten fünf Beispielpaare stammen von der Internetseite des Matching-Systems FOAM (Ehrig, 2007a). Die drei Ontologiepaare Russia 1+2, Russia A+B und Russia C+D wurden von Studenten erstellt und enthalten Konzepte, die aus Reiseinformationsseiten über Russland stammen. Die Beziehungen zwischen den Konzepten wurden von den Erstellern manuell zugewiesenDie beiden Tourismus-Ontologien sind als Seminararbeiten einer Universität entstanden; sie enthalten Klassen und Relationen, die Tourismusinformationen zum Bundesland

| Ontologie | Anzahl Tripel | Größe (XML-Datei) | Inhalt |
|---|---|---|---|
| NCI Thesaurus | 543497 | ca. 32 MB | Enthält Begriffsdefinitionen für Krankheiten, Medikamente, Diagnosen und Behandlungen, für Anatomie, Organismen, Gene und Proteine (Maryland Information and Network Dynamics Lab Semantic Web Agents Project, 2007). |
| Gene Ontology | 464842 | ca. 42 MB | Wird aus verschiedenen medizinischen Datenbank gespeist und enthält Begriffs- und Synonymdefinitionen für Zellkomponenten, biologische Prozesse und molekulare Funktionen (Gene Ontology Consortium, 2007). |
| eCl@ss | 374796 | ca. 25 MB | Enthält eine Taxonomie von handelbaren Produkten und deren Eigenschaften (Hepp, 2006). |
| UNSPSC | 82957 | ca. 6 MB | Enthält eine Taxonomie von handelbaren Produkten ohne Eigenschaften (Klein, 2002). |

**Tabelle A.3.:** Große Ontologien

Mecklenburg-Vorpommern enthalten. Die beiden People+Pets-Ontologien enthalten Informationen zu Haustieren und deren Besitzern. (Ehrig, 2007b, S. 89 u. S. 211 ff.). Die beiden Sport-Ontologien beinhalten Konzepte zur Beschreibung von Sportereignissen und von Fußballspielen, sie enthalten einige gemeinsame, aber auch viele unterschiedliche Konzepte.

| Name | Anzahl Tripel | Größe Referenzmapping | Inhalt |
|---|---|---|---|
| Russia 1+2 | 1408 / 1678 | 223 | Beschreibung von Russland |
| Russia A+B | 1408 / 912 | 285 | Beschreibung von Russland |
| Russia C+D | 940 / 957 | 215 | Beschreibung von Russland |
| Sport/Soccer | 1928 / 853 | 150 | Beschreibung von Sportereignissen |
| Tourism | 1485 / 1918 | 226 | Beschreibung von Tourismus in Mecklenburg-Vorpommern |
| People+Pets | 641 / 606 | 93 | Informationen zu Haustieren und Menschen |

**Tabelle A.4.:** Ontologiepaare mit Referenzmapping

# B. Testergebnisse

In diesem Kapitel werden die Testergebnisse, die im Text nur auszugsweise und zusammengefasst dargestellt wurden, im Einzelnen protokolliert.

Tabelle B.1 zeigt die Partitionsgrößen, die sich bei der Verarbeitung der Testontologien mit den jeweiligen Partitionierungsalgorithmen ergeben haben. In der ersten Teilspalte ist jeweils die Zahl der vom Partitionierungsalgorithmus erzeugten Fragmente, in der zweiten und dritten deren durchschnittliche Größe ohne Rand und mit Rand der Breite 1 aufgeführt. Die Annotationen, die Randelemente kennzeichnen, sind dabei nicht berücksichtigt. Mit dem $\varepsilon$-Connections-Algorithmus konnten nicht alle Testontologien partitioniert werden.

Tabelle B.2 zeigt die Ergebnisse, die mit dem INRIA-Matching-System zum einen ohne Partitionierung, zum anderen mit zwei Varianten des Baseline-Algorithmus erzielt wurden. Tabelle B.3 zeigt die entsprechenden Daten für zwei Varianten des Islands-Algorithmus und den $\varepsilon$-Connections-Algorithmus. Dabei sind jeweils die Anzahl der ingesamt gefundenen Mappings (*abs*), die Anzahl der True Positives (*tp*), der False Positives (*fp*) und der False Negatives (*fn*) sowie der mittlere Konfidenzwert der True Positives (*c(tp)*) und der False Positives (*c(fp)*) aufgezeichnet. Für den $\varepsilon$-Connections-Algorithmus sind nur bei denjenigen Testfällen Ergebnisse aufgeführt, in denen der Algorithmus beide Ontologien partitioneren konnten und für mindestens eine Ontologie mehr als Partition erzeugt hat.

Die Tabellen B.4 und B.5 zeigen die entsprechenden Daten für das System FOAM.

In Kapitel 4.4.2 wurde die Optimierung mit Hilfe von überlappenden Partitionen dargestellt. In den Tabellen B.6 und B.7 sind die gleichen Daten für überlappende Partitionen mit Randbreite 1 für das Matching-System INRIA eingetragen, in den Tabellen B.8 und B.9 die entsprechenden Daten für das System FOAM. Die unpartitionierten Daten tauchen hier nicht auf, da es in diesem Fall naturgemäß keine überlappenden Partitionen geben kann.

In Kapitel 4.4.3 wurde die Optimierung mit Hilfe eines Filters dargestellt. Die Tabellen B.10 und B.11 zeigen die optimalen Werte für die Filterschranke $\tau$. In den Tabellen B.12 und B.13 sind die Daten für den jeweils optimalen Wert von $\tau$ mit em Matching-System INRIA und verschiedenen Partitionierungsverfahren eingetragen, in den Tabellen B.14 und B.15 die entsprechenden Daten für das System FOAM.

In Kapitel 4.4.4 wurde die Kombination von überlappenden Partitionen und Filtern diskutiert. In den Tabellen B.16 und B.17 sind die gleichen Daten für überlappende Partitionen mit Randbreite 1 für das Matching-System INRIA eingetragen, in den Tabellen B.18 und B.19 die entsprechenden Daten für das System FOAM. Die unpartitionierten Daten sind hier wieder nicht enthalten, da hier keine überlappenden Partitionen möglich sind.

Drei Algorithmen des Matching-Werkzeugs CROSI wurden nur auf unpartitionierten Daten getestet. Die Ergebnisse sind in den Tabellen B.20, B.21 und B.22 dargestellt.

| Algorithmus | Baseline-250 | | | Baseline-500 | | | Islands-50 | | | Islands-100 | | | $\varepsilon$-Connections | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Randgröße | | 0 | 1 | | 0 | 1 | | 0 | 1 | | 0 | 1 | | 0 | 1 |
| Russia 1 | 8 | 240.1 | 814.9 | 4 | 433.5 | 1028.0 | 14 | 103.6 | 265.3 | 5 | 280.0 | 465.2 | – | | |
| Russia 2 | 10 | 234.2 | 896.3 | 5 | 427.0 | 1122.0 | 15 | 120.9 | 349.9 | 8 | 219.9 | 431.9 | 1 | 1202.0 | 1681.0 |
| Russia A | 8 | 240.1 | 814.9 | 4 | 433.5 | 1028.0 | 14 | 103.6 | 265.2 | 5 | 280.0 | 465.2 | – | | |
| Russia B | 5 | 235.6 | 628.2 | 3 | 349.7 | 637.3 | 4 | 207.5 | 335.3 | 2 | 412.5 | 479.0 | 1 | 637.0 | 915.0 |
| Russia C | 5 | 235.0 | 626.4 | 3 | 349.0 | 626.4 | 9 | 96.2 | 212.3 | 5 | 169.8 | 318.4 | 1 | 671.0 | 943.0 |
| Russia D | 5 | 238.4 | 626.6 | 3 | 359.0 | 624.6 | 6 | 144.2 | 311.3 | 1 | 855.0 | 938.0 | 1 | 685.0 | 960.0 |
| Sport | 9 | 242.6 | 628.1 | 5 | 422.2 | 878.8 | 22 | 90.8 | 145.1 | 10 | 195.9 | 279.3 | 18 | 46.9 | 129.3 |
| Soccer | 4 | 233.0 | 473.8 | 2 | 446.0 | 938.0 | 18 | 46.1 | 113.0 | 6 | 129.8 | 165.8 | 6 | 67.6 | 128.2 |
| Tourism A | 8 | 224.2 | 820.4 | 4 | 418.0 | 1042.5 | 23 | 55.9 | 137.2 | 11 | 115.6 | 240.9 | 1 | 1051.0 | 1488.0 |
| Tourism B | 10 | 238.6 | 960.4 | 5 | 449.8 | 1300.6 | 36 | 50.2 | 133.2 | 16 | 112.9 | 213.4 | 1 | 1347.0 | 1921.0 |
| People+Pets A | 3 | 227.0 | 518.3 | 2 | 325.5 | 539.0 | 5 | 218.4 | 406.6 | 1 | 615.0 | 626.0 | 3 | 152.3 | 364.0 |
| People+Pets B | 3 | 215.7 | 499.0 | 2 | 305.5 | 508.5 | 4 | 185.5 | 367.3 | 1 | 580.0 | 591.0 | 3 | 143.0 | 344.0 |

**Tabelle B.1.:** Partitionsanzahl und -durchschnittsgröße bei den Testontologien

| | **unpartitioniert** | | | | | | **Baseline-250** | | | | | | **Baseline-500** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 176 | 57 | 119 | 104 | 0.970 | 0.727 | 728 | 60 | 668 | 101 | 0.967 | 0.684 | 418 | 58 | 360 | 103 | 0.979 | 0.699 |
| Russia A+B | 219 | 160 | 59 | 125 | 1.000 | 0.745 | 528 | 160 | 368 | 125 | 1.000 | 0.706 | 434 | 160 | 274 | 125 | 1.000 | 0.683 |
| Russia C+D | 171 | 134 | 37 | 81 | 1.000 | 0.747 | 418 | 134 | 284 | 81 | 1.000 | 0.709 | 325 | 134 | 191 | 81 | 1.000 | 0.715 |
| Sport/Soccer | 375 | 85 | 290 | 65 | 0.970 | 0.777 | 1104 | 98 | 1006 | 52 | 0.946 | 0.724 | 635 | 92 | 543 | 58 | 0.956 | 0.751 |
| Tourism | 390 | 191 | 199 | 35 | 0.957 | 0.755 | 2158 | 202 | 1956 | 24 | 0.944 | 0.670 | 1175 | 198 | 977 | 28 | 0.948 | 0.722 |
| People+Pets | 176 | 57 | 119 | 104 | 0.979 | 0.727 | 728 | 60 | 668 | 101 | 0.967 | 0.684 | 418 | 58 | 360 | 103 | 0.979 | 0.699 |

**Tabelle B.2.:** Ergebnisse mit INRIA, unpartitioniert und partitioniert mit Baseline-Algorithmus

| | Islands-50 | | | | | | Islands-100 | | | | | | $\varepsilon$-Connections | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 1810 | 58 | 1752 | 103 | 0.964 | 0.643 | 1076 | 57 | 1019 | 104 | 0.974 | 0.651 | – | | | | | |
| Russia A+B | 596 | 135 | 461 | 150 | 1.000 | 0.666 | 346 | 140 | 206 | 145 | 1.000 | 0.666 | – | | | | | |
| Russia C+D | 622 | 104 | 518 | 111 | 1.000 | 0.667 | 141 | 108 | 33 | 107 | 1.000 | 0.749 | – | | | | | |
| Sport/Soccer | 5380 | 96 | 5284 | 54 | 0.942 | 0.631 | 2059 | 88 | 1971 | 62 | 0.961 | 0.639 | 2208 | 95 | 2113 | 55 | 0.947 | 0.639 |
| Tourism | 10674 | 189 | 10485 | 37 | 0.950 | 0.627 | 4978 | 189 | 4789 | 37 | 0.948 | 0.637 | – | | | | | |
| People+Pets | 190 | 62 | 128 | 31 | 0.983 | 0.669 | 68 | 58 | 10 | 35 | 0.992 | 0.793 | 128 | 61 | 67 | 32 | 0.993 | 0.679 |

