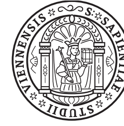


DIPLOMARBEIT

Titel der Diplomarbeit

„Evaluierung von Änderungen zwischen Ontologieversionen“

Verfasser

Günter Wildmann

angestrebter akademischer Grad

Magister der Sozial- und Wirtschaftswissenschaften (Mag. rer. soc. oec.)

Wien, 2008

Studienkennzahl lt. Studienblatt: A 175

Studienrichtung lt. Studienblatt: Wirtschaftsinformatik

Betreuer: O.Univ.-Prof. Dipl.-Ing. Dr. Johann Eder

INHALTSVERZEICHNIS

1 Einführung und Motivation 1

2 Ontologien 3
- 2.1 Aufbau von Ontologien 3
  - 2.1.1 Instanzen 4
  - 2.1.2 Eigenschaften 4
  - 2.1.3 Klassen 5
- 2.2 Versionen von Ontologien 5
  - 2.2.1 Strukturelle Änderungen 6
  - 2.2.2 Semantische Änderungen 6
- 2.3 Einsatzmöglichkeiten von Ontologien 6
  - 2.3.1 Semantic Web 7
  - 2.3.2 Semantic MediaWiki 7
  - 2.3.3 Biomedizinische Ontologien 8

3 OWL 9
- 3.1 Die drei OWL Untersprachen 9
- 3.2 Aufbau von OWL Dateien 10
  - 3.2.1 OWL-Klassen 10
  - 3.2.2 OWL-Eigenschaften 13
  - 3.2.3 OWL-Instanzen 15

4 Vergleichsverfahren 17
- 4.1 OntoCompare 17
  - 4.1.1 Datenstruktur und Operationen 17
  - 4.1.2 Einlesen einer OWL-Datei 18
  - 4.1.3 Arbeitsweise des Algorithmus 26
- 4.2 PromptDiff 29
  - 4.2.1 Protégé-Editor 29
  - 4.2.2 PromptDiff-Modul 30

5 Durchführung der Vergleiche 33
- 5.1 Ermittlung der Ergebnisdarstellung 33
  - 5.1.1 Hinzufügen einer Klasse 34
  - 5.1.2 Einfügen einer Klasse 35
  - 5.1.3 Löschen einer Klasse 36
  - 5.1.4 Umbenennen einer Klasse 36
  - 5.1.5 Einfügen einer Eigenschaft 37
  - 5.1.6 Löschen einer Eigenschaft 38
  - 5.1.7 Umbenennen einer Eigenschaft 38
  - 5.1.8 Einfügen einer Instanz 39
  - 5.1.9 Löschen einer Instanz 39
  - 5.1.10 Umbenennen einer Instanz 39
  - 5.1.11 Bewertung der Ergebnisdarstellung 40
- 5.2 Genauigkeitsprüfung 41
  - 5.2.1 Hinzufügen von Elementen 41
  - 5.2.2 Elemente hinzufügen und löschen 43
  - 5.2.3 Elemente hinzufügen, löschen und umbenennen 46
  - 5.2.4 Bewertung der Genauigkeitsprüfungen 48
- 5.3 Leistungsprüfung 49

5.3.1 Zur Leistungsprüfung verwendete Ontologien 49
5.3.2 Leistungsprüfung OntoCompare 50
5.3.3 Leistungsprüfung PromptDiff 52
5.3.4 Bewertung der Leistungsprüfung 54

6 Zusammenfassung 57

A Kurzfassung 61

B Abstract 63

C Lebenslauf 65

ABBILDUNGSVERZEICHNIS

Abbildung 2.1 Darstellung von Instanzen 4
Abbildung 2.2 Darstellung von Eigenschaften 4
Abbildung 2.3 Darstellung von Klassen 5
Abbildung 4.1 Mensch-Tier Ontologie 19
Abbildung 4.2 Zuteilung der Klassen zu Ebenen im Graph 22
Abbildung 4.3 Klassenhierarchie 23
Abbildung 4.4 OntoCompare in Pseudocode 28
Abbildung 4.5 Die Benutzerschnittstelle des Protégé-Editors 29
Abbildung 4.6 Der PromptTab 30
Abbildung 5.1 PromptDiff Ergebnis 'Elemente hinzufügen' 43
Abbildung 5.2 PromptDiff Ergebnis 'Elemente hinzufügen und löschen' 46
Abbildung 5.3 PromptDiff Ergebnis 'Elemente hinzufügen, löschen und umbenennen' 48

TABELLENVERZEICHNIS

Tabelle 4.1 Abbildung der Beschränkungen in Slots 25
Tabelle 4.2 Abhängigkeitsmatrix der heuristischen Regeln 32
Tabelle 5.1 Eckdaten der Gene-Ontologien 50
Tabelle 5.2 Eckdaten der SoPharm-Ontologien 50
Tabelle 5.3 Werte des OntoCompare-Vergleichslaufes GO_20070306 mit GO_20070305 51
Tabelle 5.4 Werte des OntoCompare-Vergleichslaufes GO_20070309 mit GO_20070305 52
Tabelle 5.5 Werte des OntoCompare-Vergleichslaufes GO_20070309 mit GO_20070306 52
Tabelle 5.6 Werte des OntoCompare-Vergleichslaufes SO_1.2 mit SO_1.0 52
Tabelle 5.7 Werte des OntoCompare-Vergleichslaufes SO_1.3 mit SO_1.0 53
Tabelle 5.8 Werte des OntoCompare-Vergleichslaufes SO_1.3 mit SO_1.2 53
Tabelle 5.9 Werte des PromptDiff-Vergleichslaufes GO_20070306 mit GO_20070305 53
Tabelle 5.10 Werte des PromptDiff-Vergleichslaufes GO_20070309 mit GO_20070305 54
Tabelle 5.11 Werte des PromptDiff-Vergleichslaufes GO_20070309 mit GO_20070306 54
Tabelle 5.12 Werte des PromptDiff-Vergleichslaufes SO_1.2 mit SO_1.0 54
Tabelle 5.13 Werte des PromptDiff-Vergleichslaufes SO_1.3 mit SO_1.0 55
Tabelle 5.14 Werte des PromptDiff-Vergleichslaufes SO_1.3 mit SO_1.2 55
Tabelle 5.15 Änderungsraten der Ontologien 55
Tabelle 5.16 Laufzeitunterschiede zwischen den Vergleichsverfahren 56

1 EINFÜHRUNG UND MOTIVATION

Der Einsatz von Ontologien ermöglicht es, Wissen zu modellieren, strukturieren und auszutauschen, um von unterschiedlichen Systemen darauf zuzugreifen. Wissen verändert sich über die Zeit, es wird ergänzt, berichtigt oder in einem neuen Zusammenhang bewertet. Die Darstellung des Wissens mittels Ontologien muß diese Veränderungen mitvollziehen[1]. Meist sind nur Versionen von Ontologien zu gewissen Zeitpunkten vorhanden, ein Änderungsverzeichnis fehlt, die Änderung der Inhalte und Strukturen ist somit unbekannt. Will man herausfinden, welche Änderungen es zwischen zwei Ontologieversionen gegeben hat, muß man mittels Vergleichsverfahren versuchen, die Änderungen nachträglich zu ermitteln[2].

Die Änderungen zwischen zwei Ontologieversionen sind meist nicht bekannt.

Für die formale Beschreibung von Ontologien haben sich, meist für den Einsatzzweck optimiert, verschiedene Sprachen entwickelt. Viele Beschreibungssprachen decken nur einen Teilbereich des möglichen Umfanges ab, ein Überführen einer Ontologie von einer Sprache in eine andere ist daher oft nicht möglich. Mit der Einführung der Web Ontology Language (OWL) wurde eine Notation geschaffen, die diese Beschränkungen umgehen soll und so eine universelle Sprache darstellt.

OWL ist eine universelle Notationssprache für Ontologien.

An österreichischen Universitäten wurde der Vergleichsalgorithmus OntoCompare[3] entwickelt, um mit Techniken, die aus dem Data-Warehousing bekannt sind, Veränderungen zwischen zwei Versionsständen einer Ontologie zu ermitteln. Die interne Darstellung einer Ontologie beruht bei diesem Verfahren auf einem gerichteten, azyklischen Graphen mit genau einem Wurzelelement. Im Rahmen dieser Arbeit wurde ein Modul entwickelt, mit dessen Hilfe man OWL-Dateien in das Datenformat von OntoCompare übersetzen kann, um somit Versionen von OWL-Ontologien miteinander vergleichen zu können. Dadurch wird es möglich, die Veränderungen in Ontologien, die im tatsächlichen Gebrauch sind, zu ermitteln.

OntoCompare ermittelt Änderungen zwischen Ontologieversionen.

Interessant dabei war die Qualität der Erkennung von Veränderungen und die Leistungsfähigkeit hinsichtlich des Umfangs der Ontologien. Die Genauigkeit der ermittelten Versionsunterschiede wurde durch gezielte Veränderungen an der Referenz OWL-Ontologie des World Wide Web Consortium ermittelt, die Leistungsfähigkeit durch ausgesuchte Ontologien größeren Umfangs.

Als Referenz wurden alle Untersuchungen auch mit dem Verfahren PromptDiff aus dem Protégé-Projekt[4] durchgeführt. Protégé ist ein freier, Open Source Ontologie-Editor und ein Knowledge-Base Framework und wurde am Stanford Center for Biomedical Informatics Research der Stanford University School of Medicine entwickelt. Es wurde die Version 3.2.1 des Protégé-OWL Editors verwendet, da die Version 4.0 erst im Laufe der Arbeit verfügbar wurde und sie sich bis zum Abschluß der Arbeit erst im Alpha-Stadium befand, in dem PromptDiff noch nicht integriert war. Erste Tests zeigten aber wesentliche Verbesserungen im

Referenzergebnisse werden mit PromptDiff ermittelt.

1 Siehe dazu [12]
2 Siehe dazu [15]
3 Siehe dazu [5]
4 Protege Web-Site `http://protege.stanford.edu/`

Umgang mit OWL-Dateien, so werden diese in Version 4.0 zum Beispiel nicht mehr verändert, wenn man sie abspeichert.

Bei den Ontologieversionen handelt es sich um ein und dieselbe Ontologie, die zu zwei verschiedenen Zeitpunkten (entspricht zwei verschiedenen Wissensständen) beobachtet wird. Es werden nicht zwei von unterschiedlichen Quellen erzeugte Ontologien, die den selben Wissensbereich modellieren, verglichen (sogenanntes merging)[5].

Als Beurteilungskriterien für die Algorithmen werden die Erkennungsrate und die Bearbeitungsgeschwindigkeit herangezogen, wobei jedes Kriterium für sich ermittelt und keine Gewichtung vorgenommen wird.

5 Siehe dazu [14]

2 ONTOLOGIEN

Ontologie bedeutet aus dem Griechischen frei übersetzt *"Lehre von den Ordnungs-, Begriffs- und Wesensbestimmungen des Seienden"*[1]. In der Informatik verwendet man Ontologien im Bereich der Wissensverarbeitung und versteht darunter eine explizite, formale Spezifikation einer gemeinsamen Konzeptualisierung[2](Begriffsbestimmung). Formal ist eine Ontologie die Aussage einer logischen Theorie[3].

„An ontology is an explicit specification of a conceptualization." T. Gruber

Ontologien definieren Ausdrücke, mit deren Hilfe Wissen beschrieben und dargestellt werden kann. Sie werden von Personen, Datenbanken und Applikationen verwendet, um Wissen über ein bestimmtes Gebiet (z.B. Automobil-Reparatur, Medizin, Finanzwesen) zu teilen. Diese Notation ist für Computer verarbeitbar und ermöglicht es, die Repräsentationen für einzelne Gebiete auch gebietsübergreifend zu vernetzen.

Im Gegensatz zur Taxonomie, die einfache Hierarchien verwendet, verkörpert eine Ontologie ein Netz von Hierarchien, in dem Informationen über logische Beziehungen miteinander verknüpft sind. Diese Beziehungen beruhen auf Eigenschaften, die den Informationen spezifisch zugewiesen werden. Miteinander in Beziehung stehende Elemente sind semantisch verknüpft. Entscheidend dabei ist, daß auf die Informationen Inferenz- und Integritätsregeln definiert werden, die es ermöglichen, den Informationsbestand auf seine Gültigkeit zu prüfen und Schlußfolgerungen daraus abzuleiten.

Aus einer Ontologie können dadurch Fakten abgeleitet werden, die nicht explizit darin aufgeführt sind, wobei diese Ableitung auf einem oder mehreren, verteilten Dokumenten beruhen kann.

2.1 AUFBAU VON ONTOLOGIEN

Ähnlich einer Datenbank, die aus Schema und Inhalt besteht, bestehen Ontologien aus Klassenhierarchien und Instanzen mit zugehörigen Eigenschaften[4]. Zusätzlich besitzen Ontologien auch eine formale Beschreibung der Daten sowie Regeln über deren Zusammenhang, der Beziehungen der Elemente zueinander. Im traditionellen Sinne bestehen Ontologien aus *classes* (Klassen), *slots* (Eigenschaften) mit ihren *slot restrictions* (Beschränkungen, Regeln die auf Eigenschaften definiert sind) und *instances* (Instanzen)[5]. Die Regeln, die auf Eigenschaften definiert sind, erlauben es, Schlüsse aus den Daten zu ziehen, neue Daten abzuleiten und die Gültigkeit von Daten zu überprüfen. Daraus ergibt sich aber auch ein wesentlicher Unterschied zu Datenbanken: Bei Ontologien ist das Schema bereits Dateninhalt und wird bei einer Abfrage als Teil des Ergebnisses zurückgeliefert[6].

Ontologien bestehen aus Klassen, Instanzen und Eigenschaften.

1 [4, S. 551]
2 Vgl. [8, S. 1f.]
3 [9, S. 2]
4 Vgl. [10, S. 12f.]
5 Siehe dazu [1]
6 Vgl. [14, S. 3]

2.1.1 *Instanzen*

Eine Instanz ist die konkrete Ausprägung einer Klasse.

Instanzen repräsentieren Objekte oder Subjekte in einer Ontologie (z.B. Wien, Tisch, Sachbearbeiter). Instanzen werden auch als Individuen[7] bezeichnet und sind eine konkrete Ausprägung einer Klasse (ein Objekt ist die Instanz einer Klasse).

Abbildung 2.1: Darstellung von Instanzen

2.1.2 *Eigenschaften*

Eigenschaften setzen Instanzen zueinander in Beziehung.

Eigenschaften[8] sind binäre Relationen zwischen zwei Instanzen, setzen somit zwei Ausprägungen von Klassen miteinander in Beziehung.

In Erweiterung des Beispiels der Instanzen kann man eine weitere Instanz *Kärnten* und eine weitere Instanz *Österreich* einführen und für die Instanzen *Kärnten* und *Klagenfurt* die Eigenschaft *ist Hauptstadt von* und für *Kärnten* und *Österreich* die Eigenschaft *liegt in* definieren.

Eigenschaften können die folgenden Charakteristiken aufweisen:

- funktional
- transitiv
- symmetrisch

Über diese Charakteristiken können die Eigenschaften näher beschrieben beziehungsweise beschränkt werden.

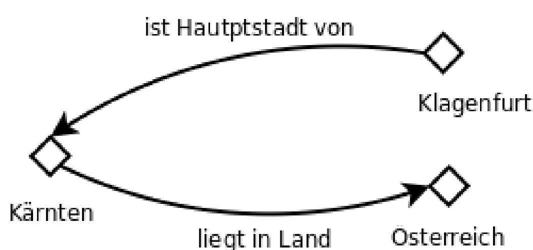


Abbildung 2.2: Darstellung von Eigenschaften

7 englisch *Individuals*
8 englisch *Properties*

2.1.3 *Klassen*

Klassen[9] sind als eine Menge an Instanzen zu verstehen, wobei die Instanzen gewisse Kriterien erfüllen müssen, um einer Klasse anzugehören. So könnten die Städte Wien und Klagenfurt Instanzen einer Klasse *Hauptstadt* sein. Klassen können in einer Hierarchie organisiert sein (Taxonomie), wobei die Unterklassen die Eigenschaften der Oberklasse erben und Mitglieder aller Oberklassen sind. Wenn man die Klasse *Hauptstadt* als Unterklasse der Klasse *Stadt* definiert, dann sind die Instanzen *Wien* und *Klagenfurt* auch Instanzen der Klasse *Stadt*. Zusätzlich zu hierarchischen Anordnungen können weitere Einschränkungen für Unterklassen definiert werden. So gibt es etwa die Möglichkeit, alle Unterklassen als bedeckend zu definieren und damit auszusagen, daß jede Instanz Mitglied einer der Unterklassen sein muss, die Basisklasse kann also keine Instanzen besitzen. Klassen werden über Bedingungen definiert, die erfüllt sein müssen, um Mitglied dieser Klasse zu sein.

Klassen fassen gleichartige Objekte zusammen.

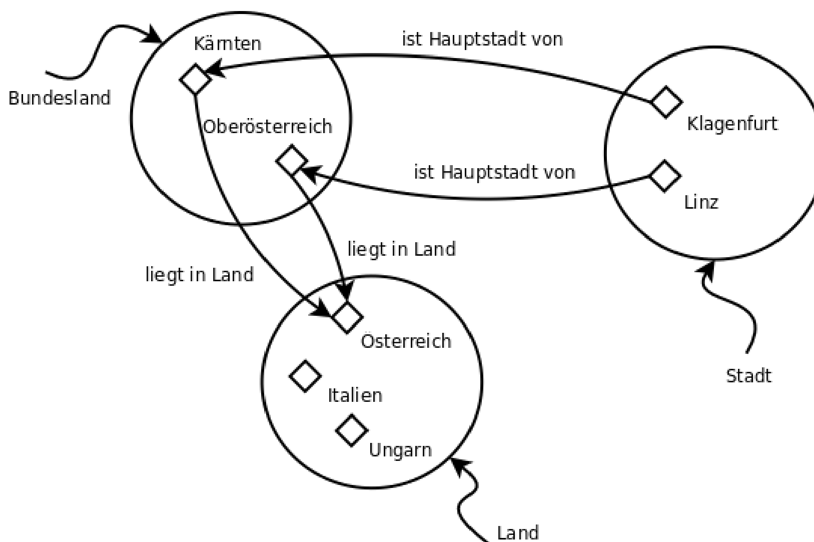


Abbildung 2.3: Darstellung von Klassen

2.2 VERSIONEN VON ONTOLOGIEN

Ontologien stellen Wissen aus Bereichen der realen Welt dar. So wie sich die Welt ändert, sind auch die sie beschreibenden Ontologien Änderungen unterworfen. Üblicherweise verfügt man über verschiedene Versionen der Ontologien, die einer Momentaufnahme zu einem gewissen Zeitpunkt entsprechen, die Art und die Menge der Änderungen dazwischen sind aber meist unbekannt. Oft ist es aber für die korrekte Interpretation der Information hilfreich oder sogar notwendig, die Änderungen zwischen den einzelnen Versionen zu kennen. Wenn man zum Beispiel die gemessene Rechenleistung heutiger Computersysteme mit jener der Systeme von vor 30 Jahren vergleichen möchte, so hat sich in dieser Zeit sowohl die Art der Computer als auch die Methode der Leistungsermittlung geändert. Für einen aussagekräftigen Vergleich ist das Wissen um diese Änderungen notwendig.

9 englisch *Classes*

Heutige Ontologien sind oft so groß, daß sie nicht mehr von einem Autor alleine bearbeitet werden können, sondern Änderungen und Erweiterungen von mehreren Personen eingepflegt werden müssen. Oft befinden sich die Bearbeiter an unterschiedlichen Orten und stellen die geänderten Versionen mittels Computernetzwerk zur Verfügung. Auch in diesem Fall ist es von Interesse zu sehen, welche Änderungen an einer Ontologie durchgeführt wurden. Im Grunde handelt es sich bei dieser Anforderung um Versionsmanagement.

Die Problematiken des Versionsmanagements sind aus vielen Bereichen der Dokumentenverwaltung im Allgemeinen und der Datenbanken- und Softwareerstellung im Besonderen bekannt. Wohingegen bei Dokumenten oder Programm-Codes sich eine Analyse auf Dateiinhalte beschränkt, sind bei Ontologien auch die semantischen Aspekte zu berücksichtigen. Die große Herausforderung ist es, herauszufinden, ob sich zwischen zwei betrachteten Ontologie-Versionen Bestehendes geändert hat oder ob Elemente hinzu- oder weggekommen sind.

Mit Versionen von Ontologien sind dabei Änderungen an einer bestimmten Ontologie über einen Zeitverlauf gemeint, die Unterschiede zwischen Ontologien zum selben Wissensgebiet aber, erstellt von verschiedenen Autoren und somit leicht unterschiedlich modelliert[10], das sogenannte Ontologie-Mapping[11], werden nicht betrachtet.

2.2.1 *Strukturelle Änderungen*

Änderungen oder Erweiterungen an den Klassen, den Instanzen oder an den Eigenschaften einer Ontologie werden als strukturelle Änderungen bezeichnet[12]. Es kann sich dabei um zusätzliche Klassen in der Klassenhierarchie handeln, mit deren Hilfe Wissen expliziter dargestellt werden kann, um Änderungen bei Instanzen, die Veränderungen des Wissens repräsentieren oder um Änderungen bei den Eigenschaften, die Zusammenhänge neu oder anders definieren.