**Tabelle B.3.:** Ergebnisse mit INRIA, partitioniert mit Islands- und $\varepsilon$-Connections-Algorithmus

| | unpartitioniert | | | | | | Baseline-250 | | | | | | Baseline-500 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 222 | 125 | 97 | 36 | 0.977 | 0.476 | 3542 | 129 | 3413 | 32 | 0.971 | 0.170 | 1458 | 129 | 1329 | 32 | 0.966 | 0.225 |
| Russia A+B | 284 | 280 | 4 | 5 | 1.000 | 0.272 | 1334 | 284 | 1050 | 1 | 1.000 | 0.167 | 538 | 284 | 254 | 1 | 1.000 | 0.283 |
| Russia C+D | 233 | 212 | 21 | 3 | 1.000 | 0.663 | 963 | 212 | 751 | 3 | 1.000 | 0.207 | 401 | 212 | 189 | 3 | 1.000 | 0.340 |
| Sport/Soccer | 186 | 131 | 55 | 19 | 0.958 | 0.342 | 1624 | 137 | 1487 | 13 | 0.955 | 0.156 | 645 | 137 | 508 | 13 | 0.964 | 0.212 |
| Tourism | 296 | 190 | 106 | 36 | 0.987 | 0.368 | 4107 | 209 | 3898 | 17 | 0.959 | 0.118 | 1632 | 204 | 1428 | 22 | 0.960 | 0.177 |
| People+Pets | 93 | 90 | 3 | 3 | 0.997 | 0.700 | 187 | 89 | 98 | 4 | 0.983 | 0.308 | 111 | 89 | 22 | 4 | 0.997 | 0.287 |

**Tabelle B.4.:** Ergebnisse mit FOAM, unpartitioniert und partitioniert mit Baseline-Algorithmus

| | Islands-50 | | | | | | Islands-100 | | | | | | $\varepsilon$-Connections | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 1997 | 135 | 1862 | 26 | 0.943 | 0.135 | 652 | 128 | 524 | 33 | 0.974 | 0.193 | – | | | | | |
| Russia A+B | 704 | 259 | 445 | 26 | 1.000 | 0.130 | 332 | 264 | 68 | 21 | 1.000 | 0.239 | – | | | | | |
| Russia C+D | 677 | 182 | 495 | 33 | 1.000 | 0.151 | 234 | 186 | 48 | 29 | 1.000 | 0.406 | – | | | | | |
| Sport/Soccer | 2619 | 123 | 2496 | 27 | 0.976 | 0.118 | 696 | 121 | 575 | 29 | 0.974 | 0.217 | 864 | 126 | 738 | 24 | 0.991 | 0.180 |
| Tourism | 5834 | 187 | 5647 | 39 | 0.966 | 0.067 | 1721 | 184 | 1537 | 42 | 0.970 | 0.097 | – | | | | | |
| People+Pets | 140 | 86 | 54 | 7 | 0.996 | 0.392 | 88 | 85 | 3 | 8 | 0.997 | 0.700 | 128 | 84 | 44 | 9 | 0.999 | 0.302 |

**Tabelle B.5.:** Ergebnisse mit FOAM, partitioniert mit Islands- und $\varepsilon$-Connections-Algorithmus

| | Baseline-250 | | | | | | Baseline-500 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 264 | 57 | 207 | 104 | 0.979 | 0.717 | 220 | 57 | 163 | 104 | 0.979 | 0.722 |
| Russia A+B | 275 | 160 | 115 | 125 | 1.000 | 0.731 | 240 | 160 | 80 | 125 | 1.000 | 0.737 |
| Russia C+D | 206 | 134 | 72 | 81 | 1.000 | 0.733 | 189 | 134 | 55 | 81 | 1.000 | 0.735 |
| Sport/Soccer | 451 | 87 | 364 | 63 | 0.965 | 0.765 | 381 | 85 | 296 | 65 | 0.970 | 0.777 |
| Tourism | 698 | 192 | 506 | 34 | 0.957 | 0.738 | 494 | 191 | 303 | 35 | 0.958 | 0.749 |
| People+Pets | 79 | 65 | 14 | 28 | 0.993 | 0.750 | 80 | 65 | 15 | 28 | 0.993 | 0.727 |

**Tabelle B.6.:** Ergebnisse mit INRIA und überlappenden Partitionen, partitioniert mit Baseline-Algorithmus

| | Islands-50 | | | | | | Islands-100 | | | | | | $\varepsilon$-Connections | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 428 | 56 | 372 | 105 | 0.972 | 0.677 | 292 | 56 | 236 | 105 | 0.979 | 0.695 | – | | | | | |
| Russia A+B | 266 | 133 | 133 | 152 | 1.000 | 0.710 | 201 | 138 | 63 | 147 | 1.000 | 0.730 | – | | | | | |
| Russia C+D | 207 | 101 | 106 | 114 | 1.000 | 0.713 | 135 | 104 | 31 | 111 | 1.000 | 0.748 | – | | | | | |
| Sport/Soccer | 1481 | 88 | 1393 | 62 | 0.961 | 0.661 | 1427 | 87 | 1340 | 63 | 0.963 | 0.637 | 2208 | 95 | 2113 | 55 | 0.947 | 0.639 |
| Tourism | 3180 | 179 | 3001 | 47 | 0.963 | 0.653 | 1667 | 177 | 1490 | 49 | 0.967 | 0.661 | – | | | | | |
| People+Pets | 80 | 59 | 21 | 34 | 0.992 | 0.746 | 68 | 58 | 10 | 35 | 0.992 | 0.793 | 117 | 61 | 56 | 32 | 0.993 | 0.670 |

**Tabelle B.7.:** Ergebnisse mit INRIA und überlappenden Partitionen, partitioniert mit Islands- und $\varepsilon$-Connections-Algorithmus

| | Baseline-250 | | | | | | Baseline-500 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 477 | 136 | 341 | 25 | 0.970 | 0.497 | 363 | 132 | 231 | 29 | 0.966 | 0.547 |
| Russia A+B | 297 | 282 | 15 | 3 | 1.000 | 0.470 | 291 | 284 | 7 | 1 | 1.000 | 0.645 |
| Russia C+D | 249 | 211 | 38 | 4 | 1.000 | 0.551 | 236 | 212 | 24 | 3 | 1.000 | 0.645 |
| Sport/Soccer | 459 | 134 | 325 | 16 | 0.990 | 0.458 | 279 | 134 | 145 | 16 | 0.989 | 0.427 |
| Tourism | 527 | 199 | 328 | 27 | 0.979 | 0.370 | 363 | 195 | 168 | 31 | 0.990 | 0.490 |
| People+Pets | 102 | 91 | 11 | 2 | 0.995 | 0.351 | 95 | 90 | 5 | 3 | 0.997 | 0.638 |

**Tabelle B.8.:** Ergebnisse mit FOAM und überlappenden Partitionen, partitioniert mit Baseline-Algorithmus

| | Islands-50 | | | | | | Islands-100 | | | | | | $\varepsilon$-Connections | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 391 | 125 | 266 | 36 | 0.991 | 0.495 | 249 | 125 | 124 | 36 | 0.976 | 0.534 | – | | | | | |
| Russia A+B | 296 | 256 | 40 | 29 | 1.000 | 0.351 | 267 | 262 | 5 | 23 | 1.000 | 0.250 | – | | | | | |
| Russia C+D | 274 | 179 | 95 | 36 | 1.000 | 0.372 | 208 | 182 | 26 | 33 | 1.000 | 0.470 | – | | | | | |
| Sport/Soccer | 559 | 131 | 428 | 19 | 0.986 | 0.333 | 344 | 130 | 214 | 20 | 0.991 | 0.392 | 667 | 133 | 534 | 17 | 0.984 | 0.232 |
| Tourism | 893 | 181 | 712 | 45 | 0.991 | 0.306 | 411 | 181 | 230 | 45 | 0.990 | 0.323 | – | | | | | |
| People+Pets | 93 | 85 | 8 | 8 | 0.9963 | 0.554 | 87 | 82 | 5 | 11 | 0.997 | 0.817 | 95 | 86 | 9 | 7 | 0.997 | 0.485 |

**Tabelle B.9.:** Ergebnisse mit FOAM und überlappenden Partitionen, partitioniert mit Islands- und $\varepsilon$-Connections-Algorithmus

| **Testfall** | **unpartitioniert** | **Baseline-250** | **Baseline-500** | **Islands-50** | **Islands-100** | **$\varepsilon$-Connections** |
|---|---|---|---|---|---|---|
| Russia 1+2 | 0.975 | 0.975 | 0.975 | 0.975 | 0.975 | – |
| Russia A+B | 0.975 | 0.975 | 0.975 | 0.975 | 0.975 | – |
| Russia C+D | 0.935 | 0.975 | 0.975 | 0.975 | 0.975 | – |
| Sport/Soccer | 0.925 | 0.930 | 0.930 | 0.930 | 0.925 | 0.985 |
| Tourism | 0.900 | 0.900 | 0.900 | 0.910 | 0.910 | – |
| People+Pets | 0.735 | 0.945 | 0.945 | 0.945 | 0.740 | 0.945 |

**Tabelle B.10.:** Optimale Werte für $\tau$ beim Matching-System INRIA

| **Testfall** | **unpartitioniert** | **Baseline-250** | **Baseline-500** | **Islands-50** | **Islands-100** | **$\varepsilon$-Connections** |
|---|---|---|---|---|---|---|
| Russia 1+2 | 0.980 | 1.000 | 1.000 | 0.990 | 1.000 | – |
| Russia A+B | 0.525 | 1.000 | 1.000 | 0.995 | 0.990 | – |
| Russia C+D | 0.990 | 0.990 | 1.000 | 0.995 | 0.990 | – |
| Sport/Soccer | 0.240 | 0.920 | 0.890 | 0.990 | 0.990 | 0.965 |
| Tourism | 0.935 | 0.990 | 0.990 | 0.960 | 0.960 | – |
| People+Pets | 0.505 | 1.000 | 0.810 | 0.910 | 0.505 | 0.980 |

**Tabelle B.11.:** Optimale Werte für $\tau$ beim Matching-System FOAM

| | **unpartitioniert,** $\tau = 0.908$ | | | | | | **Baseline-250,** $\tau = 0.95$ | | | | | | **Baseline-500,** $\tau = 0.95$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 56 | 41 | 15 | 76 | 1.000 | 0.997 | 56 | 41 | 15 | 76 | 1.000 | 0.997 | 56 | 41 | 15 | 76 | 1.000 | 0.997 |
| Russia A+B | 162 | 160 | 2 | 125 | 1.000 | 0.983 | 162 | 160 | 2 | 125 | 1.000 | 0.983 | 162 | 160 | 2 | 125 | 1.000 | 0.983 |
| Russia C+D | 137 | 134 | 3 | 81 | 1.000 | 1.000 | 139 | 134 | 5 | 81 | 1.000 | 0.987 | 138 | 134 | 4 | 81 | 1.000 | 0.992 |
| Sport/Soccer | 85 | 73 | 12 | 77 | 0.998 | 0.972 | 80 | 71 | 9 | 79 | 1.0 | 0.991 | 80 | 71 | 9 | 79 | 1.0 | 0.991 |
| Tourism | 169 | 161 | 8 | 65 | 0.992 | 0.970 | 153 | 147 | 6 | 79 | 0.998 | 0.994 | 153 | 147 | 6 | 79 | 0.998 | 0.994 |
| People+Pets | 65 | 63 | 2 | 30 | 1.000 | 1.000 | 65 | 63 | 2 | 30 | 1.000 | 1.000 | 65 | 63 | 2 | 30 | 1.000 | 1.000 |

**Tabelle B.12.:** Ergebnisse mit INRIA und Filter, unpartitioniert und partitioniert mit Baseline-Algorithmus

| | **Islands-50,** $\tau = 0.952$ | | | | | | **Islands-100,** $\tau = 0.906$ | | | | | | $\varepsilon$**-Connections,** $\tau = 0.965$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 54 | 39 | 15 | 78 | 1.000 | 0.997 | 55 | 40 | 15 | 77 | 1.000 | 0.997 | – | | | | | |
| Russia A+B | 137 | 135 | 2 | 150 | 1.000 | 0.983 | 142 | 140 | 2 | 145 | 1.000 | 0.983 | – | | | | | |
| Russia C+D | 109 | 104 | 5 | 111 | 1.000 | 0.987 | 111 | 108 | 3 | 107 | 1.000 | 1.000 | – | | | | | |
| Sport/Soccer | 79 | 70 | 9 | 80 | 1.000 | 0.991 | 84 | 72 | 12 | 78 | 0.998 | 0.972 | 78 | 70 | 8 | 80 | 1.000 | 0.996 |
| Tourism | 148 | 145 | 3 | 81 | 0.998 | 1.000 | 163 | 154 | 9 | 72 | 0.994 | 0.960 | – | | | | | |
| People+Pets | 60 | 58 | 2 | 35 | 1.000 | 1.000 | 58 | 56 | 2 | 37 | 1.000 | 1.000 | 61 | 59 | 2 | 34 | 1.000 | 1.000 |

**Tabelle B.13.:** Ergebnisse mit INRIA und Filter, partitioniert mit Islands- und $\varepsilon$-Connections-Algorithmus

| | **unpartitioniert,** $\tau = 0.696$ | | | | | | **Baseline-250,** $\tau = 0.983$ | | | | | | **Baseline-500,** $\tau = 0.948$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 164 | 118 | 46 | 33 | 0.998 | 0.951 | 224 | 102 | 122 | 15 | 1.000 | 0.990 | 221 | 104 | 117 | 13 | 0.999 | 0.983 |
| Russia A+B | 281 | 280 | 1 | 5 | 1.000 | 1.000 | 321 | 284 | 37 | 1 | 1.000 | 0.988 | 321 | 284 | 37 | 1 | 1.000 | 0.981 |
| Russia C+D | 224 | 212 | 12 | 3 | 1.000 | 0.963 | 247 | 212 | 35 | 3 | 1.000 | 0.990 | 246 | 212 | 34 | 3 | 1.000 | 0.982 |
| Sport/Soccer | 136 | 124 | 12 | 26 | 0.989 | 0.956 | 127 | 99 | 28 | 51 | 0.999 | 0.990 | 146 | 126 | 20 | 24 | 0.994 | 0.990 |
| Tourism | 224 | 188 | 36 | 38 | 0.994 | 0.925 | 217 | 163 | 54 | 63 | 0.999 | 0.989 | 226 | 175 | 51 | 51 | 0.997 | 0.985 |
| People+Pets | 92 | 90 | 2 | 3 | 0.997 | 0.995 | 86 | 83 | 3 | 10 | 1.000 | 0.997 | 88 | 87 | 1 | 6 | 0.999 | 1.000 |

**Tabelle B.14.:** Ergebnisse mit FOAM und Filter, unpartitioniert und partitioniert mit Baseline-Algorithmus

| | **Islands-50,** $\tau = 0.973$ | | | | | | **Islands-100,** $\tau = 0.906$ | | | | | | $\varepsilon$**-Connections,** $\tau = 0.973$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 133 | 101 | 32 | 16 | 1.000 | 0.992 | 165 | 107 | 58 | 10 | 0.998 | 0.978 | – | | | | | |
| Russia A+B | 264 | 259 | 5 | 26 | 1.000 | 0.989 | 273 | 264 | 9 | 21 | 1.000 | 0.980 | – | | | | | |
| Russia C+D | 190 | 182 | 8 | 33 | 1.000 | 0.992 | 197 | 186 | 11 | 29 | 1.000 | 0.983 | – | | | | | |
| Sport/Soccer | 147 | 115 | 32 | 35 | 0.996 | 0.988 | 145 | 114 | 31 | 36 | 0.996 | 0.987 | 152 | 120 | 32 | 30 | 0.995 | 0.988 |
| Tourism | 193 | 171 | 22 | 55 | 0.998 | 0.988 | 194 | 168 | 26 | 58 | 0.998 | 0.975 | – | | | | | |
| People+Pets | 87 | 83 | 4 | 10 | 0.999 | 0.995 | 85 | 83 | 2 | 10 | 0.999 | 0.995 | 90 | 84 | 6 | 9 | 0.999 | 0.984 |

**Tabelle B.15.:** Ergebnisse mit FOAM und Filter, partitioniert mit Islands- und $\varepsilon$-Connections-Algorithmus

| | **Baseline-250,** $\tau = 0.95$ | | | | | | **Baseline-500,** $\tau = 0.95$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 56 | 41 | 15 | 76 | 1.000 | 0.997 | 56 | 41 | 15 | 76 | 1.000 | 0.997 |
| Russia A+B | 162 | 160 | 2 | 125 | 1.000 | 0.983 | 162 | 160 | 2 | 125 | 1.000 | 0.983 |
| Russia C+D | 137 | 134 | 3 | 81 | 1.000 | 1.000 | 137 | 134 | 3 | 81 | 1.000 | 1.000 |
| Sport/Soccer | 80 | 71 | 9 | 79 | 1.000 | 0.991 | 0 | 71 | 9 | 79 | 1.000 | 0.991 |
| Tourism | 152 | 147 | 5 | 79 | 0.998 | 1.000 | 152 | 147 | 5 | 79 | 0.998 | 1.000 |
| People+Pets | 65 | 63 | 2 | 30 | 1.000 | 1.000 | 65 | 63 | 2 | 30 | 1.000 | 1.000 |

**Tabelle B.16.:** Kombination von überlappenden Partitionen und Filtern mit dem Werkzeug INRIA, partitioniert mit Baseline-Algorithmus

| | **Islands-50,** $\tau = 0.952$ | | | | | | **Islands-100,** $\tau = 0.906$ | | | | | | $\varepsilon$**-Connections,** $\tau = 0.965$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 52 | 39 | 13 | 78 | 1.000 | 0.997 | 53 | 40 | 13 | 77 | 1.000 | 0.997 | – | | | | | |
| Russia A+B | 134 | 133 | 1 | 152 | 1.000 | 0.967 | 139 | 138 | 1 | 147 | 1.000 | 0.967 | – | | | | | |
| Russia C+D | 1103 | 101 | 2 | 114 | 1.000 | 1.000 | 106 | 104 | 2 | 111 | 1.000 | 1.000 | – | | | | | |
| Sport/Soccer | 79 | 70 | 9 | 80 | 1.000 | 0.991 | 84 | 72 | 12 | 78 | 0.998 | 0.972 | 78 | 70 | 8 | 80 | 1.000 | 0.996 |
| Tourism | 146 | 144 | 2 | 82 | 0.998 | 1.000 | 159 | 153 | 6 | 73 | 0.994 | 0.949 | – | | | | | |
| People+Pets | 59 | 57 | 2 | 36 | 1.000 | 1.000 | 58 | 56 | 2 | 37 | 1.000 | 1.000 | 61 | 59 | 2 | 34 | 1.000 | 1.000 |

**Tabelle B.17.:** Kombination von überlappenden Partitionen und Filtern mit dem Werkzeug INRIA, partitioniert mit Islands- und $\varepsilon$-Connections-Algorithmus

| | **Baseline-250,** $\tau = 0.983$ | | | | | | **Baseline-500,** $\tau = 0.948$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 155 | 105 | 50 | 12 | 0.999 | 0.991 | 187 | 106 | 81 | 11 | 0.999 | 0.982 |
| Russia A+B | 285 | 282 | 3 | 3 | 1.000 | 0.993 | 288 | 284 | 4 | 1 | 1.000 | 0.985 |
| Russia C+D | 218 | 211 | 7 | 4 | 1.000 | 0.995 | 222 | 212 | 10 | 3 | 1.000 | 0.982 |
| Sport/Soccer | 140 | 107 | 33 | 43 | 0.999 | 0.990 | 158 | 127 | 31 | 23 | 0.994 | 0.982 |
| Tourism | 210 | 177 | 33 | 49 | 0.999 | 0.989 | 213 | 189 | 24 | 37 | 0.997 | 0.988 |
| People+Pets | 89 | 87 | 2 | 6 | 1.000 | 0.995 | 91 | 88 | 3 | 5 | 0.999 | 0.990 |

**Tabelle B.18.:** Kombination von überlappenden Partitionen und Filtern mit dem Werkzeug FOAM, partitioniert mit Baseline-Algorithmus

| | **Islands-50,** $\tau = 0.973$ | | | | | | **Islands-100,** $\tau = 0.906$ | | | | | | $\varepsilon$**-Connections,** $\tau = 0.973$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
| Russia 1+2 | 144 | 102 | 42 | 15 | 1.000 | 0.991 | 155 | 105 | 50 | 12 | 0.999 | 0.982 | – | | | | | |
| Russia A+B | 258 | 256 | 2 | 29 | 1.000 | 0.983 | 263 | 262 | 1 | 23 | 1.000 | 0.943 | – | | | | | |
| Russia C+D | 185 | 179 | 6 | 36 | 1.000 | 0.994 | 190 | 182 | 8 | 33 | 1.000 | 0.973 | – | | | | | |
| Sport/Soccer | 155 | 123 | 32 | 27 | 0.996 | 0.988 | 152 | 125 | 27 | 25 | 0.995 | 0.976 | 151 | 126 | 25 | 24 | 0.995 | 0.987 |
| Tourism | 209 | 177 | 32 | 49 | 0.998 | 0.988 | 205 | 177 | 28 | 49 | 0.997 | 0.966 | – | | | | | |
| People+Pets | 86 | 82 | 4 | 11 | 0.999 | 0.994 | 84 | 80 | 4 | 13 | 1.000 | 0.994 | 86 | 84 | 2 | 9 | 0.999 | 0.995 |

**Tabelle B.19.:** Kombination von überlappenden Partitionen und Filtern mit dem Werkzeug FOAM, partitioniert mit Islands- und $\varepsilon$-Connections-Algorithmus

| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
|---|---|---|---|---|---|---|
| Russia 1+2 | 19734 | 69 | 19665 | 92 | 0.882 | 0.505 |
| Russia A+B | 12618 | 160 | 12458 | 125 | 0.938 | 0.510 |
| Russia C+D | 10401 | 135 | 10266 | 80 | 0.953 | 0.509 |
| Sport/Soccer | 6 | 1 | 5 | 149 | 0.800 | 0.495 |
| Tourism | 131045 | 193 | 130852 | 33 | 0.957 | 0.501 |
| People+Pets | 2521 | 71 | 2450 | 22 | 0.961 | 0.488 |

**Tabelle B.20.:** Ergebnisse mit dem Algorithmus `Structure` des Werkzeugs CROSI auf unpartitionierten Daten

| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
|---|---|---|---|---|---|---|
| Russia 1+2 | 19734 | 69 | 19665 | 92 | 0.882 | 0.505 |
| Russia A+B | 12618 | 160 | 12458 | 125 | 0.935 | 0.510 |
| Russia C+D | 10401 | 135 | 10266 | 80 | 0.952 | 0.509 |
| Sport/Soccer | 6 | 1 | 5 | 149 | 0.800 | 0.495 |
| Tourism | 131045 | 193 | 130852 | 33 | 0.957 | 0.501 |
| People+Pets | 2521 | 71 | 2450 | 22 | 0.961 | 0.488 |

**Tabelle B.21.:** Ergebnisse mit dem Algorithmus `StructurePlus` des Werkzeugs CROSI auf unpartitionierten Daten

| **Testfall** | *abs* | *tp* | *fp* | *fn* | *c(tp)* | *c(fp)* |
|---|---|---|---|---|---|---|
| Russia 1+2 | 6353 | 65 | 6288 | 96 | 0.995 | 0.996 |
| Russia A+B | 4415 | 60 | 4355 | 225 | 0.996 | 1.000 |
| Russia C+D | 3847 | 51 | 3796 | 164 | 1.000 | 1.000 |
| Sport/Soccer | 672 | 1 | 671 | 149 | 1.000 | 1.000 |
| Tourism | 10823 | 201 | 10622 | 25 | 0.969 | 0.978 |
| People+Pets | 256 | 35 | 221 | 58 | 1.000 | 0.986 |