2.2.2 *Semantische Änderungen*

Bei einer semantischen Änderung ändert sich die Bedeutung einer Ontologie. Wenn eine semantische Änderung nicht auch eine strukturelle Änderung nach sich zöge, bedeutete das, daß zwei strukturell gleiche Ontologien verschiedene Bedeutungen haben. Sehr wohl kann aber eine rein strukturelle Änderung keinerlei semantische Änderung bedeuten. Als Beispiel sei hier die strukturelle Änderung einer Ontologie zur Performance-Verbesserung genannt.

2.3 EINSATZMÖGLICHKEITEN VON ONTOLOGIEN

Bei Modellierungsaufgaben ist es im Allgemeinen so, daß von verschiedenen Autoren erstellte Modelle unterschiedliche Bezeichnungen und leicht unterschiedlichen Aufbau aufweisen, obwohl sie die selbe Gegebenheit beschreiben. Mittels Ontologien kann man ein Netzwerk aus diesen Modellen schaffen und einheitlich auf die Informationen zugreifen.

10 Siehe dazu [7]
11 Siehe dazu [3]
12 Siehe dazu [21]

Reine Informationssammlungen, wie sie etwa das World Wide Web darstellt, bieten zwar Zugriff zu einer Fülle an Informationen, die Suche nach bestimmten Inhalten ist allerdings nur auf Grundlage von Informationsfragmenten, meist Wörtern, möglich. So kann eine Suche nach dem Wort *Jaguar* sowohl Treffer, die sich auf das Tier, als auch solche, die sich auf das Automobil oder Kampfflugzeug beziehen, liefern. Durch die Verwendung von Ontologien werden die Informationen auch semantisch verbunden und sind danach über Bedeutungskriterien auffindbar. Mit diesen Möglichkeiten lassen sich Ontologien gut für Applikationen wie etwa semantische Suchmaschinen, Expertensysteme, Spracherkennungssysteme, Wissensmanagement und intelligente Datenbanken einsetzen.

Wissen wird mittels Ontologien so allgemein dargestellt, daß es nicht nur in einem bestimmten Anwendungsfall verwendet werden kann, sondern daß eine Vielzahl an Applikationen Schlußfolgerungen aus den Daten ziehen können.

2.3.1 *Semantic Web*

Grundgedanke von Tim Berners-Lee, dem Begründer des World Wide Web[13], war es, zusätzlich zur Informationsrepräsentation für Menschen eine Repräsentation für Maschinen zu erstellen. Zusätzlich zur Indexierung der Seiten über den Inhalt sollte auch eine Indexierung über deren Bedeutung durchgeführt werden. Für diesen Zweck wird auf Ontologien zurückgegriffen, die mittels RDF[14] oder OWL[15] formuliert sind.

Ontologien repräsentieren Wissen eines Wissensbereiches, das Semantic Web[16] will einen Schritt weiter gehen und Wissen aus allen Wissensbereichen miteinander verknüpfen. Die Strukturen der Ontologien werden dabei zusätzlich zu konventionellen Webinhalten aufgebaut.

2.3.2 *Semantic MediaWiki*

Semantic MediaWiki[17] erweitert die Enzyklopädie-Anwendung MediaWiki um die Möglichkeit semantische Auszeichnungen anzuführen.

Dadurch sollen folgende Ziele erreicht werden:

- Datenlisten sollen automatisiert werden, da deren manuelle Pflege aufwändig und fehleranfällig ist.
- Die Suchmöglichkeiten und die Trefferquote sollen verbessert werden.
- Die geradezu inflationäre Verwendung von Kategorien soll vermindert werden.
- Die Konsistenz der Daten in verschiedenen Sprachen soll verbessert werden.
- Die externe Wiederverwendung von Informationen soll ermöglicht werden.

---

13 Siehe dazu Historische Beschreibung des World Wide Web `http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html`

14 Siehe dazu [11]

15 Siehe dazu OWL Web Ontology Language `http://www.w3.org/TR/owl-features/`

16 Siehe dazu W3C Semantic Web Activity `http://www.w3.org/2001/sw/`

17 Siehe dazu Semantic MediaWiki `http://ontoworld.org/wiki/Semantic_MediaWiki`

Diese Verwendung von Ontologien in Online-Enzyklopädien kann als erster Teil eines Semantic Web gesehen werden. Eine Realisierung einer Semantic Web-fähigen Web-Site hat einer der Entwickler des Semantic MediaWiki auf Ontoworld[18] dargestellt.

2.3.3 *Biomedizinische Ontologien*

Neben dem Aufbau des Semantic Web werden derzeit besonders im biomedizinischen Bereich Ontologien eingesetzt. Sehr geschätzt werden dabei die Möglichkeiten, Daten aus verschiedenen Quellen zusammenführen zu können und die Möglichkeiten der zielgenauen Suche in großen Datenbeständen.

Die Wissensbereiche spannen sich dabei von der Antropologie über medizinische Befunde bis zu Enzymen.

Ein sehr umfangreiches Projekt stellt die Gen-Ontologie[19] Datenbank dar.

18 Siehe dazu Ontoworld `http://ontoworld.org/`
19 Siehe dazu Gene Ontology Project `http://www.geneontology.org/`

3 OWL

Für den Einsatz von Ontologien wurden diverse Notationssprachen entwickelt, um die Elemente beschreiben zu können. Viele davon waren speziell auf ihren Einsatzzweck hin optimiert, manche haben sich als Quasi-Standard etabliert. Allen gemein ist, daß sie nicht mit der Architektur des World Wide Web im Generellen und mit dem Semantic Web im Speziellen kompatibel sind. OWL wurde vom W3C[1] entwickelt, um einen Standard zu schaffen und die Unzulänglichkeiten anderer Notationssprachen zu umgehen. Die Web Ontology Language erweitert Ontologien um folgende Möglichkeiten:

Ontologien können in verschiedenen Sprachen notiert werden. OWL soll einen einheitlichen Standard schaffen.

- Möglichkeit zur Verteilung auf viele Systeme
- Skalierbarket für WWW-Anwendungen
- Kompatibilität mit WWW-Standards für einfachen Zugang und Internationalisierung
- Offenheit und Erweiterbarkeit

Technisch basiert OWL auf RDF[2], historisch gesehen ist es eine Weiterentwicklung von DAML+OIL[3] welches seinerseits eine Kombination von DAML-ONT (Darpa Agent Markup Language - Ontology) und OIL (Ontology Inference Layer) ist.

Der hohe Grad der Anerkennung der OWL als neuer Standard läßt sich unter anderem daran erkennen, daß für die meisten übrigen Formate Hilfsprogramme existieren, um sie in OWL zu übersetzen.

3.1 Die drei OWL Untersprachen

OWL ist in drei Sprachebenen mit unterschiedlicher Komplexität definiert[4]:

Die Komplexität kann bei OWL in drei Stufen gewählt werden.

OWL Lite hat die geringste Komplexität und zielt auf Benutzer ab, die nur eine Klassifikations-Hierarchie mit einfachen Beschränkungen erstellen wollen. Es sollte einfach sein, Taxonomien nach OWL Lite zu transformieren, auch die Erstellung von Programm-Werkzeugen sollte durch die geringere Komplexität erleichtert sein.

OWL DL zielt auf Anwendungen, bei denen maximale Möglichkeiten bei vollständiger Inferenz gefordert sind. Es sind alle OWL Konstrukte enthalten, teilweise ist deren Einsatz aber beschränkt, um die Möglichkeit der Schlußfolgerung zu erhalten. Der Namenszusatz DL deutet auf den Zusammenhang mit Description Logics hin.

1 Siehe dazu W3C `http://www.w3.org`
2 Siehe dazu RDF `http://www.w3.org/RDF/`
3 Siehe dazu DAML+OIL `http://www.daml.org`
4 Vgl. [13]

OWL Full bietet alle Freiheiten die RDF bereitstellt, wodurch jedoch eine vollständige Berechenbarkeit nicht mehr garantiert werden kann.

Jede höhere Sprachebene ist eine vollständige Erweiterung der darunterliegenden Ebene, sowohl in Hinsicht dessen, was sie ausdrücken kann, als auch in der Möglichkeit der Schlußfolgerung.

- Jede valide OWL Lite Ontologie ist eine valide OWL DL Ontologie.
- Jede valide OWL DL Ontologie ist eine valide OWL Full Ontologie.
- Jede valide OWL Lite Schlußfolgerung ist eine valide OWL DL Schlußfolgerung.
- Jede valide OWL DL Schlußfolgerung ist eine valide OWL Full Schlußfolgerung.

OWL Full kann als Erweiterung von RDF angesehen werden, wogegen OWL Lite und OWL DL als Erweiterung einer eingeschränkten Sicht auf RDF angesehen werden können. Jedes OWL Dokument ist ein RDF Dokument und jedes RDF Dokument ist ein OWL Full Dokument, allerdings sind nur manche RDF Dokumente valide OWL Lite und OWL DL Dokumente.

3.2 AUFBAU VON OWL DATEIEN

OWL-Dokumente definieren Klassen, Eigenschaften und Instanzen.

Ein OWL-Dokument besteht aus[5] Deklarationen von Namensräumen, einem fakultativen Kopfteil sowie aus Klassen (Class), Eigenschaften (Property) und Instanzen (Individual). Die Reihenfolge der OWL-Sprachelemente eines OWL-Dokumentes ist dabei nicht maßgebend. Eine OWL-Klasse beschreibt verschiedene Resourcen mit ähnlichen Eigenschaften. Mittels Klassen-Erweiterungen (Class-Extension) können Gruppen von Instanzen mit einer OWL-Klasse verbunden werden, wobei eine Instanz einer Klassen-Erweiterung die Instanz einer Klasse darstellt. Eigenschaften dienen zum Verbinden zweier Instanzen oder zum Binden eines Datenwertes an eine Instanz.

Die nachfolgenden Erklärungen der einzelnen Elemente verwenden Ausschnitte aus einer Referenz-Ontologie des W3C, der Wine-Ontology[6] [7].

3.2.1 *OWL-Klassen*

Es gibt zwei Arten von Konstruktoren einer Klasse:

- Klassen-Beschreibung
- Klassen-Axiom

5 Siehe dazu [20]
6 Wine-Ontology `http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf`
7 Eine graphische Darstellung der Ontologie findet sich auf: Grafik der Wine-Ontology `http://mysite.verizon.net/jflynn12/VisioOWL/VisioOWL.htm`

OWL-Klassen-Beschreibung

Die Klassen-Beschreibung gibt der Klasse den Namen oder spezifiziert die Erweiterung einer anonymen Klasse, man unterscheidet folgende Arten von Klassen-Beschreibungen[8]:

Bezeichner einer Klasse: Eine Klasse kann als benannte Klasse definiert werden.
Für die verwendete Bespieldomäne für Weinsorten sind drei Basisklassen definiert: *Winery*, *Region* und *ConsumableThing*.

```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>
```

Aufzählung der Instanzen, die die Klassen-Erweiterung zusammenstellen. Eine Klasse wird durch direkte Aufzählung ihrer Mitglieder spezifiziert. Diese Definition ist spezifiziert, die Klassenerweiterung komplett, es können keine weiteren Instanzen als dieser Klasse zugehörig deklariert werden.
Der folgende Ausschnitt definiert eine Klasse *WineColor* deren Mitglieder die Instanzen *White*, *Rose* und *Red* sind.

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#White"/>
    <owl:Thing rdf:about="#Rose"/>
    <owl:Thing rdf:about="#Red"/>
  </owl:oneOf>
</owl:Class>
```

Durchschnitt, Vereinigung und Komplementärmenge von zwei oder mehreren Klassen-Beschreibungen. Diese Operationen sind äquivalent den Operationen AND, OR und NOT in der Prädikatenlogik. Die Mitglieder einer Klasse sind ausschließlich durch die Mengenoperation spezifiziert. Das folgende Beispiel definiert, daß *WhiteWine* genau die Schnittmenge der Klasse *Wine* und der Menge der Dinge, die weiße Farbe haben, darstellt.

```
<owl:Class rdf:ID="WhiteWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#White" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Mittels *unionOf* wird festgelegt, daß die Klasse *Fruit* sowohl *SweetFruit* als auch *NonSweetFruit* einschließt.

```
<owl:Class rdf:ID="Fruit">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SweetFruit" />
    <owl:Class rdf:about="#NonSweetFruit" />
  </owl:unionOf>
</owl:Class>
```

8 Vgl. Bartroli, Alex: OWL-Sprachelemente `http://www.informatik.uni-leipzig.de/~auer/teaching/WS04WiReSW` [Stand: 24.04.2008]

Die Komplementärmenge wählt alle Instanzen aus der fraglichen Domäne aus, die nicht zu einer bestimmten Klasse gehören. Alle Instanzen, die nicht zur Klasse *ConsumableThing* gehören, werden der Klasse *NonConsumableThing* zugeordnet.

```
<owl:Class rdf:ID="ConsumableThing" />

<owl:Class rdf:ID="NonConsumableThing">
  <owl:complementOf rdf:resource="#ConsumableThing" />
</owl:Class>
```

Beschränkungen von Eigenschaften beschreiben eine anonyme Klasse, welche alle Instanzen, die die Einschränkungen erfüllen, beinhaltet.
Werte-Beschränkungen schränken den Wertebereich einer Eigenschaft ein, sobald diese Eigenschaft in einer Klassen-Beschreibung verwendet wird.
Das nachfolgende Beispiel definiert eine unbenannte Klasse, welche eine Menge von Dingen mit mindestens einer Eigenschaft *madeFromGrape* darstellt.

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#madeFromGrape"/>
  <owl:minCardinality rdf:datatype=
      "&xsd;nonNegativeInteger">
    1
  </owl:minCardinality>
</owl:Restriction>
```

Kardinalitäts-Beschränkungen definieren dabei die Anzahl an Werten, die eine Eigenschaft haben kann.

OWL-Klassen-Axiom

Klassen-Axiome enthalten zusätzliche Sprachelemente, die zusätzliche oder benötigte Informationen über eine Klasse bereitstellen. Die drei verwendeten Sprachelemente sind:

rdfs:subClassOf definiert die Klassen-Erweiterung einer Klassen-Beschreibung als Subset der Klassen-Erweiterung einer anderen Klassen-Beschreibung.
Nachfolgend wird *PotableLiquid*[9] als eine Unterklasse von *ConsumableThing* definiert.

```
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" />
  ...
</owl:Class>
```

owl:equivalentClass definiert die Klassen-Erweiterung einer Klassen-Beschreibung als gleich der Klassen-Erweiterung einer anderen Klassen-Beschreibung.
Die Definition der Klasse *Wine* in einer Essens-Ontologie wird äquivalent zu einer bereits bestehenden Klasse in der Wine-Ontology gesetzt.

```
<owl:Class rdf:ID="Wine">
  <owl:equivalentClass rdf:resource="&vin;Wine"/>
</owl:Class>
```

9 Flüssigkeiten, die trinkbar sind

owl:disjointWith besagt, daß die Klassen-Erweiterung einer Klassen-Beschreibung kein gemeinsames Mitglied mit der Klassen-Erweiterung einer anderen Klassen-Beschreibung hat.
Im nachfolgenden Beispiel wird ausgesagt, daß *Pasta* verschieden von den Klassen *Meat*, *Fowl*, *Seafood*, *Dessert* und *Fruit* ist. Zu beachten ist, daß dadurch nicht ausgesagt wird, daß *Meat* und *Fruit* voneinander verschieden sind.

```
<owl:Class rdf:ID="Pasta">
  <rdfs:subClassOf rdf:resource="#EdibleThing"/>
  <owl:disjointWith rdf:resource="#Meat"/>
  <owl:disjointWith rdf:resource="#Fowl"/>
  <owl:disjointWith rdf:resource="#Seafood"/>
  <owl:disjointWith rdf:resource="#Dessert"/>
  <owl:disjointWith rdf:resource="#Fruit"/>
</owl:Class>
```

3.2.2 *OWL-Eigenschaften*

OWL-Eigenschaften werden in zwei Kategorien unterteilt. Die Kategorie der Objekt-Eigenschaften verbindet Instanzen mit Instanzen, die Kategorie der Datentypen-Eigenschaften verbindet Instanzen mit Datenwerten.

OWL unterstützt folgende Eigenschafts-Axiom-Konstrukte:

RDF Schema Konstrukt mittels *rdfs:subPropertyOf*, *rdfs:domain* und *rdfs:range* können Eigenschaften ähnlich den Klassen in Hierarchien angeordnet werden.
Die *WineDescriptor* Eigenschaften verbinden Weine mit ihrer Farbe und ihren Geschmackskomponenten, wie Süssigkeit, Körper und Geschmack. Die Eigenschaft *hasColor* ist eine Unter-Eigenschaft der Eigenschaft *hasWineDescriptor*, welche durch *rdfs:range* auf *WineColor* begrenzt ist. Die *rdfs:subPropertyOf*-Beziehung bedeutet hier, dass alles mit einer *hasColor*-Eigenschaft mit dem Wert X, ebenfalls eine *hasWineDescriptor*-Eigenschaft mit dem Wert X besitzt.

```
<owl:Class rdf:ID="WineDescriptor" />

<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
  ...
</owl:Class>

<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range  rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource=
      "#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
  ...
</owl:ObjectProperty>
```

Logische Eigenschafts-Charakteristik mittels owl:SymmetricProperty und owl:TransitiveProperty
Die Eigenschaft *adjacentRegion* ist symmetrisch, wohingegen *locatedIn* dies nicht ist. Genauer gesagt ist dies bei

locatedIn nicht beabsichtigt. Nichts in der momentanen Wein-Ontologie untersagt, dass sie symmetrisch ist. Die *MendocinoRegion* ist der *SonomaRegion* benachbart und umgekehrt. Die *MendocinoRegion* liegt in der *CaliforniaRegion*, jedoch nicht umgekehrt.

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

<Region rdf:ID="MendocinoRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
  <adjacentRegion rdf:resource="#SonomaRegion" />
</Region>
```

Die Eigenschaft *locatedIn* ist transitiv. Da die *SantaCruzMountainsRegion* in der *CaliforniaRegion* liegt, muß sie ebenso in der *USRegion* sein.

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="&owl;Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>

<Region rdf:ID="CaliforniaRegion">
  <locatedIn rdf:resource="#USRegion" />
</Region>
```

Globale Kardinalitätsbeschränkung einer Eigenschaft *owl:functionalProperty* beschränkt eine Eigenschaft auf genau einen Wert pro Instanz. Sowohl Objekt- als auch Datentypen-Eigenschaften können als *owl:functionalProperty* deklariert werden.
Die Eigenschaft *hasVintageYear* wird als funktional definiert, da die Weinlese für jeden Wein genau in einem Jahr stattgefunden hat.

```
<owl:Class rdf:ID="VintageYear" />

<owl:ObjectProperty rdf:ID="hasVintageYear">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Vintage" />
  <rdfs:range  rdf:resource="#VintageYear" />
</owl:ObjectProperty>
```

Bei Eigenschaften, die als *owl:InversefunctionalProperty* definiert sind, gibt das Objekt der Eigenschaft eindeutig die Instanz der Eigenschaft. Invers funktionale Eigenschaften sind per Definition Objekt-Eigenschaften.
Der folgende Abschnitt besagt, daß jeder der Wein produziert ein Hersteller ist.

```
<owl:ObjectProperty rdf:ID="hasMaker" />

<owl:ObjectProperty rdf:ID="producesWine">
  <rdf:type rdf:resource=
      "&owl;InverseFunctionalProperty" />
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```

Relationen anderer Eigenschaften. Die *owl:allValuesFrom* Einschränkung verlangt, daß für jede Instanz der Klasse, welche Instanzen der spezifizierten Eigenschaft hat, die Eigenschafts-Werte alle Mitglieder der Klasse sind, welche im *owl:allValuesFrom* Abschnitt angegeben ist.
Das folgende Beispiel zeigt, daß der Hersteller von *Wine* eine *Winery* sein muß. Die *allValuesFrom* Einschränkung bezieht sich auf die *hasMaker* Eigenschaft nur dieser *Wine*-Klasse.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource=
      "&food;PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

3.2.3 *OWL-Instanzen*

Eine Möglichkeit, in OWL eine Instanz zu definieren, ist durch ihre Klassenzugehörigkeit und die Werte der Eigenschaften.
Die Region *CentralCoastRegion* wird als Instanz der Klasse *Region* definiert.

```
<Region rdf:ID="CentralCoastRegion" />
```

Die andere Möglichkeit ist eine Instanz durch ihre Identität zu definieren. Da es in OWL keine Annahmen zur Eindeutigkeit der Namen gibt, werden die Konstrukte mit einer Instanz-Identität versehen.

owl:sameAs stellt sicher, daß zwei URI-Referenzen die gleichen Instanzen referenzieren.
Beispielsweise wird *MikesFavoriteWine* mit *GenevieveTexasWhite* gleichgesetzt.

```
<Wine rdf:ID="MikesFavoriteWine">
  <owl:sameAs rdf:resource="#StGenevieveTexasWhite" />
</Wine>
```

owl:differentFrom stellt sicher, daß zwei URI-Referenzen verschiedene Instanzen referenzieren.
Das Beispiel sagt aus, daß *Sweet* unterschiedlich von *Dry* ist und daß *OffDry* unterschiedlich von *Dry* und *Sweet* ist[10].

```
<WineSugar rdf:ID="Dry" />

<WineSugar rdf:ID="Sweet">
  <owl:differentFrom rdf:resource="#Dry"/>
</WineSugar>

<WineSugar rdf:ID="OffDry">
  <owl:differentFrom rdf:resource="#Dry"/>
  <owl:differentFrom rdf:resource="#Sweet"/>
</WineSugar>
```

10 Die hier gezeigte vollständige, gegenseitige Ausschließung, kann mit *owl:distinctMembers* in Verbindung mit *owl:AllDifferent* eleganter erreicht werden.

owl:AllDifferent In Ontologien, in denen es Annahmen zur Eindeutigkeit der Instanznamen gibt, wird *owl:AllDifferent* genutzt um anzuzeigen, daß alle Instanzen paarweise disjunkt sind. Beispielsweise kann ein Rotwein kein Weißwein und kein Rose sein und auch alle andere Kombinationen sind nicht möglich.