**Tabelle B.22.:** Ergebnisse mit dem Algorithmus `HierarchyDisSim` des Werkzeugs CROSI auf unpartitionierten Daten

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Verzeichnis der Quelltextbeispiele

# Literaturverzeichnis


[Abecker und van Elst 2004] Abecker, Andreas ; van Elst, Ludger: *Ontologies for Knowledge Management*. Kap. 22, S. 435–454. Siehe (Staab und Studer, 2004)

[Altova 2007] Altova: *Altova SemanticWorks*. 2007. – Internetquelle: `http://www.altova.com/documents/SemanticWorksdatasheet.pdf`, heruntergeladen am 24.01.2008

[Antoniou u. a. 2005] Antoniou, Grigoris ; Franconia, Enrico ; van Harmelen, Frank: Introduction to Semantic Web Ontology Languages. In: (Eisinger und Maluszynski, 2005), S. 1–21

[Antoniou und van Harmelen 2004] Antoniou, Grigoris ; van Harmelen, Frank: *Web Ontology Language: OWL*. Kap. 4, S. 67–92. Siehe (Staab und Studer, 2004)

[Ashpole u. a. 2005] Ashpole, Benjamin (Hrsg.) ; Ehrig, Marc (Hrsg.) ; Euzenat, Jérôme (Hrsg.) ; Stuckenschmidt, Heiner (Hrsg.): *Integrating Ontologies '05, Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies, Banff, Canada, October 2*. 2005. (CEUR Workshop Proceedings 156)

[Aumueller u. a. 2005] Aumueller, David ; Do, Hong H. ; Massmann, Sabine ; Rahm, Erhard: Schema and ontology matching with COMA++. In: Özcan, Fatma (Hrsg.): *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*. New York : ACM, 2005, S. 906–908

[Baader u. a. 2005] Baader, Franz ; Horrocks, Ian ; Sattler, Ulrike: Description Logics as Ontology Languages for the Semantic Web. In: Hutter, Dieter (Hrsg.) ; Stephan, Werner (Hrsg.): *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, Springer, 2005 (LNCS 2605), S. 228–248

[Baeza-Yates und Ribeiro-Neto 1999] Baeza-Yates, Ricardo ; Ribeiro-Neto, Berthier: *Modern Information Retrieval*. Harlow, Essex : Pearson Education, 1999

[Bechhofer u. a. 2003a] Bechhofer, Sean ; Möller, Ralf ; Crowther, Peter: The DIG Description Logic Interface. In: Calvanese, Diego (Hrsg.) ; Giacomo, Giuseppe D. (Hrsg.) ; Franconi, Enrico (Hrsg.): *Proceedings of the 2003 International Workshop on Description Logics (DL2003), Rome, Italy September 5-7, 2003*, 2003 (CEUR Workshop Proceedings 81)

[Bechhofer u. a. 2003b] Bechhofer, Sean ; Volz, Raphael ; Lord, Phillip W.: Cooking the Semantic Web with the OWL API. In: (Fensel u. a., 2003), S. 659–675

[Berners-Lee 2006] Berners-Lee, Tim: *Artificial Intelligence and the Semantic Web*. Juli 2006. – Internetquelle: `http://www3.org/2006/Talks/0718-aaai-tbl/Overview.html`, heruntergeladen am 17.11.2007

[Berners-Lee u. a. 1998] Berners-Lee, Tim ; Fielding, Roy T. ; Masinter, Larry: *RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax*. August 1998. – Internetquelle: `http://tools.ietf.org/html/rfc3986`, heruntergeladen am 18.11.2007

[Berners-Lee u. a. 2001] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora: The Semantic Web. In: *Scientific American* 284 (2001), Nr. 5, S. 34–43

[Besana u. a. 2005] Besana, Paolo ; Robertson, Dave ; Rovatsos, Michael: Exploiting interaction contexts in P2P ontology mapping. In: Horrocks, Ian (Hrsg.) ; Sattler, Ulrike (Hrsg.) ; Wolter, Frank (Hrsg.): *Proceedings of the 2005 International Workshop on Description Logics*, 2005 (CEUR Workshop Proceedings 147)

[Bloehdorn u. a. 2005] Bloehdorn, Stephan ; Haase, Peter ; Sure, York ; Völker, Johanna ; Bevk, Matjaz ; Bontcheva, Kalina ; Roberts, Ian: D6.6.1 Report on the integration of ML, HLT and OM / University of Karlsruhe. Juli 2005. – SEKT Deliverable

[Blumauer und Fundneider 2006] Blumauer, Andreas ; Fundneider, Thomas: *Semantische Technologien in integrierten Wissensmanagement-Systemen*. S. 227–239. Siehe (Pellegrini und Blumauer, 2006)

[Blumauer und Pellegrini 2006] Blumauer, Andreas ; Pellegrini, Tassilo: *Semantic Web und semantische Technologien: Zentrale Begriffe und Unterscheidungen*. S. 9–25. Siehe (Pellegrini und Blumauer, 2006)

[Bouquet u. a. 2004] Bouquet, Paolo ; Ehrig, Marc ; Euzenat, Jerome ; Franconi, Enrico ; Hitzler, Pascal ; Krötzsch, Markus ; Serafini, Luciano ; Stamou, Giorgos ; Sure, York ; Tessaris, Sergio: Specification of a common framework for characterizing alignment / University of Karlsruhe. Dezember 2004 (2.2.1v2). – Knowledge Web Deliverable. Internetquelle: `http://www.aifb.uni-karlsruhe.de/WBS/phi/pub/kweb-221.pdf`, heruntergeladen am 5.10.2007

[Bouquet u. a. 2003] Bouquet, Paolo ; Giunchiglia, Fausto ; Harmelen, Frank van ; Serafini, Luciano ; Stuckenschmidt, Heiner: C-OWL: Contextualizing Ontologies. In: (Fensel u. a., 2003), S. 164–179

[Broekstra u. a. 2002] Broekstra, Jeen ; Kampman, Arjohn ; van Harmelen, Frank: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: Horrocks, Ian (Hrsg.) ; Hendler, James A. (Hrsg.): *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, Springer, 2002 (LNCS 2342), S. 54–68

[de Bruijn u. a. 2006] Bruijn, Jos de ; Ehrig, Marc ; Feier, Cristina ; Martín-Recuerda, Francisco ; Scharffe, François ; Weiten, Moritz: Ontology Mediation, Merging and Aligning. In: (Davies u. a., 2006b)

[Burstein und McDermott 2005] Burstein, Mark H. ; McDermott, Drew V.: Ontology Translation for Interoperability Among Semantic Web Services. In: *AI Magazine* 26 (2005), Nr. 1, S. 71–82. – Internetquelle: `http://openmap.bbn.com/~burstein/docs/AIMAG-Spring05-Burstein-McDermott.pdf`, heruntergeladen am 27.01.2008. Seitennummerierung folgt der Internetquelle.

[Cardoso 2006] Cardoso, Jorge: *Programming the Semantic Web*. Kap. 14, S. 351–380. Siehe (Cardoso und Sheth, 2006b)

[Cardoso 2007] Cardoso, Jorge: The Semantic Web Vision: Where Are We? In: *IEEE Intelligent Systems* 22 (2007), Nr. 5, S. 84–88

[Cardoso und Sheth 2006a] Cardoso, Jorge ; Sheth, Amit: *The Semantic Web and its Applications*. Kap. 1, S. 3–33. Siehe (Cardoso und Sheth, 2006b)

[Cardoso und Sheth 2006b] Cardoso, Jorge (Hrsg.) ; Sheth, Amit P. (Hrsg.): *Semantic Web Services, Processes and Applications*. Springer, 2006 (Semantic Web and Beyond - Computing for Human Experience)

[Carroll u. a. 2004] Carroll, Jeremy J. ; Dickinson, Ian ; Dollin, Chris ; Reynolds, Dave ; Seaborne, Andy ; Wilkinson, Kevin: Jena: Implementing the Semantic Web Recommendations. In: Feldman, Stuart I. (Hrsg.) ; Uretsky, Mike (Hrsg.) ; Najork, Marc (Hrsg.) ; Wills, Craig E. (Hrsg.): *Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters*, ACM, 2004, S. 74–83

[Cohen u. a. 2003] Cohen, William W. ; Ravikumar, Pradeep ; Fienberg, Stephen E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: Kambhampati, Subbarao (Hrsg.) ; Knoblock, Craig A. (Hrsg.): *Proceedings of IJCAI-03 Workshop on Information Integration on the Web*, 2003, S. 73–78

[Corcho und Gómez-Pérez 2001] Corcho, Oscar ; Gómez-Pérez, Asunción: Solving integration problems of e-commerce standards and initiatives through ontological mappings. In: *Proceedings of the Workshop on E-Business and Intelligent Web*, 2001, S. 131–140

[Cross 2003] Cross, Valerie: Uncertainty in the Automation of Ontology Matching. In: *ISUMA '03: Proceedings of the 4th International Symposium on Uncertainty Modelling and Analysis*. Washington : IEEE Computer Society, 2003, S. 135–140

[Cruz u. a. 2006] Cruz, Isabel F. (Hrsg.) ; Decker, Stefan (Hrsg.) ; Allemang, Dean (Hrsg.) ; Preist, Chris (Hrsg.) ; Schwabe, Daniel (Hrsg.) ; Mika, Peter (Hrsg.) ; Uschold, Michael (Hrsg.) ; Aroyo, Lora (Hrsg.): *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference*. Springer, 2006. (LNCS 4273)

[Daconta u. a. 2003] Daconta, Michael C. ; Obrst, Leo J. ; Smith, Kevin T.: *The Semantic Web. A Guide to the Future of XML, Web Services, and Knowledge Management*. Indianapolis : Wiley Publishing Inc., 2003

[Davies u. a. 2004] Davies, John (Hrsg.) ; Studer, Rudi (Hrsg.) ; Warren, Paul (Hrsg.): *The Semantic Web – Research and Applications. Proceedings of the First European Semantic Web Symposium*. Springer, 2004 (LNCS 3053)

[Davies u. a. 2006a] Davies, John ; Studer, Rudi ; Warren, Paul: Introduction. Siehe (Davies u. a., 2006b), S. 1–8

[Davies u. a. 2006b] Davies, John (Hrsg.) ; Studer, Rudi (Hrsg.) ; Warren, Paul (Hrsg.): *Semantic Web Technologies. Trends and Research in Ontology-based Systems*. Chichester : John Wiley and Sons, 2006

[Davis u. a. 1993] Davis, Randall ; Shrobe, Howard E. ; Szolovits, Peter: What Is a Knowledge Representation? In: *AI Magazine* 14 (1993), Nr. 1, S. 17–33

[Dean u. a. 2005] Dean, Mike (Hrsg.) ; Guo, Yuanbo (Hrsg.) ; Jun, Woochun (Hrsg.) ; Kaschek, Roland (Hrsg.) ; Krishnaswamy, Shonali (Hrsg.) ; Pan, Zhengxiang (Hrsg.) ; Sheng, Quan Z. (Hrsg.): *Web Information Systems Engineering - WISE 2005 Workshops*. Springer, 2005. (LNCS 3807)

[Ding u. a. 2005] Ding, Li ; Finin, Timothy W. ; Joshi, Anupam ; Peng, Yun ; Pan, Rong ; Reddivari, Pavan: Search on the Semantic Web. In: *IEEE Computer* 38 (2005), Nr. 10, S. 62–69

[Ding u. a. 2004] Ding, Ying ; Fensel, Dieter ; Klein, Michel ; Omelayenko, Borys ; Schulten, Ellen: *The Role of Ontologies in eCommerce*. Kap. 30, S. 593–615. Siehe (Staab und Studer, 2004)

[Do u. a. 2002] Do, Hong H. ; Melnik, Sergey ; Rahm, Erhard: Comparison of Schema Matching Evaluations. In: Chaudhri, Akmal B. (Hrsg.) ; Jeckle, Mario (Hrsg.) ; Rahm, Erhard (Hrsg.) ; Unland, Rainer (Hrsg.): *Web, Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops*, Springer, 2002 (LNCS 2593), S. 221–237

[Do und Rahm 2002] Do, Hong H. ; Rahm, Erhard: COMA - A System for Flexible Combination of Schema Matching Approaches. In: Bernstein, Philip A. (Hrsg.) ; Ioannidis, Yannis E. (Hrsg.) ; Ramakrishnan, Raghu (Hrsg.) ; Papadias, Dimitris (Hrsg.): *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases*, Morgan Kaufmann, 2002, S. 610–621

[Do und Rahm 2007] Do, Hong-Hai ; Rahm, Erhard: Matching large schemas: Approaches and evaluation. In: *Information Systems* 32 (2007), Nr. 6, S. 857–885

[Dostal und Jeckle 2004] Dostal, Wolfgang ; Jeckle, Mario: Semantik und Web Services. In: *JavaSPEKTRUM* 4 (2004), S. 58–62

[Dustdar u. a. 2006] Dustdar, Schahram ; Fensel, Dieter ; Linder, Markus ; Otruba, Heinrich ; Pellegrini, Tassilo ; Schliefnig, Martin: *The realization of Semantic Web based E-Commerce and its impact on Business, Consumers and the Economy*. April 2006. – Internetquelle: `http://www.austriapro.at/arbeitskreise/pva/37uebersichtsartikel.pdf`, heruntergeladen am 16.11.2007

[Ehrig 2007a] Ehrig, Marc: *Framework for Ontology Alignment and Mapping. Test Ontologies and Alignments*. März 2007. – Internetquelle: `http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/ontologies.htm`, heruntergeladen am 16.12.2007

[Ehrig 2007b] Ehrig, Marc ; Jain, Ramesh (Hrsg.) ; Sheth, Amit (Hrsg.): *Ontology Alignment - Bridging the Semantic Gap*. New York : Springer, 2007 (Semantic Web and Beyond. Computing for Human Experience)

[Ehrig u. a. 2005] Ehrig, Marc ; Haase, Peter ; Hefke, Mark ; Stojanovic, Nenad: Similarity for Ontologies – A Comprehensive Framework. In: *Proceedings of the 13th European Conference on Information Systems, Information Systems in a Rapidly Changing Economy*, 2005, S. 1509–1518

[Ehrig und Staab 2004a] Ehrig, Marc ; Staab, Steffen: QOM - Quick Ontology Mapping. In: (McIlraith u. a., 2004), S. 683–697

[Ehrig und Staab 2004b] Ehrig, Marc ; Staab, Steffen: QOM - Quick Ontology Mapping / Institut AIFB, Universität Karlsruhe. August 2004. – Forschungsbericht. Internetquelle: `http://www.aifb.uni-karlsruhe.de/WBS/meh/publications/ehrig04QOMTR.pdf`, heruntergeladen am 26.10.2007

[Ehrig und Sure 2004] Ehrig, Marc ; Sure, York: Ontology Mapping - An Integrated Approach. In: (Davies u. a., 2004), S. 76–91

[Ehrig und Sure 2005] Ehrig, Marc ; Sure, York: FOAM - Framework for Ontology Alignment and Mapping. Results of the Ontology Alignment Initiative. In: (Ashpole u. a., 2005), S. 72–76

[Eisinger und Maluszynski 2005] Eisinger, Norbert (Hrsg.) ; Maluszynski, Jan (Hrsg.): *Reasoning Web, First International Summer School*. Springer, 2005. (LNCS 3564)

[Euzenat und Valtchev 2004] Euzenat, Jérôme ; Valtchev, Petko: Similarity-Based Ontology Alignment in OWL-Lite. In: de Mántaras, Ramon L. (Hrsg.) ; Saitta, Lorenza (Hrsg.): *Proceedings of the 16th Eureopean Conference on Artificial Intelligence*. Amsterdam, Fairfax, Lancaster, Tokyo : IOS Press, 2004, S. 333–337. – ISBN 1-58603-452-9

[Euzenat 2004] Euzenat, Jérôme: An API for Ontology Alignment. In: (McIlraith u. a., 2004), S. 698–712

[Euzenat u. a. 2005] Euzenat, Jérôme ; Guègan, Philippe ; Valtchev, Petko: OLA in the OAEI 2005 Alignment Contest. In: (Ashpole u. a., 2005), S. 97–102

[Euzenat u. a. 2004] Euzenat, Jérôme ; Loup, David ; Touzani, Mohamed ; Valtchev, Petko: Ontology alignment with OLA. In: Sure, York (Hrsg.) ; Corcho, Oscar (Hrsg.) ; Euzenat, Jérôme (Hrsg.) ; Hughes, Todd (Hrsg.): *Proceedings of the 3rd ISWC2004 workshop on Evaluation of Ontology-based tools*, 2004, S. 59–68

[Euzenat u. a. 2008] Euzenat, Jérôme ; Mocan, Adrian ; Scharffe, François: *Ontology Alignments. An Ontology Management Perspective*. Kap. 6, S. 177–206. Siehe (Hepp u. a., 2008)

[Euzenat u. a. 2006] Euzenat, Jérôme ; Mochol, Malgorzata ; Shvaiko, Pavel ; Stuckenschmidt, Heiner ; Šváb, Ondřej ; Svátek, Vojtěch ; Hage, Willem R. van ; Yatskevich, Mikalai: Results of the Ontology Alignment Evaluation Initiative 2006. In: (Shvaiko u. a., 2006)

[Euzenat und Shvaiko 2007] Euzenat, Jérôme ; Shvaiko, Pavel: *Ontology Matching*. Berlin, Heidelberg, New York : Springer, 2007

[Fensel 2002] Fensel, Dieter: Language Standardization for the Semantic Web: The Long Way from OIL to OWL. In: Plaice, John (Hrsg.) ; Kropf, Peter G. (Hrsg.) ; Schulthess, Peter (Hrsg.) ; Slonim, Jacob (Hrsg.): *Distributed Communities on the Web, 4th International Workshop*, Springer, 2002 (LNCS 2468), S. 215–227

[Fensel 2004] Fensel, Dieter: *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Zweite Auflage. Springer, 2004

[Fensel u. a. 2003] Fensel, Dieter (Hrsg.) ; Sycara, Katia P. (Hrsg.) ; Mylopoulos, John (Hrsg.): *The Semantic Web - ISWC 2003, Second International Semantic Web Conference*. Springer, 2003. (LNCS 2870)

[Fischer 2004] Fischer, Dietrich H.: Ein Lehrbeispiel für eine Ontologie: OpenCyc. In: *Information - Wissenschaft & Praxis* 55 (2004), Nr. 3, S. 139–142

[Gamma u. a. 1996] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software*. 5., korrigierter Nachdruck. Addison-Wesley, 1996

[Geiger 1995] Geiger, Kyle: *Inside ODBC*. Unterschleißheim : Microsoft Press Deutschland, 1995

[Gene Ontology Consortium 2007] The Gene Ontology Consortium: *The Gene Ontology*. 2007. – Internetquelle: `http://www.thegeneontology.org`, heruntergeladen am 30.08.2007

[Gennari u. a. 2003] Gennari, John H. ; Musen, Mark A. ; Fergerson, Ray W. ; Grosso, William E. ; Crubézy, 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 (2003), Januar, Nr. 1, S. 89–123

[Getty Research Institute 2006] Getty Research Institute: *About the TGN*. 2006. – Internetquelle: `http://www.getty.edu/research/conducting_research/vocabularies/tgn/about.html`, heruntergeladen am 13.09.2007

[Ghidini u. a. 2007] Ghidini, Chiara ; Serafini, Luciano ; Tessaris, Sergio: On Relating Heterogeneous Elements from Different Ontologies. In: Kokinov, Boicho N. (Hrsg.) ; Richardson, Daniel C. (Hrsg.) ; Roth-Berghofer, Thomas (Hrsg.) ; Vieu, Laure (Hrsg.): *Modeling and Using Context, 6th International and Interdisciplinary Conference*, Springer, 2007 (LNCS 4635), S. 234–247

[Giunchiglia und Shvaiko 2003] Giunchiglia, Fausto ; Shvaiko, Pavel: Semantic matching. In: *The Knowledge Engineering Review* Bd. 18. New York, NY, USA : Cambridge University Press, 2003, S. 265 – 280

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

[Giunchiglia u. a. 2007] Giunchiglia, Fausto ; Yatskevich, Mikalai ; Shvaiko, Pavel: Semantic Matching: Algorithms and Implementation. In: *Journal on Data Semantics* 9 (2007), Nr. 4601, S. 1–38

[Gómez-Pérez u. a. 2004] Gómez-Pérez, Asunción ; Fernández-López, Mariano ; Corcho, Oscar: *Ontological Engineering*. Springer, 2004 (Advanced Information and Knowledge Processing)

[Grau u. a. 2006] Grau, Bernardo C. ; Parsia, Bijan ; Sirin, Evren: Combining OWL ontologies using epsilon-Connections. In: *Journal of Web Semantics* 4 (2006), Nr. 1, S. 40–59. – Internetquelle: `http://www.mindswap.org/2004/multipleOnt/papers/EconnJWS.pdf`, heruntergeladen am 17.10.2007. Seitennummerierung wie in Internetquelle.

[Grau u. a. 2005] Grau, Bernardo C. ; Parsia, Bijan ; Sirin, Evren ; Kalyanpur, Aditya: Modularizing OWL Ontologies. In: Sleeman, Derek (Hrsg.) ; Alani, Harith (Hrsg.) ; Brewster, Christopher (Hrsg.) ; Noy, Natasha (Hrsg.): *Proceedings of the KCAP-2005 Workshop on Ontology Management*, 2005. – Internetquelle: `http://www.mindswap.org/2004/multipleOnt/papers/Modularization.pdf`, heruntergeladen am 17.10.2007. Seitenzahlen folgen dieser Quelle.

[Grobelnik und Mladenić 2006] Grobelnik, Marko ; Mladenić, Dunja: Knowledge Discovery for Ontology Construction. In: (Davies u. a., 2006b), S. 9–27

[Gruber 1993] Gruber, Thomas R.: A translation approach to portable ontology specifications. In: *Knowledge Acquisition* 5 (1993), Juni, Nr. 2, S. 199–220

[Gruber 1995] Gruber, Thomas R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In: *International Journal of Human Computer Studies* 43 (1995), Nr. 5-6, S. 907–928

[Gruninger und Lee 2002] Gruninger, Michael ; Lee, Jintae: Ontology Applications and Design. In: *Communications of the ACM* 45 (2002), Februar, Nr. 2, S. 39–41

[Guo und Heflin 2006] Guo, Yuanbo ; Heflin, Jeff: A Scalable Approach for Partitioning OWL Knowledge Bases. In: Wache, Holger (Hrsg.) ; Stuckenschmidt, Heiner (Hrsg.) ; Parsia, Bijan (Hrsg.) ; Guo, Yuanbo (Hrsg.) ; Finin, Tim (Hrsg.) ; Beckett, Dave (Hrsg.): *Proceedings of the Second International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)*, 2006, S. 47–60. – Internetquelle: `http://www.cs.vu.nl/~holger/ssws2006/Proceedings-SSWS06.pdf`, heruntergeladen am 18.12.2007

[Haarslev und Möller 2003a] Haarslev, Volker ; Möller, Ralf: Racer: A Core Inference Engine for the Semantic Web. In: Sure, York (Hrsg.) ; Corcho, Oscar (Hrsg.): *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools*, 2003, S. 27–36

[Haarslev und Möller 2003b] Haarslev, Volker ; Möller, Ralf: Racer: An OWL Reasoning Agent for the Semantic Web. In: Yao, Jing Tao (Hrsg.) ; Lingras, Pawan (Hrsg.): *Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems*, 2003, S. 91–95