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <vin:WineColor rdf:about="#Red" />
    <vin:WineColor rdf:about="#White" />
    <vin:WineColor rdf:about="#Rose" />
  </owl:distinctMembers>
</owl:AllDifferent>
```

4 VERGLEICHSVERFAHREN

4.1 ONTOCOMPARE

OntoCompare wurde am Institut für Knowledge and Business Engineering der Universität Wien sowie am Institut für Informatik-Systeme der Universität Klagenfurt entwickelt und basiert auf Methoden, die zum Vergleich von Datenbeständen in Data Warehouses verwendet werden[1].

4.1.1 *Datenstruktur und Operationen*

OntoCompare verwendet eine RDAG-Datenstruktur.

Die zugrundeliegende Datenstruktur ist ein gerichteter, azyklischer Graph mit einem einzigen Wurzelknoten[2][3]. Die Klassen und Instanzen der Ontologie werden als Knoten abgebildet, die semantischen Beziehungen zwischen diesen Elementen als Kanten. Jeder Knoten besitzt eine eindeutige ID, eine Bezeichnung[4], implementationsspezifische Attribute und einen Satz an Slots. Jede Kante verbindet zwei Knoten gerichtet miteinander, immer vom Elternknoten zum Kindknoten. Jede Kante ist einem Kantentyp zugehörig, wobei es zyklische und azyklische Typen gibt. Um eine Ontologie in dieser Form abbilden zu können, müssen ein virtueller Wurzelknoten hinzugefügt und Slots verwendet werden, um zyklische Abhängigkeiten aufzulösen[5]. Eine zyklische Kante wird dabei durch eine neue Kante vom virtuellen Wurzelknoten zum betreffenden Knoten und einen zusätzlichen Slot ersetzt. Auf diese Art läßt sich jede Ontologie in einen RDAG transformieren, eine eindeutige Rückführung in die ursprüngliche Ontologie ist dabei ebenfalls jederzeit möglich.

Änderungen zwischen zwei verschiedenen Ontologieversionen werden anhand der Änderungen in der Struktur und dem Inhalt der Graphen ermittelt. Folgende Operationen sind zur Anwendung auf einen RDAG definiert:

Insert Node fügt einen Knoten an einen oder mehrere Elternknoten an, wobei der Kantentyp für jede Anbindung gesondert definiert werden kann.

Delete Node löscht einen Knoten aus dem Graphen. Der zu löschende Knoten darf keine Kind-Knoten besitzen, alle Kanten die mit diesem Knoten verbunden sind, werden ebenfalls gelöscht.

Insert Edge erzeugt eine neue Kante zwischen einem Eltern- und einem Kind-Knoten, wobei der Kantentyp definiert werden kann. Durch diese Kante darf sich keine zyklische Abhängigkeit ergeben!

Delete_Edge löscht eine Kante aus dem Graphen.

---

1 Siehe dazu [2]
2 Rooted Directed Acyclic Graph, RDAG
3 Siehe dazu [6]
4 Der Name des Knoten.
5 Siehe dazu [5]

Insert Slot fügt einen Slot eines bestimmten Typs zu einem Knoten hinzu.

Delete Slot löscht einen Slot von einem Knoten.

Update Node ändert die Attribute eines Knotens.

Rename Node ändert die Bezeichnung eines Knotens.

Change Edge Type ändert den Typ einer Kante im Graph.

Mit diesen Operationen ist es möglich, jeden RDAG in einen beliebigen anderen RDAG überzuführen. OntoCompare analysiert den Ursprungsgraph und den Graph in der neuen Version und ermittelt die Art und die Reihenfolge der Operationen, um den einen Graph in den anderen umzuwandeln. Die so ermittelten Operationen repräsentieren daher die Änderungen zwischen den Ontologie-Versionen. Das Ergebnis des Vergleichsalgorithmus ist ein Edit-Skript, dessen Befehle der Reihe nach auf den Ursprungsgraph angewendet, den Zielgraph ergeben.

4.1.2 *Einlesen einer OWL-Datei*

OntoCompare wurde um die Möglichkeit des Einlesens einer OWL-Ontologie erweitert.

Ontologien, die im OWL-Format vorliegen, müssen in die Graphenstruktur umgewandelt werden, um sie mittels OntoCompare vergleichen zu können. Zu diesem Zweck wurde im Rahmen dieser Arbeit ein Modul entwickelt, welches eine OWL-Ontologie in einen geeigneten RDAG umwandelt. Für das parsing der OWL-Datei wird das Protégé-OWL API[6] verwendet. Die daraus generierten Objekte werden nach einem fixen Schema traversiert und daraus ein Graph generiert.

Die Reihenfolge dabei ist:

1. Klassen
2. Eigenschaften
3. Instanzen

Die Abbildung der Klassen erfolgt als Knoten[7] im Graph, wobei die eindeutige ID des Knotens bei seiner Erzeugung von OntoCompare generiert und die Bezeichnung der Klasse auf den mittels *getBrowserText* ausgelesenen Wert der RDF-Ressource gesetzt wird. Als Attribut wird willkürlich die Zahl Eins zugewiesen, da OWL-Ontologien keine implementationsspezifischen Klassen-Attribute benötigen. Aus der Klassenliste werden die Wurzelklassen ermittelt und der Ebene 1 im Graph zugeordnet, wobei die Klasse *owl:Thing* an die erste Stelle gesetzt wird. Es wird eine virtuelle Wurzelklasse erzeugt und daraus die oberste Ebene[8] im Graph gebildet. Für die Klassen einer Ebene werden die direkten Unterklassen gesucht und daraus die nächste Ebene gebildet. Das Ergebnis ist eine Zuteilung der Klassen zu einer Ebene im zu erzeugenden Graph.

Die Beziehungen zwischen den Klassen werden mittels Kanten[9] vom Typ *IS_A* realisiert. Zuerst wird jeweils die kürzeste Beziehung jeder

6 Siehe dazu protege-owl api `http://protege.stanford.edu/plugins/owl/api/index.html`
7 englisch: node
8 Ebene 0 bildet die oberste Ebene im Graph
9 englisch: edge

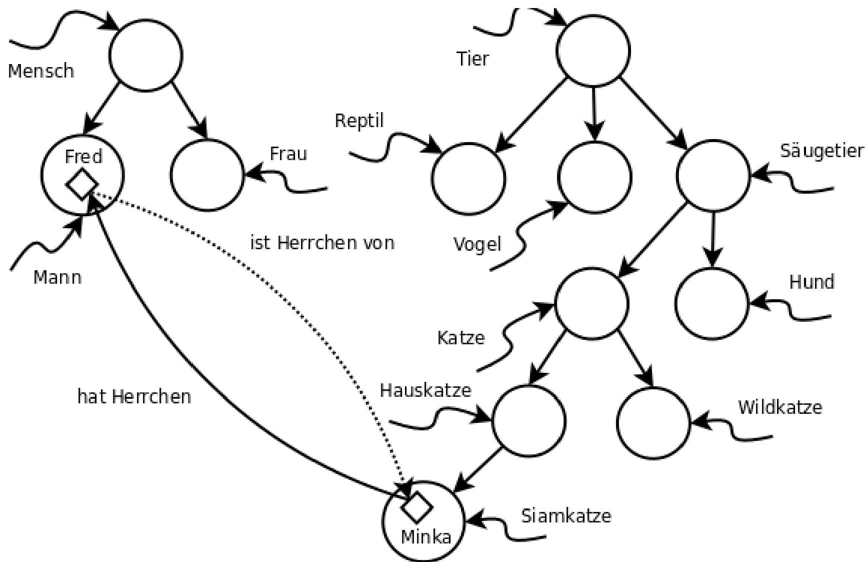


Abbildung 4.1: Mensch-Tier Ontologie

Klasse zu ihrer Mutterklasse angelegt. In der Folge werden etwaige weitere Mutter-Tochter-Klassen Beziehungen eingetragen, wobei zyklische Beziehungen durch das Hinzufügen eines Slots abgebildet werden.

Für jede Klasse werden ihre Eigenschaften mittels Slot an den Knoten im Graph angehängt. Mit den Meta-Eigenschaften wird ebenso verfahren.

Die Instanzen jeder Klasse werden als Knoten abgebildet, welche mittels Kanten vom Typ *UNDEFINED_EDGE_TYPE* zum Klassen-Knoten in Beziehung gesetzt werden. Die ID der Knoten wird durch OntoCompare automatisch erzeugt, die Bezeichnung wird auf den mittels *getName* ermittelten Namen der Instanz gesetzt. Als Attribut wird die Zahl Drei willkürlich festgelegt, um einen anderen Wert als bei Klassen zu verwenden.

Die Umwandlung einer OWL-Datei in einen RDAG anhand einer simplen Ontologie.

Die Funktionsweise der Umwandlung einer OWL-Ontologie in einen RDAG wird anhand der Beispiel-Ontologie in Abbildung 4.1 erläutert. Sie zeigt eine mögliche Klassenhierarchie für Tiere und Menschen, die zwei Instanzen, eine Siamkatze `Minka` und einen Mann *Fred*, enthält.

Es ist eine Eigenschaft `hatBesitzer` definiert, deren Domäne `Tier` und deren Wertebereich `Mensch` ist, sowie deren inverse Eigenschaft `istBesitzer` mit der Domäne `Mensch` und dem Wertebereich `Tier`. Von der Eigenschaft `hatBesitzer` sind die Eigenschaft `hatHerrchen` mit der Domäne `Tier` und dem Wertebereich `Mann` und die Eigenschaft `hatFrauchen` mit der Domäne `Tier` und dem Wertebereich `Frau` abgeleitet. Die Ableitungen der inversen Eigenschaft `istBesitzer` heißen `istHerrchen` mit der Domäne `Mann` und dem Wertebereich `Tier` und `istFrauchen` mit der Domäne `Frau` und dem Wertebereich `Tier`.

Diese Ontologie wird in OWL wie folgt notiert:

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

```
<owl:Ontology rdf:about=""/>
<owl:Class rdf:ID="Reptil">
  <owl:disjointWith>
    <owl:Class rdf:ID="Säugetier"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:ID="Vogel"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Tier"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Wildkatze">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Katze"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:ID="Hauskatze"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:ID="Mann">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Mensch"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:ID="Frau"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:ID="Siamkatze">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Hauskatze"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class/>
<owl:Class rdf:ID="Hund">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Säugetier"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#Katze"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Frau">
  <owl:disjointWith rdf:resource="#Mann"/>
  <rdfs:subClassOf rdf:resource="#Mensch"/>
</owl:Class>
<owl:Class rdf:about="#Katze">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Säugetier"/>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Hund"/>
</owl:Class>
<owl:Class rdf:about="#Hauskatze">
  <rdfs:subClassOf rdf:resource="#Katze"/>
  <owl:disjointWith rdf:resource="#Wildkatze"/>
</owl:Class>
<owl:Class rdf:about="#Säugetier">
  <owl:disjointWith>
    <owl:Class rdf:about="#Vogel"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Reptil"/>
  <rdfs:subClassOf rdf:resource="#Tier"/>
</owl:Class>
<owl:Class/>
<owl:Class rdf:about="#Vogel">
  <owl:disjointWith rdf:resource="#Säugetier"/>
  <owl:disjointWith rdf:resource="#Reptil"/>
  <rdfs:subClassOf rdf:resource="#Tier"/>
```