[Hamp und Feldwig 1997] Hamp, Birgit ; Feldwig, Helmut: GermaNet – A Lexical-Semantic Net for German. In: Vossen, Piek (Hrsg.) ; Adriaens, Geert (Hrsg.) ; Calzolari, Nicoletta (Hrsg.) ; Sanfilippo, Antonio (Hrsg.) ; Wilks, Yorick (Hrsg.): *Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*. New Brunswick, New Jersey : Association for Computational Linguistics, 1997, S. 9–15

[Harris und Gibbins 2003] Harris, Stephen ; Gibbins, Nicholas: 3store: Efficient Bulk RDF Storage. In: Volz, Raphael (Hrsg.) ; Decker, Stefan (Hrsg.) ; Cruz, Isabel F. (Hrsg.): *Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, 2003 (CEUR Workshop Proceedings 89)

[Harris und Shadbolt 2005] Harris, Stephen ; Shadbolt, Nigel: SPARQL Query Processing with Conventional Relational Database Systems. In: (Dean u. a., 2005), S. 235–244

[Hendler 2001] Hendler, James: Agents and the Semantic Web. In: *IEEE Intelligent Systems* 16 (2001), Nr. 2, S. 30–37

[Hepp 2006] Hepp, Martin: *eClassOWL 5.1. Products and Services Ontology for e-Business. User's Guide*. 2006. – Internetquelle: `http://www.heppnetz.de/eclassowl/eclassOWL-Primer-final.pdf`, heruntergeladen am 6.9.2007

[Hepp 2008] Hepp, Martin: *Ontologies: State of the Art, Business Potential, and Grand Challenges*. Kap. 1, S. 3–22. Siehe (Hepp u. a., 2008)

[Hepp u. a. 2008] Hepp, Martin (Hrsg.) ; De Leenheer, Pieter (Hrsg.) ; de Moor, Aldo (Hrsg.) ; Sure, York (Hrsg.): *Ontology Management. Semantic Web, Semantic Web Services, and Business Applications*. Springer, 2008 (Semantic Web and Beyond. Computing for Human Experience)

[Heymans u. a. 2008] Heymans, Stijn ; Ma, Li ; Anicic, Darko ; Ma, Zhilei ; Steinmetz, Nathalie ; Pan, Yue ; Mei, Jing ; Fokoue, Achille ; Kalyanpur, Aditya ; Kershenbaum, Aaron ; Schonberg, Edith ; Srinivas, Kavitha ; Feier, Cristina ; Hench, Graham ; Wetzstein, Branimir ; Keller, Uwe: *Ontology Reasoning with Large Data Repositories*. Kap. 4, S. 89–128. Siehe (Hepp u. a., 2008)

[Hitzler u. a. 2008] Hitzler, Pascal ; Krötzsch, Markus ; Rudolph, Sebastian ; Sure, York: *Semantic Web. Grundlagen*. Erste Auflage. Springer, 2008 (eXamen.press)

[Holmes und McCabe 2002] Holmes, David ; McCabe, M. C.: Improving Precision and Recall for Soundex Retrieval. In: *2002 International Symposium on Information Technology (ITCC 2002), 8-10 April 2002, Las Vegas, NV, USA*. New York : IEEE Computer Society, 2002, S. 22–27

[Horridge u. a. 2007] Horridge, Matthew ; Bechhofer, Sean ; Noppens, Olaf: Igniting the OWL 1.1 Touch Paper: The OWL API. In: Golbreich, Christine (Hrsg.) ; Kalyanpur, Aditya (Hrsg.) ; Parsia, Bijan (Hrsg.): *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, 2007 (CEUR Workshop Proceedings 258)

[Horrocks u. a. 2004] Horrocks, Ian ; Patel-Schneider, Peter F. ; Boley, Harold ; Tabet, Said ; Grosof, Benjamin ; Dean, Mike: *SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission*. Mai 2004. – Online-Quelle: `http://www.w3.org/Submission/SWRL/`, heruntergeladen am 16.10.2007

[Horrocks u. a. 2003] Horrocks, Ian ; Patel-Schneider, Peter F. ; Harmelen, Frank van: From SHIQ and RDF to OWL: The Making of a Web Ontology Language. In: *Journal of Web Semantics* 1 (2003), Nr. 1, S. 7–26. – Internetquelle: `http://www.cs.vu.nl/~frankh/postscript/JWS03.pdf`, heruntergeladen am 14.12.2007. Die Seitennummerierung folgt der Internetquelle

[Hu u. a. 2006a] Hu, Wei ; Cheng, Gong ; Zheng, Dondong ; Zhong, Xinyu ; Qu, Yuzhong: The Results of Falcon-AO in the OAEI 2006 Campaign. In: (Shvaiko u. a., 2006), S. 124 – 133

[Hu und Qu 2006] Hu, Wei ; Qu, Yuzhong: Block Matching for Ontologies. In: (Cruz u. a., 2006), S. 300–313

[Hu u. a. 2006b] Hu, Wei ; Zhao, Yuanyuan ; Qu, Yuzhong: Partition-Based Block Matching of Large Class Hierarchies. In: (Mizoguchi u. a., 2006)

[ISO 1986] ISO: *ISO 2788. Guidelines for the establishment and development of monolingual thesauri.* 1986. – Internetquelle: `http://www.collectionscanada.ca/iso/tc46sc9/standard/2788e.htm`, heruntergeladen am 6.12.2007

[ISO/IEC 2002] ISO/IEC: *Topic Maps (Second Edition).* Mai 2002. – Internetquelle: `http://www.y12.doe.gov/sgml/sc34/document/0322_files/iso13250-2nd-ed-v2.pdf`, heruntergeladen am 6.12.2007

[java.net 2007a] *JXTA CMS Homepage.* 2007. – Internetquelle: `https://jxse-cms.dev.java.net/`, heruntergeladen am 7.1.2008

[java.net 2007b] *JXTA SOAP Homepage.* 2007. – Internetquelle: `https://soap.dev.java.net/`, heruntergeladen am 7.1.2008

[John und Drescher 2006] John, Michael ; Drescher, Jörg: *Semantische Technologien im Informations- und Wissensmanagement: Geschichte, Anwendungen und Ausblick.* S. 241–255. Siehe (Pellegrini und Blumauer, 2006)

[Kalfoglou und Hu 2005] Kalfoglou, Yannis ; Hu, Bo: CROSI Mapping System (CMS) - Result of the 2005 Ontology Alignment Contest. In: (Ashpole u. a., 2005), S. 77–84

[Kalfoglou u. a. 2005] Kalfoglou, Yannis ; Hu, Bo ; Reynolds, Dave ; Shadbolt, Nigel: CROSI Project, Final Report / School of Electronics and Computer Science, University of Southampton. Southampton, UK, 2005. – Forschungsbericht. Internetquelle: `http://eprints.ecs.soton.ac.uk/11717/`, heruntergeladen am 24.10.2007

[Kalfoglou und Schorlemmer 2005] Kalfoglou, Yannis ; Schorlemmer, Marco: Ontology Mapping: The State of the Art. In: Kalfoglou, Yannis (Hrsg.) ; Schorlemmer, Marco (Hrsg.) ; Sheth, Amit P. (Hrsg.) ; Staab, Steffen (Hrsg.) ; Uschold, Michael (Hrsg.): *Semantic Interoperability and Integration*, 2005 (Dagstuhl Seminar Proceedings 04391), S. 1–31

[Kalyanpur u. a. 2006] Kalyanpur, Aditya ; Parsia, Bijan ; Sirin, Evren ; Grau, Bernardo C. ; Hendler, James: Swoop: A 'Web' Ontology Editing Browser. In: *Journal of Web Semantics* 4 (2006), Nr. 2, S. 144–153. – Internetquelle: `http://www.mindswap.org/papers/SwoopJWS_Revised.pdf`, heruntergeladen am 12.01.2008. Seitennummerierung wie in Internetquelle

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

[Kifer 2005] Kifer, Michael: *Rules and Ontologies in F-Logic.* S. 22–34. Siehe (Eisinger und Maluszynski, 2005)

[Kiryakov u. a. 2005] Kiryakov, Atanas ; Ognyanov, Damyan ; Manov, Dimitar: OWLIM - A Pragmatic Semantic Repository for OWL. In: (Dean u. a., 2005), S. 182–192

[Klein 2001] Klein, Michel: Combining and relating ontologies: an analysis of problems and solutions. In: Gomez-Perez, Asuncion (Hrsg.) ; Gruninger, Michael (Hrsg.) ; Stuckenschmidt, Heiner (Hrsg.) ; Uschold, Michael (Hrsg.): *Workshop on Ontologies and Information Sharing, IJCAI'01*, 2001, S. 52–63

[Klein 2002] Klein, Michel: *DAML+OIL and RDF Schema representation of UNSPSC*. 2002. – Internetquelle: `http://www.cs.vu.nl/~mcaklein/unspsc/`, heruntergeladen am 16.11.2007

[Knublauch u. a. 2004] Knublauch, Holger ; Fergerson, Ray W. ; Noy, Natalya F. ; Musen, Mark A.: The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In: (McIlraith u. a., 2004), S. 229–243

[Kunzmann u. a. 2003] Kunzmann, Peter ; Burkard, Franz-Peter ; Wiedmann, Franz: *dtv-Atlas Philosophie*. 11., aktualisierte Auflage. München : Deutscher Taschenbuch Verlag, 2003

[Lee 2004] Lee, Ryan: Scalability Report on Triple Store Applications / Massachusetts Institute of Technology. 2004. – Forschungsbericht. Internetquelle: `http://simile.mit.edu/reports/stores/stores.pdf`, heruntergeladen am 25.01.2008

[Liebig 2006] Liebig, Thorsten: Reasoning with OWL – System Support and Insights / Universität Ulm. 2006 (Informatik Bericht Nr. 2006-04). – Forschungsbericht. Internetquelle: `http://www.informatik.uni-ulm.de/ki/Liebig/papers/TR-U-Ulm-2006-04.pdf`, heruntergeladen am 26.01.2008

[Liu und Hu 2005] Liu, Baolin ; Hu, Bo: An Evaluation of RDF Storage Systems for Large Data Applications. In: *International Conference on Semantics, Knowledge and Grid*, IEEE Computer Society, 2005, S. 59–61

[Lüngen und Storrer 2006] Lüngen, Harald ; Storrer, Angelika: Domain ontologies and word nets in OWL: Modelling options. In: *Proceedings of the International Workshop „Ontologies in Text Technology“*, 2006, S. 3–10

[Lopez u. a. 2006] Lopez, Vanessa ; Sabou, Marta ; Motta, Enrico: PowerMap: Mapping the Real Semantic Web on the Fly. In: (Cruz u. a., 2006), S. 414–427

[Maedche und Staab 2002] Maedche, Alexander ; Staab, Steffen: Measuring Similarity between Ontologies. In: Gómez-Pérez, Asunción (Hrsg.) ; Benjamins, Richard (Hrsg.): *Proc. Of the European Conference on Knowledge Acquisition and Management - EKAW-2002. Madrid, Spain, October 1-4, 2002*, Springer, 2002 (LNCS/LNAI 2473), S. 251–263

[Malucelli u. a. 2005] Malucelli, Andreia ; Oliveira, Eugenio ; Palzer, Daniel: Combining Ontologies and Agents to help in Solving the Heterogeneity Problem in E-Commerce Negotiations. In: Lee, Sang goo (Hrsg.) ; Bussler, Christoph (Hrsg.) ; Shim, Simon (Hrsg.): *Proceedings of the International Workshop on Data Engineering Issues in E-Commerce*, IEEE Computer Society, 2005, S. 26–38

[Maryland Information and Network Dynamics Lab Semantic Web Agents Project 2007] Maryland Information and Network Dynamics Lab Semantic Web Agents Project: *National Cancer Institute Thesaurus*. 2007. – Internetquelle: `http://www.mindswap.org/2003/CancerOntology/`, heruntergeladen am 30.08.2007

[Masolo u. a. 2003] Masolo, Claudio ; Borgo, Stefano ; Gangemi, Aldo ; Guarino, Nicola ; Oltramari, Alessandro: WonderWeb Deliverable D18 – Ontology Library (final) / Laboratory For Applied Ontology, Trento, Italien. 2003. – Forschungsbericht. Internetquelle: `http://wonderweb.semanticweb.org/deliverables/documents/D18.pdf`, heruntergeladen am 6.12.2007

[Massmann u. a. 2006] Massmann, Sabine ; Engmann, Daniel ; Rahm, Erhard: COMA++: Results for the Ontology Alignment Contest OAEI 2006. In: (Shvaiko u. a., 2006), S. 107–114

[May 2006] May, Wolfgang: *Reasoning im und für das Semantic Web*. S. 485–503. Siehe (Pellegrini und Blumauer, 2006)

[McBride 2004] McBride, Brian: *The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS*. Kap. 3, S. 51–66. Siehe (Staab und Studer, 2004)

[McIlraith u. a. 2004] McIlraith, Sheila A. (Hrsg.) ; Plexousakis, Dimitris (Hrsg.) ; Harmelen, Frank van (Hrsg.): *The Semantic Web - ISWC 2004: Proceedings of the Third International Semantic Web Conference*. Springer, 2004. (LNCS 3298)

[Meilicke u. a. 2006] Meilicke, Christian ; Stuckenschmidt, Heiner ; Tamilin, Andrei: Improving Automatically Created Mappings using Logical Reasoning. In: (Shvaiko u. a., 2006), S. 61–72

[Meyer 2005] Meyer, Harald: Semantic Web Services. In: *ObjektSpektrum* Online-Ausgabe SOA (2005). – Internetquelle: `http://www.sigs.de/publications/os/2005/SOA/meyer_SOA_03_05.pdf`, heruntergeladen am 22.10.2007

[Milne 2000] Milne, A. A.: *Winnie-the-Pooh*. London : Egmont Books Limited, 2000

[Mitra u. a. 2005] Mitra, Prasenjit ; Liu, Peng ; Pan, Chi-Chun: *Privacy-preserving Ontology Matching*. 2005. – First International Workshop on Contexts and Ontologies: Theory, Practice and Applications, Poster Session held in conjunction with The Twentieth National Conference on Artificial Intelligence. Online-Quelle: `http://www.dit.unitn.it/%7Epavel/cando/Pictures/Posters/WW501MitraP.pdf`, heruntergeladen am 16.10.2007

[Mitra und Wiederhold 2002] Mitra, Prasenjit ; Wiederhold, Gio: Resolving Terminological Heterogeneity In Ontologies. In: Jérôme Euzenat, Nicola G. (Hrsg.) ; Stuckenschmidt, Heiner (Hrsg.): *Proceedings of the ECAI-02 Workshop on Ontologies and Semantic Interoperability*, 2002, S. 45–50

[Mizoguchi u. a. 2006] Mizoguchi, Riichiro (Hrsg.) ; Shi, Zhongzhi (Hrsg.) ; Giunchiglia, Fausto (Hrsg.): *The Semantic Web - ASWC 2006, First Asian Semantic Web Conference*. Springer, 2006. (LNCS 4183)

[Mochol u. a. 2006] Mochol, Malgorzata ; Jentzsch, Anja ; Euzenat, Jérôme: Applying an Analytic Method for Matching Approach Selection. In: (Shvaiko u. a., 2006), S. 37–48

[Motik und Sattler 2006] Motik, Boris ; Sattler, Ulrike: A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In: Hermann, Miki (Hrsg.) ; Voronkov, Andrei (Hrsg.): *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006, Phnom Penh, Cambodia, November 13-17, 2006, Proceedings*, Springer, 2006 (LNCS 4246), S. 227–241

[Musen 1989] Musen, Mark A.: *Automated Generation of Model-Based Knowledge-Acquisition Tools*. Online-Version unter `http://books.google.com/books?id=g42Il1Q8UnUC&hl=de`. Morgan Kaufmann, 1989

[Nagarajan 2006] Nagarajan, Meenakshi: *Semantic Annotations in Web Services*. Kap. 2, S. 35–61. Siehe (Cardoso und Sheth, 2006b)

[Niles und Pease 2001] Niles, Ian ; Pease, Adam: Towards a standard upper ontology. In: *FOIS*, 2001, S. 2–9

[Noy 2004] Noy, Natalya F.: Semantic Integration: A Survey Of Ontology-Based Approaches. In: *SIGMOD Record* 33 (2004), Nr. 4, S. 65–70

[Noy und Musen 2000] Noy, Natalya F. ; Musen, Mark A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, AAAI Press / The MIT Press, 2000, S. 450–455

[Noy und Musen 2001] Noy, Natalya F. ; Musen, Mark A.: Anchor-PROMPT: Using non-local context for semantic matching. In: *Proceedings of the IJCAI 2001 workshop on ontology and information sharing*, 2001, S. 63–70

[Noy und Musen 2003] Noy, Natalya F. ; Musen, Mark A.: The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. In: *International Journal of Human-Computer Studies* 59 (2003), Dezember, Nr. 6, S. 983–1024

[Oberle und Spyns 2004] Oberle, Daniel ; Spyns, Peter: *The Knowledge Portal "OntoWeb"*. Kap. 25, S. 499–516. Siehe (Staab und Studer, 2004)

[Ognyanoff u. a. 2006] Ognyanoff, Damyan ; Kiryakov, Atanas ; Velkov, Rouslan ; Yankova, Milena: D2.6.3 A scalable repository for massive semantic annotation / Ontotext Lab, Sirma Group Corp. 2006. – SEKT Deliverable. Online-Quelle: `http://www.ontotext.com/publications/SEKT_D2.6.3.PDF`, heruntergeladen am 06.02.2008

[Omelayenko 2001] Omelayenko, Borys: Ontology Integration Tasks in Business-to-Business E-Commerce. In: Monostori, Laszlo (Hrsg.) ; Váncza, József (Hrsg.) ; Ali, Moonis (Hrsg.): *Engineering of Intelligent Systems, Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE)*, Springer, 2001 (LNCS 2070), S. 119–124

[ontoprise GmbH 2007] ontoprise GmbH: *OntoStudio 2.0 Product Documentation*. 2007. – Internetquelle: `http://www.ontoprise.de/content/e1171/e1249/e1672/e1673/OntoStudio_2.0_docu_ger.pdf`, heruntergeladen am 24.01.2008

[Paulheim u. a. 2007] Paulheim, Heiko ; Rebstock, Michael ; Fengel, Janina: Context-Sensitive Referencing for Ontology Mapping Disambiguation. In: *Proceedings of the 2007 Workshop on Context and Ontologies Representation and Reasoning (C&O:RR-2007). Computer Science Research Report #115*, Roskilde University, 2007, S. 47–56

[Pellegrini und Blumauer 2006] Pellegrini, Tassino (Hrsg.) ; Blumauer, Andreas (Hrsg.): *Semantic Web. Wege zur vernetzten Wissensgesellschaft*. Springer, 2006

[Polleres u. a. 2006] Polleres, Axel ; Lausen, Holger ; Rubén Lara und: *Semantische Beschreibung von Web Services*. S. 505–524. Siehe (Pellegrini und Blumauer, 2006)

[Rahm und Bernstein 2001] Rahm, Erhard ; Bernstein, Philip A.: A survey of approaches to automatic schema matching. In: *The VLDB Journal* 10 (2001), Nr. 4, S. 334–350

[Rahm u. a. 2004] Rahm, Erhard ; Do, Hong H. ; Massmann, Sabine: Matching Large XML Schemas. In: *SIGMOD Record* 33 (2004), Nr. 4, S. 26–31

[Rebstock u. a. 2008] Rebstock, Michael ; Fengel, Janina ; Paulheim, Heiko: *Ontologies-based Business Integration*. Springer, 2008

[Rebstock u. a. 2005] Rebstock, Michael ; Fengel, Janina ; Petry, Matthias: Entwicklung von Anwendungskomponenten zur ontologienbasierten E-Business-Integration. In: *Querschnitt – Beiträge aus Forschung und Entwicklung der Fachhochschule Darmstadt* Bd. 19, Fachhochschule Darmstadt, 2005, S. 26–35

[Reif 2006] Reif, Gerald: *Semantische Annotation*. S. 405–418. Siehe (Pellegrini und Blumauer, 2006)

[van Rijsbergen 1979] Rijsbergen, Keith van: *Information Retrieval*. Zweite Auflage. London : Butterworths, 1979. – Online-Ausgabe: `http://www.dcs.gla.ac.uk/Keith/Preface.html`, heruntergeladen am 20.12.2007

[Russell und Norvig 2003] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence. A Modern Approach*. Second Edition. New Jersey : Pearson Education, 2003

[Scharffe und de Bruijn 2005] Scharffe, François ; de Bruijn, Jos: A language to specify mappings between ontologies. In: Chbeir, Richard (Hrsg.) ; Dipanda, Albert (Hrsg.) ; Yétongnon, Kokou (Hrsg.): *Proceedings of the 1st International Conference on Signal-Image Technology and Internet-Based Systems*, Dicolor Press, 2005, S. 267–271

[Schlicht und Stuckenschmidt 2006] Schlicht, Anne ; Stuckenschmidt, Heiner: Towards Structural Criteria for Ontology Modularization. In: Haase, Peter (Hrsg.) ; Honavar, Vasant (Hrsg.) ; Kutz, Oliver (Hrsg.) ; Sure, York (Hrsg.) ; Tamilin, Andrei (Hrsg.): *Proceedings of the 1st International Workshop on Modular Ontologies*, 2006 (CEUR Workshop Proceedings 232)

[Schoop und Jertila 2004] Schoop, Mareike ; Jertila, Aida: Electronic Commerce in the Semantic Web Era. In: Bichler, Martin (Hrsg.) ; Holtmann, Carsten (Hrsg.) ; Kirn, Stefan (Hrsg.) ; Müller, Jörg P. (Hrsg.) ; Weinhardt, Christof (Hrsg.): *Coordination and Agent Technology in Value Networks, Multikonferenz Wirtschaftsinformatik*. Berlin : GITO Verlag, 2004

[Schreder u. a. 2007] Schreder, Bernhard ; Wahler, Alexander ; Linder, Markus ; Schliefnig, Martin ; Hollerer, Svetlana: Enabling Semantic Web-ready E-Commerce Solutions. In: *Proceedings of the First European Semantic Technology Conference (ESTC), CD-ROM*, 2007

[Seidenberg und Rector 2006] Seidenberg, Julian ; Rector, Alan L.: Web ontology segmentation: analysis, classification and use. In: Carr, Les (Hrsg.) ; Roure, David D. (Hrsg.) ; Iyengar, Arun (Hrsg.) ; Goble, Carole A. (Hrsg.) ; Dahlin, Michael (Hrsg.): *Proceedings of the 15th international conference on World Wide Web*, ACM, 2006, S. 13–22

[Shadbolt u. a. 2006] Shadbolt, Nigel ; Berners-Lee, Tim ; Hall, Wendy: The Semantic Web Revisited. In: *IEEE Intelligent Systems* 21 (2006), Nr. 3, S. 96–101

[Shvaiko und Euzenat 2005] Shvaiko, Pavel ; Euzenat, Jérôme: A Survey of Schema-Based Matching Approaches. In: *Journal on Data Semantics* (2005), Nr. 3730, S. 146–171

[Shvaiko u. a. 2006] Shvaiko, Pavel (Hrsg.) ; Euzenat, Jérôme (Hrsg.) ; Noy, Natalya (Hrsg.) ; Stuckenschmidt, Heiner (Hrsg.) ; Benjamins, Richard (Hrsg.) ; Uschold, Michael (Hrsg.): *Proceedings of the 1st International Workshop on Ontology Matching*. Bd. *225*. 2006