```
  </owl:Class>
  <owl:ObjectProperty rdf:ID="istBesitzer">
    <rdfs:range rdf:resource="#Tier"/>
    <rdfs:domain rdf:resource="#Mensch"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="hatBesitzer"/>
    </owl:inverseOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hatBesitzer">
    <rdfs:domain rdf:resource="#Tier"/>
    <rdfs:range rdf:resource="#Mensch"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hatHerrchen">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="istHerrchen"/>
    </owl:inverseOf>
    <rdfs:subPropertyOf rdf:resource="#hatBesitzer"/>
    <rdfs:range rdf:resource="#Mann"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#istHerrchen">
    <owl:inverseOf rdf:resource="#hatHerrchen"/>
    <rdfs:subPropertyOf rdf:resource="#istBesitzer"/>
    <rdfs:domain rdf:resource="#Mann"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hatFrauchen">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="istFrauchen"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#Frau"/>
    <rdfs:subPropertyOf rdf:resource="#hatBesitzer"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#istFrauchen">
    <rdfs:subPropertyOf rdf:resource="#istBesitzer"/>
    <owl:inverseOf rdf:resource="#hatFrauchen"/>
    <rdfs:domain rdf:resource="#Frau"/>
  </owl:ObjectProperty>
  <Mann rdf:ID="Fred">
    <istHerrchen>
      <Siamkatze rdf:ID="Minka">
        <hatHerrchen rdf:resource="#Fred"/>
      </Siamkatze>
    </istHerrchen>
  </Mann>
  <owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
      <Siamkatze rdf:about="#Minka"/>
    </owl:distinctMembers>
  </owl:AllDifferent>
  <owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
      <Mann rdf:about="#Fred"/>
    </owl:distinctMembers>
  </owl:AllDifferent>
</rdf:RDF>
```

Nach der Verarbeitung der Klassen ergibt sich deren Zuordnung zu Hierarchieebenen im Graph wie in Abbildung 4.2. Die virtuelle Wurzelklasse *virtual_root* wurde von OntoCompare erzeugt und bildet die einzige Klasse in der Ebene 0. Tatsächliche Wurzelklassen einer OWL-Ontologie sind unter anderem *owl:Thing* und *owl:AnonymousClass*[10], die in diesem Fall die einzigen Kind-Klassen des virtuellen Wurzelementes sind.

Schritt 1: Ermitteln der Wurzelklassen und erzeugen eines virtuellen Wurzelknotens.

10 Das Protégé-OWL API unterscheidet strikt zwischen benannten Klassen und anonymen Klassen. Benannte Klassen werden als Basis für Instanzen verwendet, während anonyme Klassen zur Spezifikation von logischen Eigenschaften (Beschränkungen) von benannten Klassen verwendet werden.

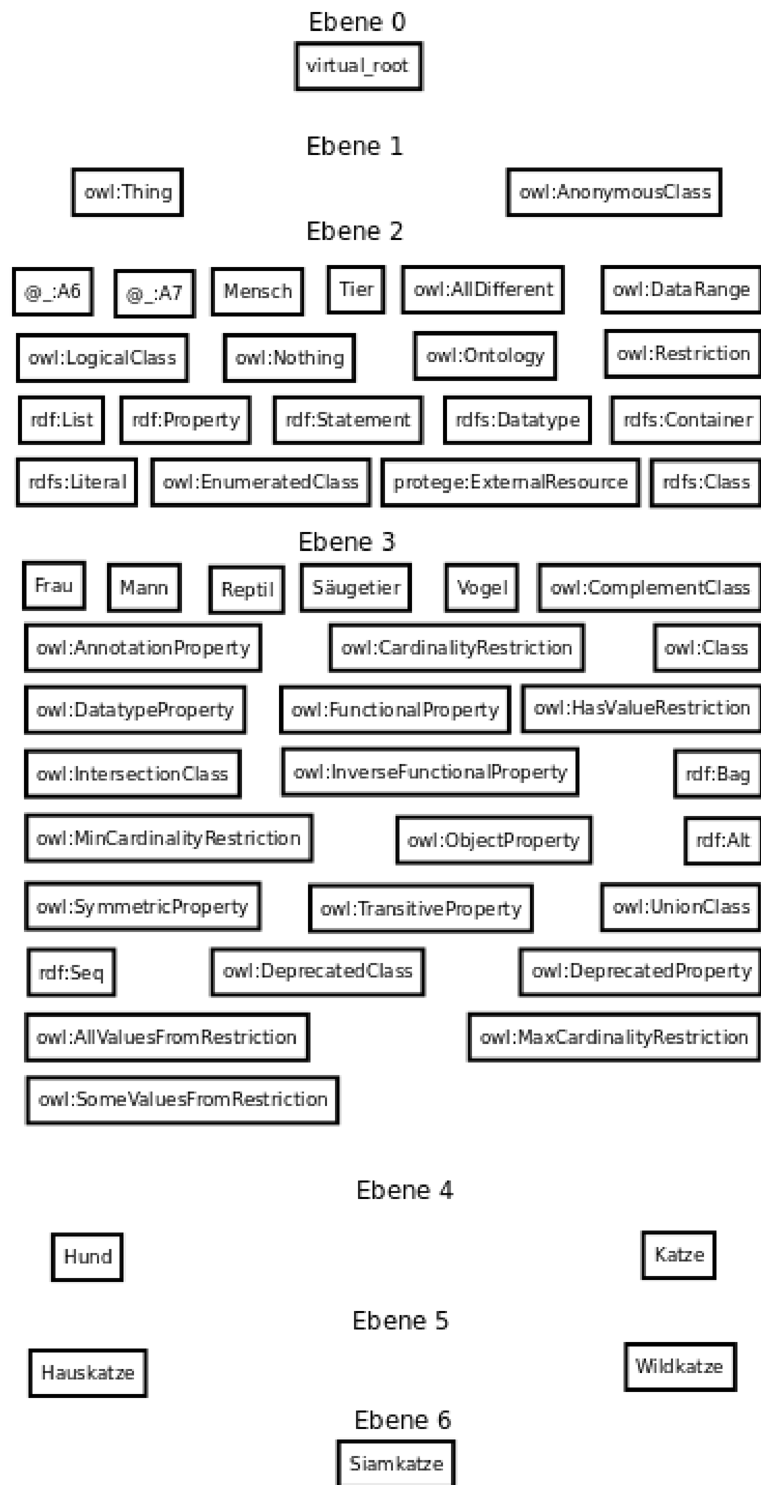


Abbildung 4.2: Zuteilung der Klassen zu Ebenen im Graph

Schritt 2: Klassenhierarchie erzeugen.

Im nächsten Schritt werden die Klassen der einzelnen Ebenen mit ihren direkten Mutterklassen verbunden. Abbildung 4.3 zeigt den sich daraus ergebenden Graph.

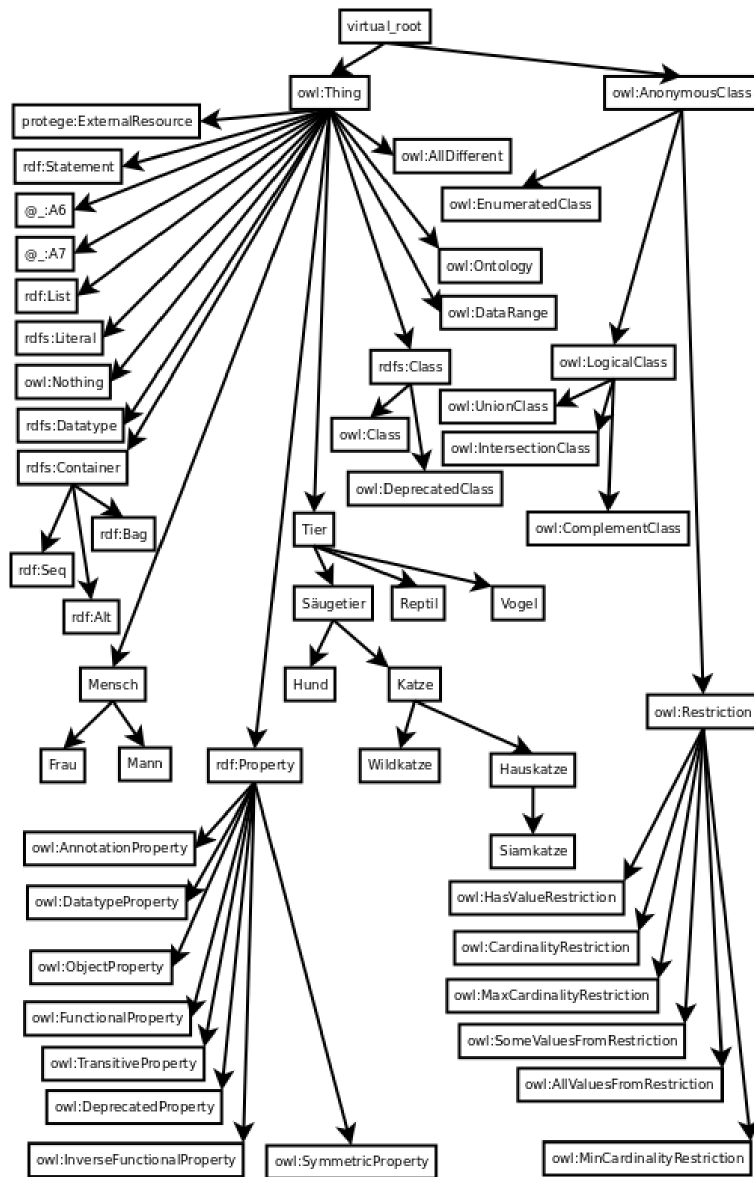


Abbildung 4.3: Klassenhierarchie

Danach werden die RDF-Property-Elemente als Knoten angelegt und mittels Kante vom Typ *IS_A* an das virtuelle Wurzelelement angefügt. Aus folgenden Eigenschaften wurden im gegenständlichen Beispiel Knoten erzeugt:

Schritt 3: Behandeln der Eigenschaften.

- rdfs:domain
- rdfs:range
- owl:inverseOf
- protege:classificationStatus
- protege:inferredSuperclassOf
- protege:inferredSubclassOf
- protege:inferredType

- owl:onProperty
- owl:allValuesFrom
- owl:hasValue
- owl:maxCardinality
- owl:minCardinality
- owl:cardinality
- owl:valuesFrom
- owl:someValuesFrom
- owl:differentFrom
- owl:sameAs
- owl:disjointWith
- owl:complementOf
- owl:intersectionOf
- owl:unionOf
- owl:equivalentProperty
- rdfs:subClassOf
- rdfs:subPropertyOf
- owl:equivalentClass
- rdf:value
- rdfs:member
- owl:imports
- rdf:first
- rdf:rest
- owl:distinctMembers
- rdf:object
- rdf:predicate
- rdf:subject
- rdf:type
- owl:oneOf
- rdfs:label
- owl:versionInfo
- rdfs:comment
- :PAL-DESCRIPTION
- :PAL-NAME

- :PAL-RANGE
- :PAL-STATEMENT
- rdfs:isDefinedBy
- rdfs:seeAlso
- owl:backwardCompatibleWith
- owl:incompatibleWith
- owl:priorVersion
- :TO
- :FROM
- istBesitzer
- hatBesitzer
- hatHerrchen
- istHerrchen
- hatFrauchen
- istFrauchen

Als nächstes werden alle Beschränkungen der Klassen abgearbeitet. Für jede Beschränkung wird dem entsprechenden Knoten im Graph ein Slot hinzugefügt, der die Referenz auf die beteiligte Klasse und die Art der Beschränkung enthält. In der Mensch-Tier-Ontologie handelt es sich ausschließlich um *owl:disjointWith* Beschränkungen gegenüber den anderen Klassen in der selben Hierarchieebene.

Schritt 4: Beschränkungen der Klassen abbilden.

Klasse	beteiligte Klasse	Beschränkung
Reptil	Säugetier	owl:disjointWith
Reptil	Vogel	owl:disjointWith
Säugetier	Vogel	owl:disjointWith
Säugetier	Reptil	owl:disjointWith
Vogel	Säugetier	owl:disjointWith
Vogel	Reptil	owl:disjointWith
Wildkatze	Hauskatze	owl:disjointWith
Hauskatze	Wildkatze	owl:disjointWith
Katze	Hund	owl:disjointWith
Hund	Katze	owl:disjointWith
Mann	Frau	owl:disjointWith
Frau	Mann	owl:disjointWith

Tabelle 4.1: Abbildung der Beschränkungen in Slots

Im Anschluß werden die Meta-Eigenschaften der Ontologie verarbeitet und analog den Beschränkungen als Slot angelegt. Die verwendete Ontologie besitzt keine Meta-Eigenschaften, es werden keine Slots angelegt.

Schritt 5: Meta-Eigenschaften verarbeiten.

Darauf folgt die Ermittlung der Instanzen, die Erstellung eines Kno-

Schritt 6: Instanzen abbilden.

tens für jede Instanz und die Verbindung mit der umschliessenden Klasse mittels Kante vom Typ *UNDEFINED_EDGE_TYPE*. In der Mensch-Tier-Ontologie existieren nur die Instanzen *Fred* und *Minka*, die an *Mann* beziehungsweise *Katze* gebunden werden. Die Eigenschaften, die mittels Slot den Instanzen hinzugefügt werden, sind *hatHerrchen* für *Minka* und *istHerrchen* für *Fred*.

4.1.3 *Arbeitsweise des Algorithmus*

Für die fertig aufgebauten Graphen werden zuerst Ähnlichkeiten bei den Knoten gesucht. OntoCompare geht dabei davon aus, daß Knoten ihren Namen nicht ändern, die hierarchische Position im Graphen gleich bleibt, also Blatt-Knoten nicht zu inneren Knoten werden und umgekehrt und daß Nachfolger und Attribute sich nicht wesentlich ändern[11]. Über einen parametrierbaren Schwellwert kann die zugelassene Abweichung von diesen Annahmen beeinflusst werden. Um die Komplexität der Suche nach ähnlichen Knoten drastisch zu verringern, ordnet der Vergleichsalgorithmus die Kind-Knoten jedes Knotens alphabetisch nach ihrem Namen. Für beide Graphen wird danach eine Liste an Blattknoten erstellt, wobei die Graphen von links nach rechts traversiert werden. Aus diesen beiden Listen wird die längste gemeinsame Folge an Knoten genommen und in der Menge der identen Knoten gespeichert. Danach wird mit den inneren Knoten ebenso verfahren. Aus den noch verbleibenden Knoten wird eine Liste gebildet, aus der ebenfalls die längste gemeinsame Knotenfolge ermittelt und in die Menge der identen Knoten eingefügt wird. Bei diesem dritten Durchlauf sind Blattknoten und innere Knoten gemischt. Durch diese Reihenfolge wird dem Umstand Rechnung getragen, daß innere Knoten selten zu Blattknoten werden und umgekehrt. Diese Vorgehensweise bei der Ermittlung ähnlicher Knoten ist heuristisch, falsch oder nicht erkannte Ähnlichkeiten sind daher möglich.

Idente Knoten werden über die längste gemeinsame Folge an Knoten ermittelt.

Da bei der Ähnlichkeitsermittlung mittels längster gemeinsamer Knotenfolge die Bezeichnung eines Knotens ein wesentliches Kriterium war, werden die verbleibenden Knoten auf mögliche Namensänderung untersucht. Dabei werden aus Laufzeitgründen nur Knotenpaare betrachtet, die die gleichen Eltern haben. Um auch umbenannte Eltern zu berücksichtigen, werden alle Knoten gewertet, die im Ursprungsgraph zumindest ein gleiches Elternteil oder ein bereits als umbenannten Knoten identifiziertes Elternteil haben. Die solcherart ermittelten Knoten werden der Menge der möglicherweise umbenannten Knoten zugewiesen. Diese Menge wird entsprechend der hierarchischen Position der Knoten im Graph sortiert: Knoten, die nur mit Blattknoten verbunden sind vor Knoten, die nur mit inneren Knoten verbunden sind vor Knoten, die sowohl mit inneren als auch mit Blattknoten verbunden sind. Die oben angeführten Annahmen bezüglich der Ähnlichkeit von Knoten werden bis auf die Annahme über den Knoten-Namen auf die Menge der möglicherweise umbenannten Knoten angewandt. Wenn der ermittelte Wert für die Ähnlichkeit über einem Schwellwert liegt, wird der Knoten als *vermutlich umbenannt* deklariert im anderen Fall als *vermutlich nicht umbenannt*. Sollte sich für einen Knoten mehrfach das Ergebnis *vermutlich umbenannt* ergeben, so wird die Lösung mit dem höheren Ergebniswert gewertet. Auch die Ermittlung der umbenannten

Umbenannte Knoten müssen die gleichen Eltern haben.

11 Vgl. [6, S. 4]

Knoten ist heuristisch, ungenaue Ergebnisse sind daher möglich, lassen sich aber durch Anpassung der Parameter minimieren.

Mit der Ermittlung der ähnlichen und der umbenannten Knoten ist die Vorkalkulationsphase beendet. Der Vergleichsalgorithmus selbst ist unterteilt in fünf Phasen, wobei jede Phase für die Ermittlung eines Teils der Änderungs-Operationen zuständig ist. Für die Betrachtung der Funktionsweise der einzelnen Phasen führen wir den Begriff des Partner-Knotens ein. Der Partner-Knoten ist ein Knoten im Ursprungsgraph, der während der Prüfung auf Ähnlichkeit oder Umbenennung einem Knoten aus dem Zielgraph zugeordnet wurde.

Änderungen werden in fünf Phasen ermittelt.

Einfügephase: Der Zielgraph wird in topologischer Reihenfolge traversiert. Wird dabei ein Knoten gefunden der weder bei der Ähnlichkeitsprüfung noch bei der Prüfung auf Umbenennung erkannt wurde, so ist dieser Knoten ein neuer Knoten. Da der Zielgraph in topologischer Reihenfolge traversiert wurde kann man sicher sein, daß die Eltern des neuen Knotens einen Partnerknoten im Ursprungsgraph haben, es kann also einfach eine *Insert Node*-Operation für den neuen Knoten angelegt werden.

Knoten-Änderungsphase: Der Zielgraph wird in topologischer Reihenfolge traversiert. Wenn bei einem Knoten Unterschiede bei den Attributen bezüglich seines Partner-Knotens ermittelt werden, wird die Operation *Update Node* an das Edit-Skript angehängt.

Slot-Änderungsphase: Der Zielgraph wird in topologischer Reihenfolge traversiert. Für jeden Slot der im Knoten des Zielgraphs belegt ist und im Partnerknoten nicht belegt ist, wird das Edit-Skript um eine *Insert Slot* Operation ergänzt, wenn sich ein Slot des Partnerknotens im Zielknoten nicht mehr findet, um eine *Remove Slot* Operation.

Kanten-Änderungsphase: Der Zielgraph wird in topologischer Reihenfolge traversiert. Für jeden Knoten im Zielgraph wird geprüft, ob die Kanten zu den Elternknoten gleich sind zu denen des Partnerknotens. Bei Unterschieden werden *Insert Edge*, *Delete Edge* oder *Change Edge Type* Operationen an das Edit-Skript angefügt.

Löschphase: Der Ursprungsgraph wird in Postorder traversiert. Werden dabei Knoten gefunden die im Zielgraph nicht vorhanden sind, so wird das Edit-Skript um eine *Delete Node* Operation ergänzt.

Bis auf die Löschphase werden alle Phasen während einer einzigen, topologischen Traversierung abgearbeitet, lediglich für die Löschphase ist noch eine Postorder Traversierung des Graphen notwendig.

Funktion $vergleicheOntologien(v_{alt}, v_{neu})$

1. Sei v_{alt}die Ursprungsontologie und v_{neu} die Zielontologie
2. Edit Skript $\mathbb{E} = \emptyset$;
3. Menge der ähnlichen Knoten $\mathbb{M} = findeAehnlicheKnoten(v_{alt}, v_{neu})$;
4. Menge der umbenannten Knoten $\mathbb{R} = findeUmbenannteKnoten(v_{alt}, v$
5. $\mathbb{E} = \mathbb{E} \cup \mathbb{R}$;
6. Sei $x$der momentane Knoten bei topologischem traversieren von v_{neu};
   a) IF kein Partnerknoten für x existent
      i. $\mathbb{Z} = \{(p.partner, typ)|(p, x, typ)$ ist eine Kante in $v_{neu}\}$;
      ii. $\mathbb{E} = \mathbb{E} \cup KnotenEinfuegen(x.name, x.attribute, x.slots, \mathbb{Z})$;
      iii. $w = KnotenEinfuegen(x.name, x.attribute, x.slots, \mathbb{Z})$ angewendet auf v_{alt};
      iv. $\mathbb{M} = \mathbb{M} \cup (w, x)$;
   b) ELSE
      i. Sei w der Partnerknoten von x;
      ii. IF $w.attribute \neq x.attribute$ $\mathbb{E} = \mathbb{E} \cup KnotenAendern(w, x.attribute)$;
      iii. IF $gemeinsameSlots(w, x) \neq 1$
         A. Für jeden Slot s_0 in w der nicht in x: $\mathbb{E} = \mathbb{E} \cup SlotLoeschen(w, s_0)$;
         B. Für jeden Slot s_n in x der nicht in w: $\mathbb{E} = \mathbb{E} \cup SlotEinfuegen(w, s_n)$;
      iv. Sei $\mathbb{Y} = \{(y, typ_1)|(u, x, typ_1)$ eine Kante in $v_{neu}\}$;
      v. Sei $\mathbb{U} = \{(u, typ_2)|(u, w, typ_2)$ eine Kante in $v_{alt}\}$;
      vi. Für jedes Paar $(y, typ_1) \in \mathbb{Y}$ mit $u = y.partner$, $(u, typ_2) \in \mathbb{U}$ und $typ_1 \neq typ_2 : \mathbb{E} = \mathbb{E} \cup aendereKantenTyp(u, w, typ_1)$;
      vii. Für jedes Paar $(y, typ_1) \in \mathbb{Y}$ mit $u = y.partner$, $(u, -) \notin \mathbb{U}$: $\mathbb{E} = \mathbb{E} \cup KantenEinfuegen(u, w, typ_1)$;
      viii. Für jedes Paar $(u, typ_2) \in \mathbb{U}$ mit $y = u.partner$, $(y, -) \notin \mathbb{Y}$: $\mathbb{E} = \mathbb{E} \cup KanteLoeschen(u, w)$;
7. Sei w der momentane Knoten bei bottom-up traversieren von v_{alt};
   a) IF kein Partnerknoten für w existent
      i. $\mathbb{E} = \mathbb{E} \cup KnotenLoeschen(w)$;
      ii. Lösche w von v_{alt};
8. RETURN $\mathbb{E}$

Abbildung 4.4: OntoCompare in Pseudocode

Abbildung 4.4[12] zeigt die Arbeitsweise des Algorithmus in Pseudocode. Der Rechenaufwand des Algorithmus steigt sowohl mit der Anzahl der Knoten im Graph als auch mit der Differenz zwischen den beiden Graphen quadratisch an[13].

12 Vgl. Abbildung aus [5, S. 10]
13 Vgl. [5, S. 10]

4.2 PROMPTDIFF

PROMPTDIFF kann als Erweiterungsmodul in den Protégé-Editor[14] eingebunden werden und ermöglicht Vergleiche zwischen Ontologie-Versionen.

4.2.1 *Protégé-Editor*

Der Protégé-Editor ist ein in JAVA erstelltes Programm mit einer grafischen Benutzerschnittstelle und im Bereich freier Software das bedeutendste Werkzeug zu Bearbeitung von Ontologien.

Der Editor ist eines der bedeutendsten Werkzeuge zur Bearbeitung von Ontologien.

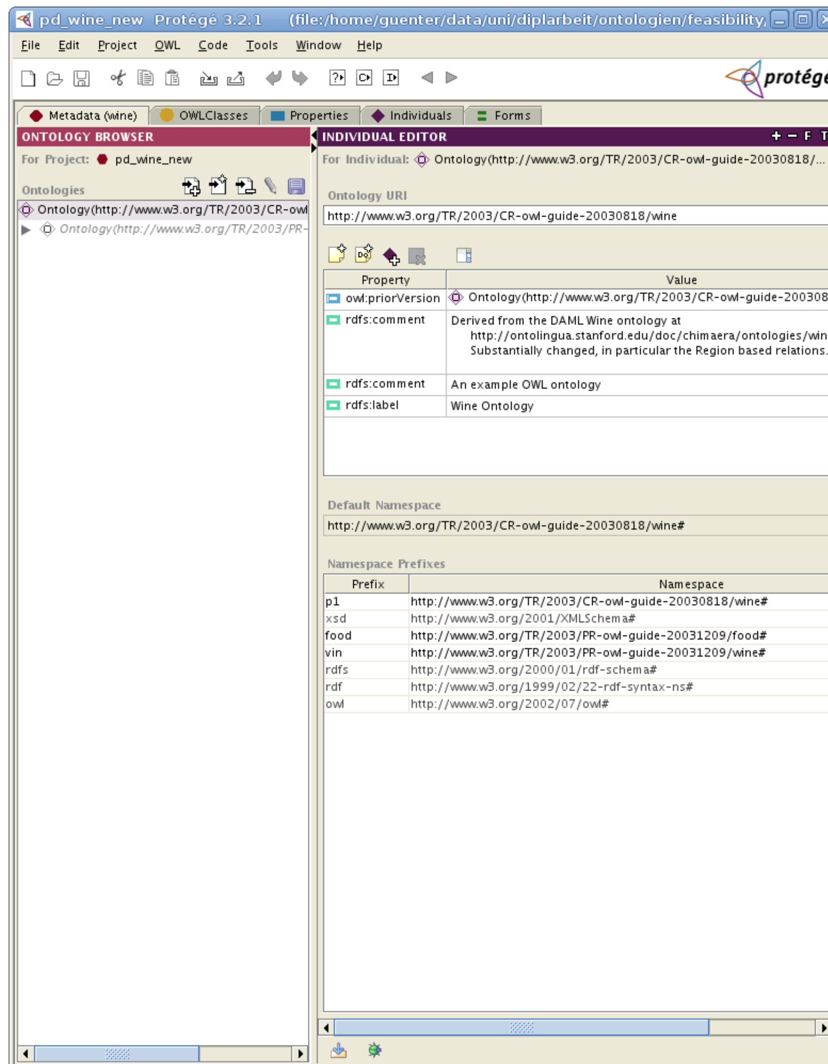


Abbildung 4.5: Die Benutzerschnittstelle des Protégé-Editors

Über einen Modulmechanismus kann der Editor auch von Programmierern ausserhalb des Kern-Entwicklunsteams um Funktionen erweitert werden. Eine Erweiterung ist der PromptTab, über den man PROMPTDIFF nutzen kann.

14 Siehe dazu The Protege Ontology Editor `http://protege.stanford.edu/`

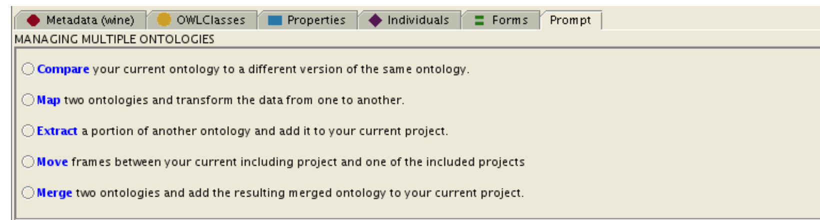


Abbildung 4.6: Der PromptTab

Das Ergebnis eines Vergleichslaufes wird im Editor dargestellt[15] und kann durch den Benutzer weiter analysiert werden. PromptDiff versteht sich als Expertensystem mit dessen Hilfe Ontologien bearbeitet werden können, ein vollautomatisches System zur Ermittlung von Ontologieunterschieden ist es nicht.

4.2.2 *PromptDiff-Modul*

PromptDiff verwendet eine Tripel-Datenstruktur.

Ursprünglich[16] wurde PromptDiff für das Ermitteln der Unterschiede zwischen verschiedenen Versionsständen von Ontologien entwickelt, mittlerweile gibt es auch die Option zum Auffinden von Gemeinsamkeiten von Ontologien aus verschiedenen Quellen[17]. Angelehnt an das *Open Knowledge Base Connectivity* Protokoll werden bei Prompt-Diff Ontologien intern als Klassen, Klassenhierarchien, Instanzen von Klassen, Slots, Slot-Anhänge an Klassen, um Eigenschaften darzustellen, und Facetten um Beschränkungen bei Slot-Werten darzustellen, modelliert[18].

Der PromptDiff-Algorithmus besteht aus zwei Teilen:

- einer erweiterbaren Menge aus heuristischen Regeln
- einem Algorithmus, der die heuristischen Regeln in geeigneter Reihenfolge aufruft und deren Ergebnisse als strukturelle Unterschiede zwischen den Ontologien darstellt

Heuristische Regeln ermitteln die strukturellen Unterschiede.

Jede heuristische Regel vergleicht die beiden Ontologien in Hinblick auf einen speziellen Teil der Struktur und trägt erkannte Änderungen in eine Ergebnistabelle ein. Die Tabelle enthält Angaben über den Frame der Ursprungs-Ontologie, den Frame der Ziel-Ontologie, die Kennzeichnung einer etwaigen Umbenennung, die zugrundeliegende Operation und den Grad der Ähnlichkeit. Zugrundeliegende Operationen können *add*, *delete*, *split*, *merge* und *map* sein, Grad der Ähnlichkeit kann *unchanged*, *isomorphic* oder *changed* sein. Auch das Vergleichsverfahren PromptDiff verwendet Heuristiken, die bei unpassender Parametrierung falsche Ergebnisse liefern können. Die Entwickler wollen das Verfahren als halbautomatisch verstanden wissen und geben mit der Ausgabe des Grades der Ähnlichkeit dem Anwender Hinweise über die Qualität der Erkennung. Bei *unchanged* muß der Benutzer die Zeile nicht weiter beachten, die Definitionen der beide betrachteten Elemente sind unverändert, bei *isomorphic* sind die Frames oder Slots einander zugeordnet, aber nicht identisch, die Ausgabe *changed* bedeutet, daß nicht alle Teile eines Frames oder Slots einander eindeutig zugeordnet werden konnten.

15 Siehe dazu Abschnitt 5.2
16 Siehe dazu [19]
17 Siehe dazu [16]
18 Vgl. [17, S. 2]

Der Algorithmus ruft die heuristischen Regeln in der geeigneten Reihenfolge auf und kombiniert deren Ergebnisse. Dabei darf eine heuristische Regel nur neue Änderungen in die Ergebnistabelle eintragen oder Einträge, die Null-Werte enthalten, durch neue Einträge ersetzen, sie darf aber keine Einträge löschen. Durch dieses Vorgehen ist gewährleistet, daß die Regeln voneinander unabhängig sind und das Regel-Set beliebig erweitert werden kann und auch, daß der Algorithmus gesichert terminiert[19]. Für einen Vergleich werden Frames aus beiden Ontologien herangezogen, wobei ein Frame vom Typ Klasse, Slot, Facette oder Instanz sein kann. Die Liste der heuristischen Regeln ist beliebig erweiterbar, nachfolgend werden die sechs ursprünglichen Regeln vorgestellt.

Frames des selben Typs mit selbem Namen: Wenn zwei Frames vom selben Typ den selben Namen haben, dann werden sie als ident eingestuft.

Geschwister mit den selben Suffixen oder Prefixen: Wenn die Namen von Frames auf der gleichen Hierarchieebene um das gleiche Suffix oder Prefix erweitert oder reduziert werden, so werden sie als ident angesehen.

Einzelnes nicht identes Geschwister: Wenn alle bis auf ein Geschwister ident sind, dann wird auch beim verbleibenden Geschwister eine Identität unterstellt.

Einzelner nicht identer Slot: Wenn alle bis auf einen Slots ident sind, dann wird auch beim verbleibenden Slot Identität unterstellt.

Nicht idente inverse Slots: Bei Wissensmodellen, die inverse Beziehungen erlauben, können auch dadurch Identitäten gefunden werden. Wenn ein Slot als ident eingestuft wurde und es eine inverse Beziehung von diesem Slot gibt, so kann auch der invers refernzierte Slot als ident angesehen werden.

Geteilte Klassen: Diese Regel untersucht, ob es sich bei einer neuen Klasse in einer Hierarchieebene um die Teilung einer zuvor bestehenden Klasse handeln kann.

Simple Regeln werden in geeigneter Reihenfolge aufgerufen.

Jede dieser Regeln ist sehr simpel, die Stärke des Vergleichsverfahrens ergibt sich durch die geeignete Kombination und Abfolge dieser Regeln. So wird jede Regel nur auf Frames angewendet, die noch nicht identifiziert wurden. Die Anzahl der zu überprüfenden Frames nimmt daher rasch ab[20]. Es ist zu erkennen, daß Regeln oft nur ganz bestimmte Typen von Frames untersuchen und daher nur für diese Typen ein Ergebnis liefern können. Die Regel *Einzelner nicht identer Slot* untersucht nur Slots, keine Klassen, sie kann also in der Ergebnistabelle nur Einträge für Slots ändern oder hinzufügen. Dieser Umstand wird vom Algorithmus ausgenutzt, indem eine Abhängigkeitsmatrix der Regeln erstellt und damit der Aufruf der einzelnen Regeln besser gesteuert wird. Liefert eine Regel ein Ergebnis zurück, dann werden alle Regeln, die laut Abhängigkeitsmatrix von diesem Ergebnis beeinflusst werden, nochmals aufgerufen. Liefert eine Regel kein Ergebnis mehr zurück,

19 Vgl. [17, S. 3]
20 Vgl. [17, S. 4]

so wird sie nicht mehr aufgerufen. Tabelle 4.2[21] zeigt, daß nach einer Änderung der Ergebnistabelle durch Regel 4, die Regeln 4 und 5 wieder aufgerufen werden müssen.

Regel	benötigt Info		ändert Info		betroffene
	Klassen	Slots	Klassen	Slots	Regeln
1. Frames des selben Typs mit selbem Namen	-	-	+	+	2, 3, 4, 5
2. Einzelnes nicht identes Geschwister	+	-	+	-	2, 3, 4
3. Geschwister mit den selben Suffixen	+	-	+	-	2, 3, 4
4. Einzelner nicht identer Slot	-	+	-	+	4, 5
5. Nicht idente inverse Slots	-	+	-	+	4, 5
6. Geteilte Klassen	+	-	+	-	2, 3, 4

Tabelle 4.2: Abhängigkeitsmatrix der heuristischen Regeln

Dieses Verfahren hilft die Effizienz gegenüber dem simplen, zyklischen Aufruf aller Regeln, bis keine mehr ein Ergebnis liefert, zu steigern, der Rechenaufwand steigt trotzdem quadratisch mit der Summe der Elemente beider Ontologien an[22].

21 Vgl. [17, S. 5]
22 Vgl. [17, S. 5]

5 DURCHFÜHRUNG DER VERGLEICHE

Die durchgeführten Vergleichsläufe dienen zur Ermittlung des Verhaltens und der Leistungen der beiden Vergleichsverfahren. In Abschnitt 5.1 werden möglichst atomare Änderungen an den Ontologien vorgenommen, um zu zeigen, wie die Vergleichsverfahren diese Änderungen darstellen. Im Rahmen der Genauigkeitsprüfung unter 5.2 wird die Anzahl der richtig erkannten Änderungen ermittelt. Die Leistungsprüfung im Abschnitt 5.3 untersucht die benötigten Laufzeiten der Verfahren zur Ermittlung der Änderungen bei mittelgroßen Ontologien in Abhängigkeit der Änderungsrate.

5.1 ERMITTLUNG DER ERGEBNISDARSTELLUNG

Atomare Änderungen sollen die Darstellung von Unterschieden zwischen Versionen zeigen.

Für diese Tests wurde die Wine-Ontology[1], eine der OWL-Referenzontologien des W3C verwendet und daraus mittels bekannter Änderungen neue Ontologieversionen erzeugt. Die Wein-Ontologie ist eine OWL DL Ontologie kleineren Umfangs, bestehend aus 618 Klassen (inklusive anonymer Klassen und Systemklassen), 249 Instanzen und 67 Eigenschaften. Insgesamt besteht die Datei aus 1249 RDF-Ressourcen die bei einer Umwandlung in das Protégé Frame-Format weniger als 5000 Tripel ergeben. Die durchgeführten Änderungen an der Wein-Ontologie sollten in die Struktur der bestehenden Elemente möglichst wenig eingreifen und eine minimale Änderung darstellen. Oft sind die angeführten Änderungen aus Sicht des dargestellten Wissens unvollständig oder unnötig. Selbstverständlich sollten die Ontologien korrekt aufgebaut sein und den Test im Validator[2] bestehen. Sämtliche erzeugten Ontologieversionen wurden daher im Validator auf Gültigkeit geprüft. Um aber zum Beispiel bei 5.1.1 die Klasse *Juice* sinvoll zu definieren, hätten zumindest Abgrenzungsbedingungen zur Geschwisterklasse *Wine* eingefügt werden müssen. Diese aus Sicht der Wissensdarstellung sinnvolle zusätzlichen Änderungen wurden absichtlich nicht durchgeführt, um das Verhalten der Vergleichsverfahren bei möglichst atomaren Änderungen an einer Ontologie untersuchen zu können.

Ursprungsdatei war immer die Datei wine.rdf aus dem OWL Web Ontology Language Guide vom 10. Februar 2004. Für jedes der zehn Testszenarien wurde ein eigenes Verzeichnis angelegt und die Datei hineinkopiert. Danach wurde die Änderung an den Daten durchgeführt und in eine neue Datei gespeichert. Da Protégé den Inhalt der OWL Dateien verändert, nachdem man sie importiert und dann als Projekt abgespeichert hat, wurden immer Kopien der beiden OWL-Dateien erzeugt, auf die ausschließlich Protégé zugegriffen hat.

Die Ermittlung der Ergebnisdarstellung wurde auf einem Arbeitsplatzrechner, der mit dem Betriebsystem Ubuntu 8.04 und dem Java JDK 1.5.0_08 ausgerüstet war durchgeführt. Die Referenzvergleichsläufe wurden mit dem Protégé OWL Editor in der Version 3.2.1 Build 365 in der Vollinstallation durchgeführt, welcher das Protégé OWL-API

1 Siehe dazu Wine-Ontology `http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf`
2 Siehe dazu OWL Validator `http://www.mygrid.org.uk/OWL/Validator`

in der Version 3.0.0 enthält. Die Erweiterung von OntoCompare zum Einlesen von OWL greift auf genau dasselbe OWL-API zu.

Beim Start des Vergleichslaufes mit OntoCompare wurden dem Programm die URIs für die Ursprungs- und die Ziel-Ontologie übergeben. Die Ausgaben auf der Konsole wurden in eine Ergebnisdatei umgelenkt. Es waren keine Vor- oder Nachbereitungsarbeiten erforderlich.

PromptDiff erfordert die strikte Einhaltung eines Ablaufes.

Der Protégé OWL Editor legt ein Projekt für eingelesene Ontologien an und verändert den Inhalt der importierten Datei. Folgende Vorgehensweise muß daher bei einem Vergleichslauf eingehalten werden, um reproduzierbare Ergebnisse zu erhalten:

1. Protégé OWL Editor öffnen
2. OWL-Datei der Ursprungs-Ontologie laden
3. Ursprungs-Projekt speichern
4. Protégé OWL Editor schliessen
5. Protégé OWL Editor öffnen
6. OWL-Datei der Ziel-Ontologie laden
7. Ziel-Projekt speichern
8. Protégé OWL Editor schliessen
9. Protégé OWL Editor öffnen
10. Ziel-Projekt laden
11. Promt-Tab aktivieren
12. In den Prompt-Tab wechseln
13. Ursprungs-Projekt auswählen
14. Vergleichslauf starten

Die Ergebnistabelle wird im Benutzer-Interface angezeigt, läßt sich aber nicht abspeichern, ebenso die Statistikwerte. Es wurden daher die Ergebnisse zusammengefaßt in einer Textdatei dokumentiert.

5.1.1 *Hinzufügen einer Klasse*

Es wurde eine Klasse *Juice* als Unterklasse von *PotableLiquid* mittels

```
<owl:Class rdf:ID="Juice">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
</owl:Class>
```

in die Wein-Ontologie eingefügt. Die Klasse *Juice* hat keine Kindklassen, eine einzige Elternklasse *PotableLiquid* und eine Geschwisterklasse *Wine*.

OntoCompare erkennt die neu hinzugekommene Klasse und gibt ein Edit-Script mit der einzigen Operation *Insert Node* aus.

```
Edit script:
InsertNode Juice
  Parents: food:PotableLiquid
```

Eine Operation vom Typ *Insert Node* bedeutet dabei das Einfügen eines Knotens in den Graph und die Erstellung von Kanten des Typs *IS_A* zu den Knoten der Elternklassen.

PromptDiff erkennt die neu hinzugekommene Klasse und gibt in seiner Ergebnistabelle die Änderung *p1:Juice Add* an. Der Indikator für eine Umbenennung hat den Wert *No*, zum Grad der Ähnlichkeit wird keine Angabe gemacht. Weiters enthält die Ergebnistabelle Einträge für alle Instanzen, die mittels *owl:AllDifferent* definiert wurden. Es werden alle Instanzdefinitionen (fünf Stück) einmal dem Namensraum *vin* zugehörig und einmal dem Namensraum *p1* zugehörig neu hinzugefügt und danach wieder als gelöscht angeführt, der Grad der Ähnlichkeit wird dabei nicht ausgegeben. In der Statistik ist lediglich eine Ergänzung angeführt.

Die Definition von Instanzen bereitet PromptDiff Probleme.

Die Vermutung, daß es sich bei der Behandlung der Instanzen um ein Problem bei der Umwandlung einer OWL-Datei in das Protégé-Format handelt, wird durch einen Versuch, bei dem Ursprungs- und Ziel-Ontologie ident sind, unterstrichen. Auch bei diesem Vergleich werden die Instanzen irrtümlicher Weise als hinzugefügt und gelöscht ausgewiesen.

5.1.2 *Einfügen einer Klasse*

Es wurde die Klasse *Between* als Unterklasse von *PotableLiquid* mittels

```
<owl:Class rdf:ID="Between">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
</owl:Class>
```

in die Wein-Ontologie eingefügt und die Referenz auf die Elternklasse der Klasse *Wine*, Zeile 32, auf die Klasse *Between* gelegt.

```
<rdfs:subClassOf rdf:resource="#Between" />
```

Die Klasse *Between* wurde also in der Klassenhierarchie zwischen *PotableLiquid* und *Wine* eingefügt. Die Klasse *PotableLiquid* hat nur die Klasse *Between* als Kindklasse, die Klasse *Wine* hat nur die Klasse *Between* als Elternklasse.

OntoCompare gibt folgendes Edit-Script aus:

```
Edit script:
InsertNode Between
  Parents: food:PotableLiquid
InsertEdge from node Between to node Wine of type IS_A
RemoveEdge from node food:PotableLiquid to node Wine
```

Die Operation *Insert Node* erzeugt den Knoten *Between* und die Kante *IS_A* zum Knoten *PotableLiquid*. Das Umhängen des Knotens *Wine* wird über das Einfügen einer Kante zum Knoten *Between* und dem Löschen der Kante zu *PotableLiquid* erreicht.

PromptDiff gibt in der Ergebnistabelle die Klasse *Between* als eingefügt an, der Grad der Ähnlichkeit wird nicht angegeben. Weiters wird die Klasse *Wine* als geändert angeführt, als Grad der Ähnlichkeit wird *changed* angegeben. Das Verfahren wertet

das Einfügen einer Klasse als Einfügeoperation und als Änderungsoperation bei der Kindklasse.
Für weitere 34 Klassen wird eine Änderung unterstellt, wobei als Grad der Ähnlichkeit *isomorphic* angegeben wird. Als zusätzliche Erklärung wird *frame name and type are the same* ausgegeben. Auch bei den Eigenschaften *hasColor*, *hasWineDescriptor* und *madeFromGrape* wird eine Änderung mit einem Ähnlichkeitsgrad von *isomorphic* angegeben. Gleich wie bei 5.1.1 werden alle Instanzdefinitionen mittels *owl:AllDifferent* als gelöscht und neu hinzugefügt angeführt. Die Stelle an der die Klasse *Between* eingefügt wurde, war bewußt so gewählt, daß alle anderen Elemente der Ontologie außer der Klasse *Wine* nicht beeinflußt werden. Die eingetragenen Änderungen erscheinen daher willkürlich, sie haben mit den tatsächlichen Änderungen nichts zu tun. Der angeführte Grad der Ähnlichkeit, *isomorphic*, zeigt an, daß es sich bei den Einträgen lediglich um prüfenswerte Änderungen handelt. Die Ähnlichkeiten der beiden Elemente sind so groß, daß das Verfahren dieses Element nicht als zum Ursprungselement verschiedenes Element wertet. In der Statistik werden nur die eindeutig erkannten Änderungen angeführt, eine Ergänzung und eine direkte Änderung.

5.1.3 *Löschen einer Klasse*

Für diesen Vergleich wurde die in 5.1.2 hinzugefügte Klasse *Between* wieder gelöscht, also Referenzontologie und Vergleichsontologie miteinander vertauscht.

OntoCompare erzeugt eine Kante von *PotableLiquid* zu *Wine* vom Typ *IS_A*, entfernt die Kante vom Knoten *Between* zu *Wine* und löscht den Knoten *Between*.

```
Edit script:
InsertEdge from node food:PotableLiquid to node Wine
  of type IS_A
RemoveEdge from node Between to node Wine
DeleteNode Between
```

Beim Löschen eines Knotens werden zuvor alle Kanten zu Elternknoten gelöscht.

PromptDiff gibt in der Ergebnistabelle das Löschen der Klasse Between an. Die Änderungen an der Klasse Wine werden wie bei 5.1.2 ausgewiesen. Ebenfalls wie bei 5.1.2 werden für weitere 34 Klassen und drei Eigenschaften Änderungen vom Grad *isomorphic* angezeigt. Die Behandlung der Instanzen ist ident wie bei den vorigen Vergleichen. Die Statistik weist eine Löschung und eine direkte Änderung aus.

5.1.4 *Umbenennen einer Klasse*

Als Referenz-Ontologie wurde die in 5.1.2 um die Klasse `Between` erweiterte Wein-Ontologie verwendet. In der Vergleichs-Ontologie wurde *Between* in *Between_it* umbenannt. Die Refernz der Klasse *Wine* wurde ebenfalls auf *Between_it* aktualisiert.

```
<owl:Class rdf:ID="Between_it">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
</owl:Class>

<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="#Between_it" />
...
```

OntoCompare weist im Edit-Script eine Operation *Rename Node* aus.

```
Edit script:
RenameNode 898-Between_it(1)
```

PromptDiff zeigt in der Ergebnistabelle die Umbenennung der Klasse *Between* in *Between_it* an. Der Ähnlichkeitsgrad wird mit *changed* angegeben, der Indikator für die Umbenennung hat den Wert *Yes*. Zusätzlich ist die Bemerkung *Same superclass and subclass* angeführt. Für die Klasse *Wine* wird eine Änderung vom Umfang *isomorphic* mit der Bemerkung *frame name and type are the same* ausgegeben. Die Änderung des Klassennamens wird also erkannt, die Änderung der hierarchischen Beziehung zu einer Klasse mit neuem Namen trotzdem als mögliche Änderung ausgewiesen. Die Statistik führt lediglich die Namensänderung an. Die Instanzen werden genau wie bei den vorhergehenden Vergleichsläufen behandelt.

5.1.5 *Einfügen einer Eigenschaft*

Der Wein-Ontologie wurde eine funktionale Objekteigenschaft *hasSeller* hinzugefügt. Die Eigenschaft ist der Klasse Wine zugeordnet, steht aber sonst in keinem Hierarchieverhältnis zu anderen Eigenschaften und wird von keiner Instanz verwendet, die Verflechtung mit bereits bestehenden Elementen ist daher gering. Als Referenz-Ontologie dient wieder die unveränderte Wein-Ontologie.

```
<owl:ObjectProperty rdf:ID="hasSeller">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>
```

OntoCompare fügt die Eigenschaft *hasSeller* als Kind-Knoten des virtuellen Wurzelknotens in den Graph, ebenso die Kardinalitätsbeschränkung. Die Zuordnung der Eigenschaft zur Klasse *Wine* erfolgt über eine Kante des Typs *IS_A* vom Knoten der Kardinalitätsberschränkung zum Knoten *Wine*.

```
Edit script:
InsertNode hasSeller
  Parents: virtual_root
InsertNode hasSeller exactly 1
  Parents: virtual_root
InsertEdge from node hasSeller exactly 1 to node Wine
  of type IS_A
```

PromptDiff gibt die Eigenschaft *hasSeller* als hinzugefügt an. Die Klasse *Wine* wird als geändert gelistet, der Ähnlichkeitsgrad ist dabei *changed*, als zusätzliche Erklärung wird *frame name and type are the same* angeführt. Für die gleichen 34 Klassen wie bei 5.1.2 wird eine Änderung vom Grad *isomorphic*

angegeben, ebenso werden die gleichen Änderungen bei den Eigenschaften und Instanzen ausgewiesen.

5.1.6 *Löschen einer Eigenschaft*

Die in 5.1.5 eingeführte Eigenschaft wurde wieder gelöscht, Ziel-Ontologie war also die unveränderte Wein-Ontologie.

OntoCompare löscht die Kante von Knoten *hasSeller* zu Knoten *Wine* und danach die Knoten *hasSeller exactly 1* und *hasSeller*.

```
Edit script:
RemoveEdge from node hasSeller exactly 1 to node Wine
DeleteNode hasSeller exactly 1
DeleteNode hasSeller
```

PromptDiff führt in der Ergebnistabelle die Löschung der Eigenschaft hasSeller an, ebenso die Änderung der Klasse Wine. Alle anderen angeführten Änderungen sind exakt gleich den Einträgen der Ergebnistabelle aus dem Vergleich 5.1.2.

5.1.7 *Umbenennen einer Eigenschaft*

Als Ursprungs-Ontologie wurde für diesen Vergleich die in 5.1.5 erstellte, um die Eigenschaft hasSeller erweiterte Wein-Ontologie verwendet. In der Ziel-Ontologie wurde die Eigenschaft hasSeller auf hasRetailer umbenannt. Die Referenz auf die Eigenschaft in der Klasse *Wine* wurden entsprechend angepasst.

```
<owl:ObjectProperty rdf:ID="hasRetailer">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>

<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasRetailer" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
...
```

OntoCompare ändert den Namen des Knotens *hasSeller exactly 1* auf *hasRetailer exactly 1*, ändert danach den Namen des Knotens *hasSeller* und fügt den Slot mit Inhalt *hasRetailer* dem Knoten *hasRetailer exactly 1* hinzu. Zuletzt wird der Slot *hasSeller* am Knoten *hasRetailer exactly 1* gelöscht.

PromptDiff weist die Eigenschaft *hasSeller* auf *hasRetailer* umbenannt aus, Ähnlichkeitsgrad ist *changed*. Für die Klasse *Wine* wird auch eine Änderung ausgewiesen, ebenfalls mit Ähnlichkeitsgrad *changed*. Wieder werden Änderungen an den gleichen 34 Klassen und drei Eigenschaften mit Ähnlichkeitsgrad *isomorphic* in der Tabelle vermerkt. Auch die Einfüge- und Lösch-Operationen der Instanzen sind wieder aufgeführt. Obwohl zwei Änderungen mit Ähnlichkeitsgrad *changed* in der Ergebnistabelle aufscheinen wird in der Statistik nur eine Änderung gezählt.

5.1.8 *Einfügen einer Instanz*

Für diesen Vergleich wurde die in 5.1.2 erzeugte Ontologie als Ursprungs-Ontologie verwendet. Der Klasse *Between* wurde eine Instanz *Most* hinzugefügt um die Ziel-Ontologie zu erzeugen.

```
<Between rdf:ID="Most" />
```

OntoCompare erkennt das Einfügen der Instanz Most als Kind-Knoten des Knotens Between.