[Sirin und Parsia 2004] Sirin, Evren ; Parsia, Bijan: Pellet: An OWL DL Reasoner. In: Haarslev, Volker (Hrsg.) ; Möller, Ralf (Hrsg.): *Proceedings of the 2004 International Workshop on Description Logics*, 2004 (CEUR Workshop Proceedings 104)

[Sowa 2000] Sowa, John F.: *Knowledge Representation. Logical, Philosophical, Computational Foundations*. Pacific Grove et al. : Brooks/Cole, 2000

[Staab und Studer 2004] Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Springer, 2004 (International Handbooks on Information Systems)

[Storey u. a. 2001] Storey, Margaret-Anne ; Musen, Mark ; Silva, John ; Best, Casey ; Ernst, Neil ; Fergerson, Ray ; Noy, Natasha: Jambalaya: Interactive Visualization to Enhance Ontology Authoring and Knowledge Acquisition in Protégé. In: *Proceedings of Workshop on Interactive Tools for Knowledge Capture*, 2001. – Internetquelle: `http://www.cs.uvic.ca/~mstorey/papers/kcap2001.pdf`, heruntergeladen am 24.01.2008

[Stuckenschmidt u. a. 2004] Stuckenschmidt, Heiner ; Harmelen, Frank van ; Bouquet, Paolo ; Giunchiglia, Fausto ; Serafini, Luciano: Using C-OWL for the alignment and merging of medical ontologies. In: Hahn, Udo (Hrsg.): *First International Workshop on Formal Biomedical Knowledge Representation*, 2004 (CEUR Workshop Proceedings 102), S. 88–101

[Stuckenschmidt und Klein 2004a] Stuckenschmidt, Heiner ; Klein, Michel: Ontology Refinement – Towards Structure-Based Partitioning of Large Ontologies / Vrije Universiteit Amsterdam. Juni 2004. – Forschungsbericht. Deliverable D22 des WonderWeb-Projekts. Internetquelle: `http://wonderweb.semanticweb.org/deliverables/documents/D22.pdf`, heruntergeladen am 18.12.2007

[Stuckenschmidt und Klein 2004b] Stuckenschmidt, Heiner ; Klein, Michel: Structure-Based Partitioning of Large Concept Hierarchies. In: (McIlraith u. a., 2004), S. 289–303

[Stuckenschmidt und Klein 2005] Stuckenschmidt, Heiner ; Klein, Michel: *Structure-based Ontology Partitioning*. 2005. – Internetquelle: `http://swserver.cs.vu.nl/partitioning/`, heruntergeladen am 8.1.2008

[Sun Microsystems 2007] Sun Microsystems: *JXTA Java™Standard Edition v2.5: Programmers Guide*. Sep 2007. – Internetquelle: `https://guest@jxta-guide.dev.java.net/svn/jxta-guide/trunk/src/guide_v2.5/JXSE_ProgGuide_v2.5.pdf`

[Sycara und Paolucci 2004] Sycara, Katia ; Paolucci, Massimo: *Ontologies in Agent Architectures*. Kap. 17, S. 343–363. Siehe (Staab und Studer, 2004)

[Tochtermann und Maurer 2006] Tochtermann, Klaus ; Maurer, Hermann: *Semantic Web – Geschichte und Ausblick einer Vision*. S. 2–6. Siehe (Pellegrini und Blumauer, 2006)

[Tzitzikas und Hainaut 2006] Tzitzikas, Yannis ; Hainaut, Jean-Luc: On the visualization of large-sized ontologies. In: Celentano, Augusto (Hrsg.): *Proceedings of the working conference on Advanced visual interfaces*, ACM Press, 2006, S. 99–102

[Unicode Consortium 2007] The Unicode Consortium: *What is Unicode?* Juni 2007. – Internetquelle: `http://www.unicode.org/standard/WhatIsUnicode.html`, heruntergeladen am 18.11.2007

[United States National Library of Medicine 2007] United States National Library of Medicine: *Medical Subject Headings.* 2007. – Internetquelle: `http://www.nlm.nih.gov/mesh/`, heruntergeladen am 13.9.2007

[Verbeke u. a. 2002] Verbeke, Jerome ; Nadgir, Neelakanth ; Ruetsch, Greg ; Sharapov, Ilya: Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. In: Parashar, Manish (Hrsg.): *GRID '02: Proceedings of the Third International Workshop on Grid Computing*, Springer, 2002 (LNCS 2536), S. 1–12

[Vossen 1999] Vossen, Piek: EuroWordNet General Document. Deliverable D032D033/2D014, Part A1 / University of Amsterdam. 1999. – Forschungsbericht

[W3C 2001] Connolly, Dan ; van Harmelen, Frank ; Horrocks, Ian ; McGuinness, Deborah L. ; Patel-Schneider, Peter F. ; Stein, Lynn A.: *DAML+OIL (March 2001) Reference Description.* Dezember 2001. – Internetquelle: `http://www.w3.org/TR/daml+oil-reference`, heruntergeladen am 5.12.2007

[W3C 2004a] W3C: *Frequently Asked Questions on W3C's Web Ontology Language (OWL).* Februar 2004. – Internetquelle: `http://www.w3.org/2003/08/owlfaq.html`, heruntergeladen am 22.11.2007

[W3C 2004b] Martin, David ; Burstein, Mark ; Hobbs, Jerry ; Lassila, Ora ; McDermott, Drew ; McIlraith, Sheila ; Narayanan, Srini ; Paolucci, Massimo ; Parsia, Bijan ; Payne, Terry ; Sirin, Evren ; Srinivasan, Naveen ; Sycara, Katia: *OWL-S: Semantic Markup for Web Services.* November 2004. – Internetquelle: `http://www.w3.org/Submission/OWL-S/`, heruntergeladen am 5.1.2008

[W3C 2004c] Dean, Mike ; Schreiber, Guus ; Bechhofer, Sean ; van Harmelen , Frank ; Hendler, Jim ; Horrocks, Ian ; McGuinness, Deborah L. ; Patel-Schneider, Peter F. ; Stein, Lynn A.: *OWL Web Ontology Language Reference.* Februar 2004. – Internetquelle: `http://www.w3.org/TR/owl-ref/`, heruntergeladen am 18.11.2007

[W3C 2004d] Dan Brickley, R.V. G.: *RDF Vocabulary Description Language 1.0: RDF Schema.* Februar 2004. – Internetquelle: `http://www.w3.org/TR/rdf-schema/`

[W3C 2004e] Beckett, Dave: *RDF/XML Syntax Specification (Revised).* Februar 2004. – Online-Quelle: `http://www.w3.org/TR/rdf-syntax-grammar/`, heruntergeladen am 18.11.2007

[W3C 2004f] Klyne, Graham ; Carroll, Jeremy J.: *Resource Description Framework (RDF): Concepts and Abstract Syntax.* Februar 2004. – Internetquelle: `http://www.w3.org/TR/rdf-concepts/`, heruntergeladen am 18.11.2007

[W3C 2005a] Hawke, Sandro: *Rule Interchange Format. Working Group Charter.* November 2005. – Internetquelle: `http://www.w3.org/2005/rules/wg/charter`, heruntergeladen am 18.11.2007

[W3C 2005b] Lausen, Holger ; Polleres, Axel ; Roman, Dumitru ; de Bruijn , Jos ; Bussler, Christoph ; Domingue, John ; Fensel, Dieter ; Hepp, Martin ; Keller, Uwe ; Kifer, Michael ; König-Ries, Birgitta ; Kopecky, Jacek ; Lara, Rubén ; Lausen, Holger ; Oren, Eyal ; Polleres, Axel ; Roman, Dumitru ; Scicluna, James ; Stollberg, Michael: *Web Service Modeling Ontology (WSMO).* Juni 2005. – Internetquelle: `http://www.w3.org/Submission/WSMO/`, heruntergeladen am 5.1.2008

[W3C 2005c] Akkiraju, Rama ; Farrell, Joel ; Miller, John ; Nagarajan, Meenakshi ; Schmidt, Marc-Thomas ; Sheth, Amit ; Verma, Kunal: *Web Service Semantics - WSDL-S.* November 2005. – Internetquelle: `http://www.w3.org/Submission/WSDL-S/`, heruntergeladen am 5.1.2008

[W3C 2006a] Bray, Tim ; Paoli, Jean ; Sperberg-McQueen, C. M. ; Maler, Eve ; Yergeau, François: *Extensible Markup Language (XML) 1.0 (Fourth Edition).* August 2006. – Internetquelle: `http://www.w3.org/TR/xml/`, heruntergeladen am 18.11.2007

[W3C 2006b] Patel-Schneider, Peter F. ; Horrocks, Ian: *OWL 1.1 Web Ontology Language Overview.* Dezember 2006. – Internetquelle: `http://www.w3.org/Submission/owl11-overview/`, heruntergeladen am 24.08.2008

[W3C 2006c] Peterson, David ; Paul V. Biron, Kaiser P. ; Malhotra, Ashok ; Sperberg-McQueen, C. M.: *XML Schema 1.1 Part 2: Datatypes.* Februar 2006. – Internetquelle: `http://www.w3.org/TR/xmlschema11-2/`, heruntergeladen am 18.11.2007

[W3C 2008] Prud'hommeaux, Eric ; Seaborne, Andy: *SPARQL Query Language for RDF.* Januar 2008. – Internetquelle: `http://www.w3.org/TR/rdf-sparql-query/`, heruntergeladen am 15.01.2008

[Wandelt und Möller 2007] Wandelt, Sebastian ; Möller, Ralf: Scalability of OWL Reasoning: Role condensates. In: *Proceedings of the 3rd International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2007

[Wang u. a. 2006] Wang, Zongjiang ; Wang, Yinglin ; Zhang, Shensheng ; Shen, Ge ; Du, Tao: Matching Large Scale Ontology Effectively. In: (Mizoguchi u. a., 2006)

[Waterfeld u. a. 2008] Waterfeld, Walter ; Weiten, Moritz ; Haase, Peter: *Ontology Management Infrastructures.* Kap. 3, S. 59–87. Siehe (Hepp u. a., 2008)

[Wilkinson u. a. 2003] Wilkinson, Kevin ; Sayers, Craig ; Kuno, Harumi ; Reynolds, Dave: Efficient RDF Storage and Retrieval in Jena2. In: Cruz, Isabel F. (Hrsg.) ; Kashyap, Vipul (Hrsg.) ; Decker, Stefan (Hrsg.) ; Eckstein, Rainer (Hrsg.): *Proceedings of the First International Workshop on Semantic Web and Databases*, 2003, S. 131–150

[Witbrock 2007] Witbrock, Michael: Knowledge is more than Data. A Comparison of Knowledge Bases with Databases / Cycorp Inc. 2007. – Forschungsbericht. Internetquelle: `http://www.cyc.com/cyc/technology/whitepapers_dir/Knowledge_is_more_than_Data.pdf`, Heruntergeladen am 02.09.2007

[Witten 2005] Witten, Ian H.: *Data Mining: Practical Machine Learning Tools and Techniques.* Zweite Auflage. San Francisco : Morgan Kaufmann, 2005

[Wordnet 2007] *WordNet.* 2007. – Internetquelle: `http://wordnet.princeton.edu/`

[Zhang u. a. 2006] Zhang, Songmao ; Bodenreider, Olivier ; Golbreich, Christine: Experience in Reasoning with the Foundational Model of Anatomy in OWL DL. In: Altman, Russ B. (Hrsg.) ; Murray, Tiffany (Hrsg.) ; Klein, Teri E. (Hrsg.) ; Dunker, A. K. (Hrsg.) ; Hunter, Lawrence (Hrsg.): *Biocomputing 2006, Proceedings of the Pacific Symposium*, World Scientific, 2006, S. 200–211

[Zhang u. a. 2004] Zhang, Songmao ; Mork, Peter ; Bodenreider, Olivier: Lessons learned from aligning two representations of anatomy. In: Hahn, Udo (Hrsg.): *Proceedings of the First International Workshop on Formal Biomedical Knowledge Representation*, 2004 (CEUR Workshop Proceedings 102), S. 102–108