```
Edit script: InsertNode Most
  Parents: Between
```

PromptDiff erkennt die hinzugefügte Instanz *Most*. Weiters enthält die Ergebnistabelle wieder die Angaben über die hinzugefügten und wieder gelöschten Instanzen die mittels *owl:AllDifferent* definiert sind.

5.1.9 *Löschen einer Instanz*

Die Instanz *Most* wurde wieder aus der Ontologie gelöscht. Es wurde dazu die Ursprungs- und die Ziel-Ontologie aus 5.1.8 miteinander vertauscht.

OntoCompare erkennt das Löschen der Instanz *Most* und löscht den gleichnamigen Knoten aus dem Graph.

```
Edit script:
DeleteNode Most
```

PromptDiff erkennt die gelöschte Instanz *Most*. Weiters enthält die Ergebnistabelle wieder die Angaben über die hinzugefügten und wieder gelöschten Instanzen, die mittels *owl:AllDifferent* definiert sind.

5.1.10 *Umbenennen einer Instanz*

Die Instanz *Most* wurde umbenannt in *Mostler*.

```
<Between rdf:ID="Mostler" />
```

OntoCompare erkennt die Umbenennung der Instanz und weist eine *Rename Node* Operation aus.

```
Edit script:
RenameNode 2425-Mostler(3)
```

PromptDiff erkennt die Namensänderung der Instanz *Most*. Weiters enthält die Ergebnistabelle wieder die Angaben über die hinzugefügten und wieder gelöschten Instanzen die mittels *owl:AllDifferent* definiert sind.

5.1.11 *Bewertung der Ergebnisdarstellung*

Sowohl OntoCompare als auch PromptDiff verwenden das Protégé-OWL API für das Parsen der OWL-Dateien. Die Unterschiede liegen in der weiteren Verarbeitung der erzeugten Objekte. Während OntoCompare die RDF-Objekte für seine weiteren Betrachtungen verwendet, wird bei PromptDiff in das interne Protégé-Format, eine Triple-Repräsentation, umgewandelt. Die Vergleichsverfahren operieren auf ihrer internen Darstellung und weisen als Ergebnis auch die Veränderungen ihrer internen Datenstrukturen aus. Durch diesen Umstand und durch die unterschiedliche Benutzerschnittstelle sind die Ergebnisse nicht direkt vergleichbar, eine Änderung in einer OWL-Datei wird auf unterschiedliche Weise angezeigt. Während PromptDiff in seiner grafischen Benutzerschnittstelle dem Benutzer die Möglichkeit gibt, ermittelte Änderungen zu analysieren und zu bewerten, liefert OntoCompare ein Edit-Skript in dem alle gefundenen Änderungen aufgeführt sind, ohne weitere Analysemöglichkeit.

Unterschiedliche interne Darstellungen liefern leicht unterschiedliche Ergebnisse.

Die ausgewiesenen Änderungen sind manchmal unterschiedlich, obwohl sie in Summe zum selben Ergebnis führen. So wird in 5.1.2 die Änderung einer Elternklasse von OntoCompare als zwei Operationen gewertet (*Insert Edge* und *Remove Edge*), von PromptDiff jedoch nur als einzelne Operation (Änderung bei der Kindklasse). Betrachtet man die tatsächlich durchgeführten Änderungen in der OWL-Struktur, dann handelt es sich um eine Änderung der Referenz auf eine Elternklasse, berücksichtigt man allerdings die interne Darstellung der Ontologie in OntoCompare als Graph, dann muß zuerst eine neue Kante eingefügt und danach die vorhandene gelöscht werden, um den Graph verbunden zu halten.

Mitunter impliziert eine ausgewiesene Änderungsoperation auch Veränderungen an mehreren Stellen in den OWL-Daten. Bei 5.1.4 wird von beiden Verfahren eine Operation zur Änderung des Namens angeführt, in der OWL-Datei müssen zusätzlich auch alle Referenzen auf das geänderte Element angepasst werden. Deutlich zeigen sich die durch die interne Darstellung verursachten unterschiedlichen Ergebnisse auch bei der unter 5.1.5 durchgeführten Änderung. Da OntoCompare jede Eigenschaft getrennt als Knoten in den Graph einfügt, ergibt sich eine Operation mehr als bei PromptDiff und eine Einfügeoperation statt einer Änderungsoperation.

Die Qualität der Änderungserkennung ist sehr gut. Beide Verfahren erkannten alle durchgeführten Änderungen eindeutig. Einzig mit den Instanzdefinitionen via *owl:AllDifferent*-Konstrukt hat PromptDiff Schwierigkeiten und meldet diese immer fälschlicher Weise als hinzugefügt und gelöscht. Nach Einfüge- oder Löschaktionen, bei denen sich die Klassenhierarchie verändert hat, wurden auch anderen Klassen Änderungen unterstellt. Zu beachten ist dabei jedoch der Ansatz von PromptDiff als interaktives System[3]. Es wurden Änderungen an der internen Repräsentation der Daten erkannt, der Benutzer wird darauf hingewiesen, die Änderungen sind aber so gering, daß als Grad der Ähnlichkeit *isomorphic* angegeben wird und sie aufgrund der zusätzlich ausgegebenen Beschreibung des Eintrages verworfen werden können. In der statistischen Zusammenfassung werden daher Änderungen vom Grad *isomorphic* durch das Verfahren auch nicht als Änderungen gewertet.

3 Siehe dazu [18]

5.2 GENAUIGKEITSPRÜFUNG

Die Vergleichsläufe im Rahmen der Genauigkeitsprüfung sollen zeigen, wie gut Änderungen zwischen Ontologieversionen von den einzelnen Verfahren erkannt und wie sie klassifiziert werden. Wie unter 5.1 wurden veränderte Versionen der Wine-Ontologie als Testontologien verwendet. Auch die Anordnung der Versuchsumgebung und die Durchführung der Versuche war genau gleich wie unter 5.1 beschrieben. Aufgrund der Erfahrungen die bei der Ermittlung der Ergebnisdarstellung gewonnen wurden, wird das Problem der hinzugefügten und wieder gelöschten Instanzen bei Vergleichsläufen von PromptDiff bei den nachfolgenden Tests nicht mehr behandelt.

Unterschiede sollen möglichst gut erkannt und klassifiziert werden.

5.2.1 *Hinzufügen von Elementen*

Die erste Testanordnung behandelt den einfachsten und wohl auch am häufigsten vorkommenden Fall der Änderung zwischen zwei Ontologieversionen, das Hinzukommen von Elementen. Um das Einfügen von Elementen nachzustellen, wurden aus der Wine-Ontologie Elemente gelöscht und diese Version als Ursprungsontologie verwendet.

Aus der vollständigen Ontologie wurde die Definition der Klasse Muscadet, Zeile 870 bis Zeile 908, auskommentiert. Dieser Teil beinhaltet die Definition der Klasse, eine Kardinalitätsbeschränkung, vier weitere Beschränkungen und einer Zuordnung zu einer Collection.

```
<owl:Class rdf:ID="Muscadet">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasBody" />
      <owl:hasValue rdf:resource="#Light" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasFlavor" />
      <owl:hasValue rdf:resource="#Delicate" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasSugar" />
      <owl:hasValue rdf:resource="#Dry" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape" />
      <owl:hasValue rdf:resource="#PinotBlancGrape" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Loire" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#locatedIn" />
```

```
    <owl:hasValue rdf:resource="#MuscadetRegion" />
  </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

Weiters wurde die Definition der Instanz *MuscadetRegion*, die Zeilen 2120 bis 2122,

```
<Region rdf:ID="MuscadetRegion">
 <locatedIn rdf:resource="#LoireRegion" />
</Region>
```

als auch die Instanz SevereEtMaineMuscadet, Zeilen 2293 bis 2296, auskommentiert.

```
<Muscadet rdf:ID="SevreEtMaineMuscadet">
 <hasMaker  rdf:resource="#SevreEtMaine" />
</Muscadet>
```

Die Definition der Objekt-Eigenschaft *adjacentRegion* in den Zeilen 213 bis 217,

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

sowie der Eintrag des Verweises auf diese Eigenschaft in der Zeile 2067 wurden ebenfalls auskommentiert.

```
<adjacentRegion rdf:resource="#SonomaRegion" />
```

OntoCompare weist elf *InsertNode* Operationen und eine *InsertSlot* Operation aus.

```
Edit script:
InsertNode adjacentRegion
  Parents: virtual_root
InsertNode madeFromGrape max 1
  Parents: virtual_root
InsertNode locatedIn has MuscadetRegion
  Parents: virtual_root
InsertNode madeFromGrape has PinotBlancGrape
  Parents: virtual_root
InsertNode hasSugar has Dry
  Parents: virtual_root
InsertNode hasFlavor has Delicate
  Parents: virtual_root
InsertNode hasBody has Light
  Parents: virtual_root
InsertNode Muscadet
  Parents: owl:Thing, hasBody has Light,
    madeFromGrape has PinotBlancGrape,
    madeFromGrape max 1,
    hasFlavor has Delicate,
    hasSugar has Dry
InsertNode SevreEtMaineMuscadet
  Parents: Muscadet
InsertSlot SonomaRegion at node MendocinoRegion
InsertNode MuscadetRegion
  Parents: Region
InsertNode Loire and (locatedIn has MuscadetRegion)
  Parents: Muscadet
```

Die erste angeführte Operation fügt einen Knoten für die Objekt-Eigenschaft *adjacentRegion* als Kind-Knoten des virtuellen Wurzelknotens ein. Danach werden die fünf Beschränkungen der Klasse *Muscadet* eingefügt, in weiterer Folge

die Klasse *Muscadet* selber. Die nächste Operation fügt die Instanz *SevereEtMaineMuscadet* ein. Die einzige InsertSlot-Operation betrifft den Verweis auf die Eigenschaft *adjacentRegion*. Das Einfügen der Instanz *MuscadetRegion* und deren Beschränkung *locatedIn* wird über zwei weitere *InsertNode*-Operationen abgebildet.

PromptDiff führt eine Operation für das Hinzufügen der Klasse *Muscadet*, eine Operation für das Hinzufügen der Eigenschaft *adjacentRegion*, eine Operation für das Hinzufügen der Instanz *MuscadetRegion*, sowie eine Operation für das Hinzufügen der Instanz *SevreEtMaineMuscadet* an. Ausserdem wird eine Änderung an der Instanz *MendocinoRegion* ausgegeben.

Metadata (wine) | OWLClasses | Properties | Individuals | Forms | Prompt

Table view | Tree view

Image table

f1	f2	renamed	operation	map level	rename explanation
	p1:Muscadet	No	Add		<null>
	p1:adjacentRegion	No	Add		<null>
	AllDifferent {vin:Sweet, vi...	No	Add		<null>
	AllDifferent {vin:Light, vir...	No	Add		<null>
	AllDifferent {vin:Delicate,...	No	Add		<null>
	AllDifferent {p1:Delicate, ...	No	Add		<null>
	AllDifferent {p1:Sweet, p1...	No	Add		<null>
	AllDifferent {vin:Red, vin:...	No	Add		<null>
	AllDifferent {p1:Light, p1...	No	Add		<null>
	AllDifferent {p1:Red, p1:V...	No	Add		<null>
	AllDifferent {vin:Bancroft...	No	Add		<null>
	AllDifferent {p1:Bancroft,...	No	Add		<null>
	p1:MuscadetRegion	No	Add		<null>
	p1:SevreEtMaineMuscadet	No	Add		<null>
AllDifferent {vin:Bancroft,...		No	Delete		<null>
AllDifferent {p1:Sweet, p1...		No	Delete		<null>
AllDifferent {p1:Bancroft, ...		No	Delete		<null>
AllDifferent {p1:Red, p1:W...		No	Delete		<null>
AllDifferent {vin:Sweet, vi...		No	Delete		<null>
AllDifferent {vin:Red, vin:V...		No	Delete		<null>
AllDifferent {p1:Delicate, ...		No	Delete		<null>
AllDifferent {vin:Delicate, ...		No	Delete		<null>
AllDifferent {p1:Light, p1:...		No	Delete		<null>
AllDifferent {vin:Light, vir...		No	Delete		<null>
p1:MendocinoRegion	p1:MendocinoRegion	No	Map	Directly-changed	frame name and type are the same

Abbildung 5.1: PromptDiff Ergebnis 'Elemente hinzufügen'

5.2.2 *Elemente hinzufügen und löschen*

Zusätzlich zu den unter 5.2.1 durchgeführten Änderungen in der Ursprungsontologie, wurden für diesen Vergleichslauf alle Elemente die deutsche Weine definieren aus der Zielontologie auskommentiert. Auf diese Weise wurden aus Sicht des Vergleichsverfahrens Elemente hinzugefügt und gelöscht.

Die gelöschten Elemente waren die Definition der Klasse *GermanWine* in den Zeilen 1149 bis 1157,

```
<owl:Class rdf:ID="GermanWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#locatedIn" />
      <owl:hasValue rdf:resource="#GermanyRegion" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

die Definition der Instanz *GermanyRegion* in der Zeile 1981,

```
<Region rdf:ID="GermanyRegion" />
```

die Definitionen der deutschen Weingüter und deren Weine in den Zeilen 2241 bis 2259,

```
<Winery rdf:ID="SchlossRothermel" />

<SweetRiesling
    rdf:ID="SchlossRothermelTrochenbierenausleseRiesling">
  <locatedIn rdf:resource="#GermanyRegion" />
  <hasMaker  rdf:resource="#SchlossRothermel" />
  <hasSugar  rdf:resource="#Sweet" />
  <hasFlavor rdf:resource="#Strong" />
  <hasBody   rdf:resource="#Full" />
</SweetRiesling>

<Winery rdf:ID="SchlossVolrad" />

<SweetRiesling
    rdf:ID="SchlossVolradTrochenbierenausleseRiesling">
  <locatedIn rdf:resource ="#GermanyRegion" />
  <hasMaker  rdf:resource="#SchlossVolrad" />
  <hasSugar  rdf:resource="#Sweet" />
  <hasFlavor rdf:resource="#Moderate" />
  <hasBody   rdf:resource="#Full" />
</SweetRiesling>
```

sowie die Einträge in der Collection in den Zeilen 2389 und 2390.

```
<vin:Winery rdf:about="#SchlossRothermel" />
<vin:Winery rdf:about="#SchlossVolrad" />
```

OntoCompare weist in dem durch den Vergleichslauf generierten Edit-Skript 36 Operationen aus. Zur einfacheren Referenzierung auf die Operationen wurden in der Darstellung des Skript-Inhalts die einzelnen Operationen nummeriert.

```
Edit script:
 1 RenameNode 1736−SevreEtMaineMuscadet(3)
 2 RenameNode 273−locatedIn has MuscadetRegion(1)
 3 RenameNode 2057−MuscadetRegion(3)
 4 InsertNode adjacentRegion
     Parents: virtual_root
 5 InsertNode madeFromGrape max 1
     Parents: virtual_root
 6 RemoveSlot GermanyRegion at node locatedIn has
     MuscadetRegion
 7 InsertSlot MuscadetRegion at node locatedIn has
     MuscadetRegion
 8 InsertNode madeFromGrape has PinotBlancGrape
     Parents: virtual_root
 9 InsertNode hasSugar has Dry
     Parents: virtual_root
10 InsertNode hasFlavor has Delicate
     Parents: virtual_root
11 InsertNode hasBody has Light
     Parents: virtual_root
12 InsertNode hasBody has Full
     Parents: virtual_root
13 InsertNode hasFlavor only {Moderate Strong}
     Parents: virtual_root
14 InsertNode Muscadet
     Parents: owl:Thing,
       madeFromGrape has PinotBlancGrape,
       hasBody has Light,
       hasFlavor has Delicate,
       hasSugar has Dry,
```

```
      madeFromGrape max 1
15 RemoveSlot Full at node SevreEtMaineMuscadet
16 RemoveSlot Moderate at node SevreEtMaineMuscadet
17 RemoveSlot SchlossVolrad at node
      SevreEtMaineMuscadet
18 InsertSlot SevreEtMaine at node
     SevreEtMaineMuscadet
19 RemoveSlot Sweet at node SevreEtMaineMuscadet
20 RemoveSlot GermanyRegion at node
    SevreEtMaineMuscadet
21 InsertEdge from node Muscadet to node
    SevreEtMaineMuscadet of type IS_A
22 RemoveEdge from node SweetRiesling to node
     SevreEtMaineMuscadet
23 InsertSlot SonomaRegion at node MendocinoRegion
24 InsertSlot LoireRegion at node MuscadetRegion
25 InsertNode Loire and (locatedIn has MuscadetRegion)
    Parents: Muscadet
26 InsertNode SweetRiesling
    Parents: hasFlavor only {Moderate Strong},
      DessertWine,
      hasBody has Full
27 InsertEdge from node SweetRiesling to node Riesling
     and (hasSugar has Sweet) of type IS_A
28 RemoveEdge from node SweetRiesling to node Riesling
     and (hasSugar has Sweet)
29 DeleteNode Wine and (locatedIn has GermanyRegion)
30 DeleteNode GermanWine
31 DeleteNode
     SchlossRothermelTrochenbierenausleseRiesling
32 DeleteNode SweetRiesling
33 DeleteNode SchlossVolrad
34 DeleteNode SchlossRothermel
35 DeleteNode hasFlavor only {Moderate Strong}
36 DeleteNode hasBody has Full
```

Die Operationen 4, 5, 8, 9, 10, 11, 14, 23 und 25 sind gleich den Operationen, die unter 5.2.1 ermittelt wurden. Das Löschen der Elemente betreffend deutsche Weine wird über die Operationen 6, 15, 16, 17, 19, 20, 26 bis 36 abgebildet. Bemerkenswert daran ist, daß sich die Operationen auf Elemente die mit *Muscadet* zu tun haben beziehen. Wenn man sich die Operationen 1 bis 3 ansieht, stellt man fest, daß es sich um Umbenennungen handelt, obwohl keine Umbenennungen stattgefunden haben. OntoCompare wertet offensichtlich das Löschen der Instanz *GermanyRegion* und das Hinzufügen der *MuscadetRegion* als Umbenennung. Deutlich wird dieses Verhalten auch bei der Instanz *SevreEtMaineMuscadet* und den Operationen 21 und 22. In Operation 21 wird eine Kante zwischen dem umbenannten Knoten *SevreEtMaine-Muscadet* und dem Knoten *Muscadet* erzeugt, in Operation 22 wird die Kante die ursprünglich zwischen *SweetRiesling* und *GermanWine* bestanden hat gelöscht. Fehlerhaft sind auch die Operationen 26, 27, 28 und 32, mittels derer ein Knoten *SweetRiesling* hinzugefügt, eine Kante zum Knoten *Riesling* angelegt und danach beides wieder rückgängig gemacht wird.

PromptDiff führt die Klasse *Muscadet*, die Instanzen *MuscadetRegion*, *SevreEtMaineMuscadet* und die Eigenschaft *adjacentRegion* als hinzugefügt an. Die Klasse *GermanWine*, sowie die Instanzen *SchlossVolrad*, *SchlossVolradTrochenbierenausleseRiesling*,

Metadata (wine) | OWLClasses | Properties | Individuals | Forms | Prompt

Table view | Tree view

Image table

f1	f2	renamed	operation	map level	rename explanation
	p1:Muscadet	No	Add		<null>
	p1:adjacentRegion	No	Add		<null>
	AllDifferent {vin:Red, vin:W...	No	Add		<null>
	AllDifferent {vin:Sweet, vir...	No	Add		<null>
	AllDifferent {vin:Light, vin...	No	Add		<null>
	AllDifferent {p1:Light, p1:...	No	Add		<null>
	AllDifferent {p1:Red, p1:W...	No	Add		<null>
	AllDifferent {p1:Delicate, p...	No	Add		<null>
	AllDifferent {vin:Bancroft, ...	No	Add		<null>
	AllDifferent {p1:Sweet, p1...	No	Add		<null>
	AllDifferent {p1:Bancroft, ...	No	Add		<null>
	AllDifferent {vin:Delicate, ...	No	Add		<null>
	p1:MuscadetRegion	No	Add		<null>
	p1:SevreEtMaineMuscadet	No	Add		<null>
AllDifferent {p1:Bancrof...		No	Delete		<null>
p1:SchlossVolrad		No	Delete		<null>
AllDifferent {vin:Bancrof...		No	Delete		<null>
p1:SchlossVolradTroch...		No	Delete		<null>
p1:GermanWine		No	Delete		<null>
p1:GermanyRegion		No	Delete		<null>
AllDifferent {p1:Delicate...		No	Delete		<null>
AllDifferent {vin:Sweet, v...		No	Delete		<null>
p1:SchlossRothermelTr...		No	Delete		<null>
AllDifferent {vin:Delicate...		No	Delete		<null>
AllDifferent {p1:Red, p1...		No	Delete		<null>
p1:SchlossRothermel		No	Delete		<null>
AllDifferent {vin:Red, vir...		No	Delete		<null>
AllDifferent {vin:Light, v...		No	Delete		<null>
AllDifferent {p1:Light, p...		No	Delete		<null>
AllDifferent {p1:Sweet, p...		No	Delete		<null>
p1:MendocinoRegion	p1:MendocinoRegion	No	Map	Directly-changed	frame name and type are the same

Abbildung 5.2: PromptDiff Ergebnis 'Elemente hinzufügen und löschen'

GermanyRegion, *SchlossRothermel*, *SchlossRothermelTrochenbierenausleseRiesling* werden als gelöscht gelistet. Weiters wird eine Änderung an der Instanz *MendocinoRegion* angeführt.

5.2.3 *Elemente hinzufügen, löschen und umbenennen*

Ergänzend zu den unter 5.2.1 und 5.2.2 durchgeführten Änderungen, wurde in der Ursprungsontologie, in Zeile 587 der Name der Klasse *Sancerre* auf *Sancarre* geändert.

```
<owl:Class rdf:ID="Sancarre">
```

Außerdem wurde die Definition der Instanz *ClosDeLaPoussieSancerre*, in den Zeilen 1857 bis 1859, angepasst.

```
<Sancarre rdf:ID="ClosDeLaPoussieSancerre">
  <hasMaker rdf:resource="#ClosDeLaPoussie" />
</Sancarre>
```

Diese Änderungen stellen die Ausbesserung eines Tippfehlers beim Namen einer Klasse, also eine Umbenennung dar.

OntoCompare führt in dem erzeugten Edit-Skript 39 Operationen an. Zur Erhöhung der Übersichtlichkeit wurden die Operationen wieder nummeriert.

```
1 RenameNode 1736-SevreEtMaineMuscadet(3)
2 RenameNode 2057-MuscadetRegion(3)
3 RenameNode 372-Sancerre(1)
4 RenameNode 273-locatedIn has MuscadetRegion(1)
5 InsertNode adjacentRegion
    Parents: virtual_root
6 InsertNode madeFromGrape max 1
```

```
      Parents: virtual_root
 7 RemoveSlot GermanyRegion at node locatedIn
      has MuscadetRegion
 8 InsertSlot MuscadetRegion at node locatedIn
      has MuscadetRegion
 9 InsertNode madeFromGrape has PinotBlancGrape
      Parents: virtual_root
10 InsertNode hasSugar has Dry
      Parents: virtual_root
11 InsertNode hasFlavor has Delicate
      Parents: virtual_root
12 InsertNode hasBody has Light
      Parents: virtual_root
13 InsertNode hasBody has Full
      Parents: virtual_root
14 InsertNode hasFlavor only {Moderate Strong}
      Parents: virtual_root
15 InsertNode Muscadet
      Parents: owl:Thing,
        madeFromGrape has PinotBlancGrape,
        hasBody has Light,
        hasFlavor has Delicate,
        hasSugar has Dry,
        madeFromGrape max 1
16 RemoveSlot Full at node SevreEtMaineMuscadet
17 RemoveSlot Moderate at node SevreEtMaineMuscadet
18 RemoveSlot SchlossVolrad at node
       SevreEtMaineMuscadet
19 InsertSlot SevreEtMaine at node
       SevreEtMaineMuscadet
20 RemoveSlot Sweet at node SevreEtMaineMuscadet
21 RemoveSlot GermanyRegion at node
       SevreEtMaineMuscadet
22 InsertEdge from node Muscadet to node
      SevreEtMaineMuscadet of type IS_A
23 RemoveEdge from node SweetRiesling to node
      SevreEtMaineMuscadet
24 InsertSlot SonomaRegion at node MendocinoRegion
25 InsertSlot LoireRegion at node MuscadetRegion
26 RemoveSlot Sancarre at node Loire and (locatedIn
       has SancerreRegion)
27 InsertSlot Sancerre at node Loire and (locatedIn
       has SancerreRegion)
28 InsertNode Loire and (locatedIn has MuscadetRegion)
      Parents: Muscadet,
29 InsertNode SweetRiesling
      Parents: hasFlavor only {Moderate Strong},
      DessertWine,
      hasBody has Full
30 InsertEdge from node SweetRiesling to node Riesling
       and (hasSugar has Sweet) of type IS_A
31 RemoveEdge from node SweetRiesling to node Riesling
       and (hasSugar has Sweet)
32 DeleteNode Wine and (locatedIn has GermanyRegion)
33 DeleteNode GermanWine
34 DeleteNode
       SchlossRothermelTrochenbierenausleseRiesling
35 DeleteNode SweetRiesling
36 DeleteNode SchlossVolrad
37 DeleteNode SchlossRothermel
38 DeleteNode hasFlavor only {Moderate Strong}
39 DeleteNode hasBody has Full
```

Operation Nummer 3 weist die Umbenennung der Klasse *Sancarre* aus. Die Umbenennung des Verweises auf die Klasse bei der Definition der Instanz *ClosDeLaPoussieSancerre* wird als *RemoveSlot* (Operation Nummer 26) und *InsertSlot*

Metadata (wine) | OWLClasses | Properties | Individuals | Forms | Prompt

Table view | Tree view

Image table

f1	f2	renamed	operation	map level	rename explanation
	p1:Muscadet	No	Add		<null>
	p1:adjacentRegion	No	Add		<null>
	AllDifferent {vin:Bancroft...	No	Add		<null>
	AllDifferent {p1:Delicate,...	No	Add		<null>
	AllDifferent {vin:Light, vi...	No	Add		<null>
	AllDifferent {vin:Sweet, vi...	No	Add		<null>
	AllDifferent {p1:Sweet, p:...	No	Add		<null>
	AllDifferent {vin:Delicate,...	No	Add		<null>
	AllDifferent {p1:Light, p1...	No	Add		<null>
	AllDifferent {p1:Red, p1:\...	No	Add		<null>
	AllDifferent {p1:Bancroft,...	No	Add		<null>
	AllDifferent {vin:Red, vin:...	No	Add		<null>
	p1:MuscadetRegion	No	Add		<null>
	p1:SevreEtMaineMuscadet	No	Add		<null>
AllDifferent {p1:Sweet,...		No	Delete		<null>
AllDifferent {vin:Sweet,...		No	Delete		<null>
p1:GermanWine		No	Delete		<null>
AllDifferent {p1:Red, p...		No	Delete		<null>
p1:SchlossRothermelT...		No	Delete		<null>
p1:SchlossVolrad		No	Delete		<null>
AllDifferent {p1:Light, ...		No	Delete		<null>
AllDifferent {vin:Delica...		No	Delete		<null>
AllDifferent {vin:Bancr...		No	Delete		<null>
p1:SchlossVolradTroc...		No	Delete		<null>
p1:SchlossRothermel		No	Delete		<null>
AllDifferent {vin:Light, ...		No	Delete		<null>
AllDifferent {vin:Red, v...		No	Delete		<null>
AllDifferent {p1:Delicat...		No	Delete		<null>
AllDifferent {p1:Bancrc...		No	Delete		<null>
p1:GermanyRegion		No	Delete		<null>
p1:Sancarre	p1:Sancerre	Yes	Map	Directly-changed	multiple unmatched siblings with similar...
p1:MendocinoRegion	p1:MendocinoRegion	No	Map	Directly-changed	frame name and type are the same
p1:ClosDeLaPoussieSa...	p1:ClosDeLaPoussieSanc...	No	Map	Isomorphic	frame name and type are the same

Abbildung 5.3: PromptDiff Ergebnis 'Elemente hinzufügen, löschen und umbenennen'

(Operation 27) dargestellt, da OntoCompare eine Operation zur Umbenennung eines Slots nicht kennt (siehe 4.1.1). Alle anderen Operationen sind ident mit den unter 5.2.2 angeführten Operationen.

PromptDiff zeigt zusätzlich zu den bereits unter 5.2.2 angeführten Operationen die Umbenennung der Klasse *Sancarre* und eine Änderung an der Instanz *ClosDeLaPoussieSancerre* an. Die Änderung an der Instanzdefinition hat den Ähnlichkeitsgrad *isomorphic*.

5.2.4 *Bewertung der Genauigkeitsprüfungen*

Die Genauigkeitsprüfung zeigt Qualitätsunterschiede.

Beide Vergleichsprogramme bilden OWL-DL Ontologien vollständig in ihrer internen Darstellung ab, alle Änderungen an den Dateien führen zu Änderungen in den internen Daten und werden erkannt. Abgesehen von der Menge an Operationen die für eine Änderung ausgewiesen werden ergeben sich auch Unterschiede in der Qualität der Erkennung.

Der häufigste Fall von Änderungen zwischen zwei Versionen einer Ontologie, das Erweitern um zusätzliche Klassen, Instanzen und Eigenschaften wird von beiden Verfahren einwandfrei erkannt. Mit ein wenig Erfahrung bezüglich der Darstellung dieser Änderungen durch das jeweilige Verfahren, kann man Erweiterungen an Ontologien aufgrund der Ausgabe der Programme schön nachvollziehen.

Bei gleichzeitigen Ergänzungen und Löschungen von Elementen von einer betrachteten Ontologieversion zur nächsten, hat OntoCompare Schwierigkeiten mit der Identifizierung identer Objekte. Das Verfahren wertet das Löschen eines Knotens und das Hinzufügen eines Knotens

an der selben Stelle als Umbenennung. Ein Herauslesen der tatsächlichen Änderung aus dem Inhalt des Edit-Skripts ohne die Änderung zu kennen ist dadurch unmöglich. Eine Änderung der Position eines Elementes im Graphen macht aus dem gleichen Grund Probleme. OntoCompare reagiert mit Lösch- und Einfüge-Operationen dieses Elementes auf eine solche Situation, da es die gleiche Identität dieses Elementes nicht feststellen kann. Der Grund für die Probleme beim Identifizieren identischer Objekte in zwei Ontologieversionen liegt im parsing der OWL-Dateien mittels Protégé OWL-API. Das API wandelt primär in die interne Datendarstellung von Protégé um, beim Erstellen des RDAG für OntoCompare ist man auf den Namen des Elementes als wesentliches Identifizierungsmerkmal angewiesen. PromptDiff leistet sich keine Schwäche und erkennt alle Änderungen einwandfrei.

Die Umbenennung einer Klasse und der Referenz bei der Instanzdefinition wurde von OntoCompare erkannt und nachvollziehbar ausgewiesen. PromptDiff erkennt ebenfalls die Umbenennung der Klasse, gibt aber für die Änderung der Instanzdefinition als Ähnlichkeitsgrad *isomorphic* an. Würde man die Interaktion mit einem Nutzer nicht wünschen und PromptDiff als vollautomatisches System betreiben, müsste man einen Filter auf alle ausgewiesenen Änderungen anwenden, der alle Einträge mit Änderungsgrad *isomorphic* verwirft. In diesem Falle hätte das Verfahren eine stattgefundene Änderung nicht gemeldet.

5.3 LEISTUNGSPRÜFUNG

Die Leistungsprüfungen wurden auf einem Rechner, der mit zwei Intel Xeon 3.00GHz Prozessoren und 1GB Hauptspeicher ausgerüstet war, durchgeführt. Sowohl von PromptDiff als auch von OntoCompare wurde nur ein Prozessor ausgelastet, dieser aber zu 100%. Das eingesetzte Betriebsystem war Debian Linux in der Version 4.1.1-21, die verwendete JAVA-Version war SUN JDK 1.5.0_14-b03. Die verwendete Version von PromtDiff und dem Protégé-OWL-API waren gleich der Testumgebung bei 5.2.

5.3.1 *Zur Leistungsprüfung verwendete Ontologien*

Die verwendeten Ontologien stammen aus dem Gene-Ontology-Project[4], sowie aus dem SoPharm-Projekt[5], sind also im wissenschaftlichen Umfeld verwendete und gepflegte Ontologien, die ständigen Änderungen unterworfen sind.

Als Maß für den Umfang einer Ontologie wird die Anzahl an RDF-Ressourcen angegeben, jede Änderung entspricht der Änderung an einer RDF-Ressource. Für die Berechnung der Änderungsrate wurden die von OntoCompare ermittelten Werte herangezogen.

Die verwendeten Gene-Ontologien weisen einen geringeren Umfang und eine geringere Änderungsrate zwischen den Versionen auf als die SoPharm-Ontologien. Das Gene-Ontology-Project verwendet für Ontologien das OBO-Datenformat, die OWL-Dateien werden für etwaige Interessenten daraus erzeugt und auf täglicher Basis zur Verfügung gestellt, das Projekt selbst verwendet sie nicht. Die täglich erstellten OWL-Versionen werden nicht archiviert, es werden nur die zu Monats-

4 Siehe dazu Gene Ontology Website `http://www.geneontology.org`

5 Siehe dazu SO-Pharm Ontology Website `http://www.loria.fr/~coulet/sopharm2.0_description.php`

beginn erstellten Dateien aufgehoben. Es wurden Ontologie-Versionen gewählt, deren Umfang denen der SoPharm-Ontologien entspricht. Da erst mit 05.03.2007 begonnen wurde, die täglich erstellte OWL-Datei zu sichern und die Versionen nach dem 09.03.2007 rasant an Größe zunahmen, beschränkten sich die geeigneten Versionen auf die Stände vom 05.03.2007, 06.03.2007 und 09.03.2007. Die Eckdaten der verwendeten Gene-Ontologien, die Anzahl der Klassen (inklusive anonymer Klassen und Systemklassen), die Anzahl der Instanzen, die Anzahl der Eigenschaften, die Anzahl aller RDF-Ressourcen, sowie die Anzahl der Tripel nach Umwandlung in das Protégé-Format finden sich in der Tabelle 5.1. Die Anzahl der Tripel wird während des parsens der Datei durch das OWL-API ermittelt und in Schritten zu 5.000 Stück angegeben. Der in der Tabelle eingetragene Wert bedeutet also mehr als x Tripel aber weniger als x+5.000 Tripel. Von der SoPharm-Ontologie sind

Bezeichnung	Klassen	Instanzen	Eigenschaften	RDF-Ressourcen	Tripel
GO_20070305	28.447	43	51	28.543	120.000
GO_20070306	28.449	43	51	28.545	120.000
GO_20070309	28.497	43	51	28.593	120.000

Tabelle 5.1: Eckdaten der Gene-Ontologien

nur drei Versionen, Version 1.0, 1.2 und 1.3 verfügbar. Die Zeiträume, die zwischen der Erstellung der einzelnen Versionen liegen sind nicht bekannt. Kurz vor Abschluß dieser Arbeit wurde noch die Version 2.0 der SoPharm-Ontologie veröffentlicht, da sie sich aber noch im Alpha-Stadium befunden hatte, wurden keine Testläufe mit ihr durchgeführt. Die Tabelle 5.2 zeigt die Eckdaten der verwendeten SoPharm-Ontologie Versionen.

Bezeichnung	Klassen	Instanzen	Eigenschaften	RDF-Ressourcen	Tripel
SO_1.0	39.739	43	113	40.004	120.000
SO_1.2	40.297	44	160	40.738	125.000
SO_1.3	39.976	44	195	40.451	125.000

Tabelle 5.2: Eckdaten der SoPharm-Ontologien

5.3.2 *Leistungsprüfung OntoCompare*

Wie in 4.1.3 beschrieben, arbeitet der Algorithmus in verschiedenen Phasen. Die einzelnen Phasen wurden in eine Vorkalkulations-Phase und eine Kalkulations-Phase zusammengefasst.

Vorkalkulation Alle Berechnungen, die vom Auslösen des Vergleichslaufes bis zum Beginn des eigentlichen Vergleichs-Algorithmus durchgeführt werden sind in der Vorkalkulations-Phase zusammengefasst. Das sind konkret:

- das Einlesen beider OWL-Dateien
- der Aufbau der RDAGs aus den eigelesenen Objekten
- die Erzeugung der Hilfsdatenstrukturen für den folgenden Vergleichslauf

Kalkulation Alle Phasen die der Ermittlung der Änderungen dienen, sind der Kalkulationsphase zugeordnet.

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Einlesen der OWL-Dateien	57,4	50,09%
Aufbau der RDAGs	52,3	45,63%
Erzeugung Hilfsstrukturen	1,0	0,84%
Ermittlung ähnlicher Knoten	0,6	0,56%
Ermittlung umbenannter Knoten	1,1	0,92%
Ermittlung der Änderungen	2,2	1,96%
Summe	114,7	

Tabelle 5.3: Werte des OntoCompare-Vergleichslaufes GO_20070306 mit GO_20070305

- die Ermittlung ähnlicher Knoten
- die Ermittlung umbenannter Knoten
- Knoten-Einfügephase
- Knoten-Änderungsphase
- Slot-Änderungsphase
- Kanten-Änderungsphase
- Löschphase
- Erzeugung des Edit-Skripts aus den internen Speicherstrukturen

Die in den folgenden Ergebnissen der Vergleichsläufe angegebenen Zeiten wurden während der Durchführung des Laufes vom Algorithmus ermittelt und auf der Konsole ausgegeben.

Der Vergleich der Ontologie in der Version GO_20070306 mit der Ursprungsontologie GO_20070305 liefert 25 Operationen im Edit-Skript. Bei einer Gesamtzahl an RDF-Elementen der Ursprungsontologie von 28.543 ergibt sich eine Änderungsrate von 0,09%. Die Gesamtdauer des Vergleichslaufes betrug 114,7 Sekunden, wobei mehr als 95% für das Einlesen der OWL-Dateien und den Aufbau der RDAGs benötigt wurden. Die Kalkulationsphase war in weniger als 4% der Zeit abgeschlossen. Beim Vergleich der Ontologie GO_20070309 mit der Ontologie GO_20070305 werden 264 Operationen im Edit-Skript ausgewiesen. Der Gesamtumfang der Ursprungsontologie war 28.543 RDF-Elemente, woraus sich eine Änderungsrate von 0,92% ergibt. Auch in diesem Fall war die benötigte Zeit für das Einlesen der Daten und die Umwandlung in Graphen mit über 95% der Gesamtzeit der größte Posten in der Zeitaufstellung. An der Gesamtzeit von 114,7 Sekunden, die für den Vergleichslauf benötigt wurde, hatte die reine Kalkulationszeit von 3,9 Sekunden einen Anteil von nur knapp 4%. Der Vergleich der Ontologien GO_20070309 mit der Ontologie GO_20070306 ergab 239 Operationen. Bei einem Gesamtumfang von 28.545 RDF-Elementen in der Ursprungsontologie ergibt sich eine Änderungsrate von 0,84%. Von der Gesamtdauer des Vergleichslaufes von 114,2 Sekunden benötigte das Einlesen der Dateien und das Erstellen der Graphen mehr als 95%, die Kalkulation des Ergebnisses weniger als 4% der Zeit. Der Vergleich der SoPharm-Ontologie SO_1.2 mit der Ursprungsontologie 1.0 ergibt 2.341 eingetragene Operationen im Edit-Skript. Durch den Umfang von 40.004 RDF-Elementen der Ursprungsontologie ergibt sich eine Änderungsrate von 5,85%. Die Gesamtlaufzeit betrug 2.270,1 Sekunden, wobei mit 91,19% der Hauptteil der Zeit für die Ermittlung der umbenannten Knoten aufgewendet wurde. Die benötigte Zeit für

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Einlesen der OWL-Dateien	57,2	49,98%
Aufbau der RDAGs	52,0	45,41%
Erzeugung Hilfsstrukturen	1,0	0,85%
Ermittlung ähnlicher Knoten	0,6	0,56%
Ermittlung umbenannter Knoten	1,4	1,26%
Ermittlung der Änderungen	2,2	1,94%
Summe	114,5	

Tabelle 5.4: Werte des OntoCompare-Vergleichslaufes GO_20070309 mit GO_20070305

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Einlesen der OWL-Dateien	57,0	50,18%
Aufbau der RDAGs	52,8	45,26%
Erzeugung Hilfsstrukturen	1,0	0,84%
Ermittlung ähnlicher Knoten	0,6	0,56%
Ermittlung umbenannter Knoten	1,1	1,20%
Ermittlung der Änderungen	2,2	1,96%
Summe	114,2	

Tabelle 5.5: Werte des OntoCompare-Vergleichslaufes GO_20070309 mit GO_20070306

Vorbereitungen zur Kalkulation hat einen Anteil von 8,7% an der Gesamtdauer. Beim Vergleich der Ontologie SO_1.3 mit der Ontologie SO_1.0 wurden 3.891 Änderungen ermittelt. Bei insgesamt 40.004 RDF-Elementen ergibt sich eine Änderungsrate von 9,73%. Mit mehr als 98% nimmt die Phase der Ermittlung der umbenannten Knoten den weitaus größten Teil des 13.139,2 Sekunden dauernden Vergleichslaufes in Anspruch. Der Vergleich der Ontologie SO_1.3 mit der Ontologie SO_1.2 ergab 1.315 Änderungen bei 40.738 RDF-Elementen. Die Änderungsrate betrug daher 3,23%. Von der Gesamtdauer von 413,6 Sekunden benötigte die Vorkalkulation mehr als 55% der Zeit. Die Ermittlung der umbenannten Knoten hatte einen Anteil von 43,36%.

5.3.3 *Leistungsprüfung PromptDiff*

Der Ablauf eines Vergleichslaufes von PromptDiff läßt sich in eine Vorkalkulationsphase, eine Kalkulationsphase und eine Nachkalkulati-

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Einlesen der OWL-Dateien	46,5	2,05%
Aufbau der RDAGs	149,3	6,58%
Erzeugung Hilfsstrukturen	1,6	0,07%
Ermittlung ähnlicher Knoten	0,9	0,04%
Ermittlung umbenannter Knoten	2070,1	91,19%
Ermittlung der Änderungen	1,8	0,08%
Summe	2270,1	

Tabelle 5.6: Werte des OntoCompare-Vergleichslaufes SO_1.2 mit SO_1.0

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Einlesen der OWL-Dateien	47,1	0,36%
Aufbau der RDAGs	164,4	1,25%
Erzeugung Hilfsstrukturen	1,8	0,01%
Ermittlung ähnlicher Knoten	0,7	0,01%
Ermittlung umbenannter Knoten	12.923,5	98,36%
Ermittlung der Änderungen	1,7	0,01%
Summe	13.139,2	

Tabelle 5.7: Werte des OntoCompare-Vergleichslaufes SO_1.3 mit SO_1.0

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Einlesen der OWL-Dateien	53,4	12,91%
Aufbau der RDAGs	176,7	42,72%
Erzeugung Hilfsstrukturen	1,6	0,39%
Ermittlung ähnlicher Knoten	0,9	0,21%
Ermittlung umbenannter Knoten	179,3	43,36%
Ermittlung der Änderungen	1,7	0,42%
Summe	413,6	

Tabelle 5.8: Werte des OntoCompare-Vergleichslaufes SO_1.3 mit SO_1.2

onsphase unterteilen. In der Vorkalkulationsphase werden das Zielprojekt und das Ursprungsprojekt geladen, die Kalkulationsphase enthält den eigentlichen Vergleichslauf, in der Nachkalkulationsphase läßt PromptDiff einen *Change Analyzer* laufen und schreibt eine Log-Datei. Wie unter 5.2 beschrieben müssen das Ursprungs- und das Zielprojekt mittels Einlesen der OWL-Dateien und Abspeichern des Projektes erstellt werden. Die Zeit für diese Tätigkeiten wird bei der Leistungsermittlung nicht berücksichtigt. Die Zeitwerte für die Kalkulations- und Nachkalkulationsphase werden von PromptDiff nur sekundengenau angegeben.

Der Vergleich der Ontologie GO_20070306 mit der Ontologie GO_20070305 dauerte gesamt 103,5 Sekunden. Davon beanspruchte die Vorkalkulation 64,29% der Zeit. Auch die Nachkalkulation hatte mit 20,29% einen höheren Anteil an der Gesamtdauer als die Vergleichsphase mit 15,46%.

Der Vergleichslauf zwischen der Ontologie GO_20070309 und der Ontologie GO_20070305 benötigte 109,1 Sekunden. Daran hatte die Vorkalkulation einen Anteil von 62,42%, die Kalkulation einen Anteil von 14,67% und die Nachkalkulation einen Anteil von 22,91%.

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Vorkalkulation	66,5	64,29%
Kalkulation	16,0	15,46%
Nachkalkulation	21,0	20,29%
Summe	103,5	

Tabelle 5.9: Werte des PromptDiff-Vergleichslaufes GO_20070306 mit GO_20070305

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Vorkalkulation	68,1	62,42%
Kalkulation	16,0	14,67%
Nachkalkulation	25,0	22,91%
Summe	109,1	

Tabelle 5.10: Werte des PromptDiff-Vergleichslaufes GO_20070309 mit GO_20070305

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Vorkalkulation	67,8	62,89%
Kalkulation	17,0	15,77%
Nachkalkulation	23,0	21,34%
Summe	107,8	

Tabelle 5.11: Werte des PromptDiff-Vergleichslaufes GO_20070309 mit GO_20070306

Bei dem Vergleich der Gene-Ontologien GO_20070309 mit der Gene-Ontologie GO_20070306 benötigte PROMPTDIFF 107,8 Sekunden für die Ermittlung des Ergebnisses. An der Gesamtdauer hatte die Vorkalkulationsphase einen Anteil von 62,89%, die Kalkulationsphase eine Anteil von 15,77% und die Nachkalkulation einen Anteil von 21,34%. Beim Vergleich der SoPharm-Ontologien SO_1.2 mit SO_1.3 ergab sich eine Gesamtlaufzeit von 177,1 Sekunden. Der Anteil der Vorkalkulation lag bei 62,89%, der der Kalkulation bei 15,77% und der Anteil der Nachkalkulation lag bei 21,34%. Der Vergleich der So-Pharm Ontologie SO_1.3 mit der Ontologie SO_1.0 dauerte gesamt 186,5 Sekunden. Daran hatte die Vorkalkulation einen Anteil von 24,40%, die Kalkulation Bei dem Vergleich der Ontologien SO_1.3 mit SO_1.2 benötigte der Algorithmus 206,2 Sekunden. Dabei entfielen 23,38% auf die Vorkalkulationsphase, 23,28% auf die Kalkulationsphase und 53,35% auf die Nachkalkulationsphase.

5.3.4 *Bewertung der Leistungsprüfung*

Die Ontologien aus dem Gene-Ontology-Projekt und dem SoPharm-Projekt stellen Wissensrepräsentationen dar, die sich im wissenschaftlichen Umfeld im Einsatz befinden. Deren Aufbau und die Änderungen zwischen den Versionen entstammen einer realen Verwendung. Alle Ontologien waren von ähnlichem Umfang, die Änderungsraten reichten von knapp 1% bis zu knapp 10% zwischen zwei Versionen. Tabelle 5.15 zeigt eine Übersicht über den Umfang der Ursprungsontologie, der Anzahl der durchgeführten Änderungen in der Vergleichsontolo-

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Vorkalkulation	45,1	62,89%
Kalkulation	44,0	15,77%
Nachkalkulation	88,0	21,34%
Summe	177,1	

Tabelle 5.12: Werte des PromptDiff-Vergleichslaufes SO_1.2 mit SO_1.0

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Vorkalkulation	45,5	24,40%
Kalkulation	43,0	23,06%
Nachkalkulation	98,0	52,55%
Summe	186,5	

Tabelle 5.13: Werte des PromptDiff-Vergleichslaufes SO_1.3 mit SO_1.0

Phase	Dauer in Sek.	Anteil an Gesamtdauer
Vorkalkulation	48,2	23,38%
Kalkulation	48,0	23,28%
Nachkalkulation	110,0	53,35%
Summe	206,2	

Tabelle 5.14: Werte des PromptDiff-Vergleichslaufes SO_1.3 mit SO_1.2

gie und die Änderungsrate als das Verhältnis der Änderungen zum Umfang. Die Änderungsraten von in etwa 1% zwischen Versionen der Gen-Ontologien entsprechen den üblicherweise bei anderen Ontologien beobachteten Werten an Änderungen zwischen zwei Versionen. Die hohen Änderungsraten zwischen den Versionen der SoPharm-Ontologien sind durch einen längeren Versionierungszyklus erklärbar.

Das Einlesen der OWL-Dateien und die Ermittlung umbenannter Knoten bedingen bei OntoCompare hohen Rechenaufwand.

Bei den für OntoCompare ermittelten Werten zeigt sich, daß die Vorkalkulationsphase viel Zeit beansprucht. Insbesonders die Erstellung der Graphen aus den geparsten OWL-Objekten ist aufwändig. Bei üblichen Änderungsraten von etwa 1% beträgt der Rechenaufwand für das parsen der OWL-Datei und die Erstellung der Graphen in etwa 95% des Gesamtaufwandes. Bei höheren Änderungsraten verschiebt sich die Aufteilung des Rechenaufwandes in Richtung der Kalkulationsphase konkret zur Ermittlung der umbenannten Knoten. Bei einer Änderungsrate von knapp 10% hat die Phase der Emittlung der umbenannten Knoten einen Anteil von mehr als 98% an der Gesamtdauer des Vergleichslaufes.

Bei PromptDiff steigt der Aufwand für die Kalkulationsphase bei den durchgeführten Vergleichen von knapp 15% bei den kleinsten Ontologien zu knapp 24% bei den größten Ontologien an. Die Verteilung der übrigen Zeit wandert mit steigendem Umfang der untersuchten Ontologien von der Vorkalkulationsphase zur Nachkalkulationsphase. Ist das Verhältnis Vor- zu Nachkalkulation bei den kleinsten untersuchten Ontologien 64 zu 20, so liegt es bei den größten Ontologien im Test bei 23 zu 53. Die Änderungsrate wirkt sich nicht auf den Rechenaufwand aus.

Vergleichsontologie	Ursprungsontologie	Umfang	Änderungen	Änderungsrate
GO_20070306	GO_20070305	28.543	25	0,09%
GO_20070309	GO_20070305	28.543	264	0,92%
GO_20070309	GO_20070306	28.545	239	0,84%
SO_1.2	SO_1.0	40.004	2.341	5,85%
SO_1.3	SO_1.0	40.004	3.891	9,73%
SO_1.3	SO_1.2	40.738	1.315	3,23%

Tabelle 5.15: Änderungsraten der Ontologien

Vergleichsontologie	Ursprungsontologie	Dauer OC	Dauer PD	Geschwindigkeitsfaktor
GO_20070306	GO_20070305	114,7	103,5	1,11
GO_20070309	GO_20070305	114,5	109,1	1,05
GO_20070309	GO_20070306	114,2	107,8	1,06
SO_1.2	SO_1.0	2.270,1	177,1	12,86
SO_1.3	SO_1.0	13.139,2	186,5	70,45
SO_1.3	SO_1.2	413,6	206,2	2,01

Tabelle 5.16: Laufzeitunterschiede zwischen den Vergleichsverfahren

Der Vergleich zwischen OntoCompare und PromptDiff zeigt, daß bei kleinen Änderungsraten die Verfahren in etwa gleich schnell sind mit leichten Vorteilen für PromptDiff. Betrachtet man nur die Kalkulations- und Nachkalkulationsphase, so ist OntoCompare bis zu zehn mal schneller als PromptDiff. Natürlich ist die Vorkalkulationsphase ein Teil des Vergleichslaufes, man muß aber bedenken, daß die Erstellung der Projekte, also das erstmalige Einlesen der OWL-Dateien in PromptDiff bei den Zeiten nicht berücksichtigt wurde. Während dieses Vorganges verändert PromptDiff die ursprüngliche OWL-Datei um eigene Hilfsdaten. Ein eventuell vorhandener Geschwindigkeitsvorteil bei neuerlichem Laden der Daten kann nicht nachgewiesen werden, ist aber zu vermuten. OntoCompare bedient sich zum parsen der OWL-Dateien so wie PromptDiff dem Protégé OWL-API. Das API erzeugt während des parsing-Vorganges schon die für Protégé geeigneten Datenstrukturen, ein Aufwand der für OntoCompare nicht notwendig ist, aber nicht unterbunden werden kann. Bei großen Änderungsraten ist der Aufwand der Vorkalkulationsphase bei OntoCompare nicht mehr relevant. Hier zeigt sich die Ermittlung der umbenannten Knoten als rechenintensive Vergleichsoperation, OntoCompare benötigte insgesamt die bis zu 70-fache Rechenzeit gegenüber PromptDiff. Die quadratische Abhängigkeit des Rechenaufwandes von der Anzahl der Änderungen bei OntoCompare wird durch diese Werte deutlich. Die Tabelle 5.16 zeigt eine Aufstellung der benötigten Rechenzeit von OntoCompare (Dauer OC) und PromptDiff (Dauer PD) für die Vergleiche der Vergleichsontologie mit der Ursprungsontologie und den Faktor, um den PromptDiff dabei schneller war als OntoCompare.

6 ZUSAMMENFASSUNG

Im gleichen Maße in dem die Verwendung von Ontologien steigt, steigt auch die Notwendigkeit die Änderungen zwischen den Versionen dieser Ontologien zu ermitteln. Da bei Ontologien die Struktur der Daten verglichen werden muß, um die Veränderungen zu ermitteln, reichen Methoden für Dokumentenvergleiche nicht aus.

Ontologien können in verschiedenen Notationsprachen erstellt werden. Durch die Bemühungen des W3C scheint sich die Web Ontology Language OWL als Standard durchzusetzen.

Im Rahmen dieser Arbeit wurde ein Modul entwickelt, das es ermöglicht, das Vergleichsverfahren OntoCompare auf Ontologien, die in der OWL Notation erstellt sind, anzuwenden. Dadurch wird es möglich, Vergleichsläufe auf Ontologieversionen, wie sie in der realen Welt entstehen, durchzuführen. Die Tests zur Ermittlung der Ergebnisdarstellung lieferten ausgezeichnete Resultate. Die durchgeführten Änderungen in Ontologien wurden einwandfrei erkannt und ausgegeben. Mitunter werden die ermittelten Operationen durch die interne Datenstruktur des Vergleichsverfahrens beeinflußt, das Wissen um die interne Repräsentation einer Ontologie, um angezeigte Änderungen exakt benennen zu können ist daher hilfreich. Dies gilt aber nicht nur im Zusammenhang mit OntoCompare, sondern auch bei anderen Vergleichsverfahren wie etwa PromptDiff. Eventuell könnte ein Vergleichsverfahren, das direkter an der Datenstruktur von OWL operiert, diese Einschränkung beheben. Die Vergleichsläufe des Referenzverfahrens PromptDiff lieferten ebenfalls sehr gute Werte. Einzig die Verarbeitung des *owl:AllDiffernt* Konstruktes zur Definition von Instanzen muß verbessert werden.

Bei der Genauigkeitsprüfung wird die Verwendung des Protégé-OWL-APIs für das Einlesen der OWL-Dateien zum Nachteil von OntoCompare. Durch das nicht auf die internen Datenstruktur abgestimmte parsing fällt es dem Vergleichsverfahren schwer, identische Elemente zu erkennen. Ein für die Erstellung eines RDAG optimierter Parser könnte diese Probleme vermeiden. PromptDiff erkennt alle Veränderungen richtig, gibt aber in einem Fall eine Änderungsgrad an, der in einer nicht interaktiven Anwendung eine Änderung unerkannt lassen würde.

Ein optimierter OWL-Parser sollte erhebliche Vorteile bei der Genauigkeit und der Geschwindigkeit von OntoCompare bringen.

Wenn Ontologien große oder sehr komplexe Wissensgebiete beschreiben, dann nimmt ihr Umfang rasant zu. Aus diesem Grund spielt die Leistungsfähigkeit von Vergleichsverfahren eine große Rolle. Diese Arbeit zeigt, daß die Leistungsfähigkeit von OntoCompare bei Versionsunterschieden im üblichen Bereich durchaus mit der des Referenzverfahrens PromptDiff mithalten kann. Ein anteilsmäßig hoher Rechenaufwand entsteht beim Einlesen der Daten und der Umwandlung in die Datenstruktur von OntoCompare. Die Entwicklung eines eigenen OWL-Parsers mit möglicherweise optimiertem Vorgehen in Hinblick auf die Erstellung eines RDAG könnte die Vorteile in der Kalkulationsphase gegenüber PromptDiff nutzbar machen. Bei hohen Änderungsraten zwischen den zu vergleichenden Ontologien ist aber der Vergleichsalgorithmus selbst das begrenzende Element. Eine Erhöhung der Änderungsrate wirkt sich bei OntoCompare quadratisch auf

den Rechenaufwand aus, hier ist das Referenzverfahren PromptDiff klar im Vorteil. Für eine Verbesserung des Verhaltens wäre bei der Phase zur Ermittlung der umbenannten Knoten anzusetzen, dieser Teil des Algorithmus müßte effizienter gelöst werden.

[1] V.K. Chaudhri, A. Farquhar, R. Fikes, P.D. Karp, and J.P. Rice. Okbc: A programmatic foundation for knowledge base interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. AAAI Press/The MIT Press, 1998.

[2] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 493–504, 1996.

[3] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal*, 12:303–319, 2003.

[4] Wissenschaftlicher Rat der Dudenredaktion. *Duden Fremdwörterbuch*, volume 5. Dudenverlag, 1990.

[5] Johann Eder and Karl Wiggisser. Change detection in ontologies using dag comparison. In *Posters of the 2006 ODBASE (Ontologies, Databases, and Applications of Semantics) International Conference*, volume 4277/2006, pages 42–43, 2006.

[6] Johann Eder and Karl Wiggisser. A dag comparison algorithm and its application to temporal data warehousing. In *In: Advances in Conceptual Modeling -Theory and Practice, ER Workshops 2006*, pages 217–226, 2006.

[7] Marc Ehrig and York Sure. Ontology mapping - an integrated approach. In *In Proceedings of the First European Sematic Web Symposium, ESWS 2004*, pages 76–91. Springer Verlag, 2004.

[8] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5, 1993.

[9] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal Human-Computer Studies*, 43:907–928, 1995.

[10] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building owl ontologies using the protege-owl plugin and co-ode tools. August 2004.

[11] Graham Klyne and Jeremy J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. Recommendation, W3C, Feb. 2004.

[12] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. Managing multiple ontologies and ontology evolution in ontologging. In *in ontologging. Intelligent Information Processing*, pages 51–63. Kluwer, 2002.

[13] Deborah L. McGuinness and Frank van Harmelen. *OWL Web Ontology Language Overview*. W3C, 2004.

[14] Natalya F. Noy and Michel Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6:428–440, 2004.

[15] Natalya F. Noy, Sandhya Kunnatur, Michel Klein, and Mark A. Musen. Tracking changes during ontology evolution. In *In Proceeding of the 3rd International Semantic Web Conference (ISWC2004)*, pages 259–273. Springer-Verlag, 2004.

[16] Natalya F. Noy and Mark A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *In 17th Nat. Conf. on Artificial Intelligence (AAAI-2000)*, 2000.

[17] Natalya F. Noy and Mark A. Musen. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *in Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, pages 744–750, 2002.

[18] Natalya F. Noy and Mark A. Musen. Ontology versioning as an element of an ontology-management framework. *IEEE Intelligent Systems*, 2003.

[19] Natalya F. Noy and Mark A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004.

[20] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. *OWL Web Ontology Language Guide*. W3C, 2004.

[21] Michal Tury and Maria Bielikova. An approach to detection ontology changes. In *ICWE'06 Workshops*, 2006.

KURZFASSUNG

Ontologien ermöglichen es, Wissen auf formale Art und Weise zu repräsentieren und machen es dadurch für Mensch und Maschine verarbeitbar. Die Veränderung des Wissens im Laufe der Zeit muß auch in den Ontologien mitvollzogen werden. Die derart entstehenden Ontologieversionen stellen Wissen zu bestimmten Zeitpunkten dar, meist ist aber keine Information über die Veränderungen zwischen den einzelnen Versionen verfügbar. Da aber oft Interesse an den Änderungen zwischen Ontologieversionen besteht, werden Vergleichsverfahren eingesetzt, die diese Änderungen ermitteln können.

Für die Speicherung von Ontologien haben sich mehrere Notationssprachen entwickelt, die bei Vergleichen unterschiedlich behandelt werden müssen. Mit der Notation OWL wurde ein einheitlicher Standard geschaffen.

Diese Arbeit beschreibt die Erweiterung des Vergleichsalgorithmus OntoCompare, um die Möglichkeit OWL-Ontologien zu verarbeiten und untersucht die Ergebnisse von Versionsvergleichen von Ontologien im OWL-Format. Um zu überprüfen, wie genau Unterschiede zwischen Ontologieversionen erkannt werden, wird die Referenz OWL-Ontologie des W3C verwendet und mittels gezielter Änderungen unterschiedliche Versionen davon erzeugt.

Neben der Genauigkeit der Änderungserkennung ist bei den zunehmend größer werdenden Ontologien auch die Leistungsfähigkeit eines Vergleichsverfahrens interessant. Zu diesem Zweck werden Versionsstände von OWL-Ontologien, die in der realen Welt entstanden, zu Testläufen herangezogen.

Da die Ergebnisse von OntoCompare teilweise unter Zuhilfenahme von Heuristiken ermittelt werden, ist anzunehmen, daß der Aufbau der verwendeten Ontologien und die Art und Menge der Änderungen zwischen zwei Versionen diese beeinflussen. Um eine Referenz für die ermittelten Ergebnisse zu bekommen, werden auch die Ergebnisse des Vergleichsverfahrens PromptDiff dargestellt.

B ABSTRACT

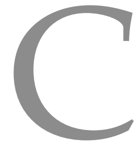
Ontologies represent knowledge in a formal way to make it usable by human and machine. The change in knowledge which occurs over time has to be represented in ontologies also. The version of one ontology represents knowledge at a certain point in time but the change between the versions is unknown. To compute the change between versions change detection procedures are used.

To store an ontology several notation-languages have evolved over time. Each of them has to be treated differently for change detection. The introduction of OWL has set a standard to overcome the shortcomings of other languages.

This work describes the enhancement of the change detection procedure OntoCompare to use OWL-ontologies and examines the results when comparing two ontologies in OWL-format. To test the correctness of the comparison results the reference OWL-ontology of the W3C is used as a basis and known changes are applied to it.

Beside the correctness of the results the performance of the algorithm becomes more important as ontologies grow in size. To test the performance real world OWL-ontology versions are used.

In some calculation steps OntoCompare uses heuristics to calculate a result. Therefore the results will likely depend on the used ontology and the number of changes. To have a reference point all calculations are also done by PromptDiff.



LEBENSLAUF

Persönliche Informationen

Nationalität: Österreich
Geburtsdatum: 23. März 1970
Geburtsort: Wien

Ausbildung

1997 - 2008 Universität Wien, Magister der Sozial- und Wirtschaftswissenschaften
Wirtschaftsinformatik

1990 - 2008 Technische Universität Wien
Informatik

1990 - 1997 Universität Wien, Erste Diplomprüfung Wirtschaftsinformatik

1984 - 1989 Höhere technische Bundeslehranstalt Wien IV
Matura Elektrotechnik mit gutem Erfolg

Berufserfahrung

Seit 10/2003 DATAMATIX Datensysteme GmbH, Technischer Leiter

Seit 02/1999 IT-Lab, Einzelunternehmen

Wehrdienst

10/1989 - 5/1990 Panzerstabsbataillon 9, Nachschub-Transport-Instandsetzung, Zwölfaxing