

Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Realisierung des COnto-Diff Algorithmus innerhalb eines
Protégé-Plugins

Bachelorarbeit

Leipzig, Oktober 2015

vorgelegt von

Swoboda, Oliver
Studiengang Bachelor Informatik

Betreuender Hochschullehrer: Dr. Anika Groß

Universität Leipzig, Institut für Informatik, Abteilung Datenbanken, Prof. Rahm

Inhaltsverzeichnis

1	Einleitung	3
1.1	Motivation	3
1.2	Ziele der Arbeit	5
1.3	Aufbau der Arbeit	6
2	Grundlagen & Verwandte Arbeiten	7
2.1	Ontologien	7
2.2	Ontologieevolution	8
2.3	COntoDiff	10
2.4	Ontologieeditoren	13
2.5	Abgrenzung der eigenen Arbeit	14
3	Konzept zur Entwicklung eines Protégé-Plugins für COntoDiff	16
3.1	Anforderungen	16
3.1.1	Internetunabhängige Umsetzung	16
3.1.2	Geringfügige Änderungen am COntoDiff-Algorithmus	17
3.1.3	Flexibler Import von Ontologieversionen	17
3.1.4	Unterstützung umfangreicher Diff- und Analyse-Funktionalitäten	17
3.1.5	Interaktion des Plugins mit den in Protégé geöffneten Ontologien	18
3.1.6	Export der Diff- und Analyseergebnisse	18
3.2	Auswahl eines geeigneten Plugintyps	18
3.3	Schematischer und struktureller Aufbau	21
3.4	Entwurf der Benutzeroberfläche	23
3.4.1	Starten des COntoDiff-Plugins	23
3.4.2	Einlesen der Ontologieversionen	24
3.4.3	Fortschritt der Berechnung	25
3.4.4	Ausgabe der Ergebnisse	25
4	Implementierung des COntoDiff-Plugins	30
4.1	Erstellen eines Protégé-Plugins	30
4.2	Architektur des Plugins	30
4.3	Anpassungen am COntoDiff-Algorithmus	31
4.4	Erstellung der Benutzeroberfläche	34
4.5	Veröffentlichung des COntoDiff-Plugins	35
5	Evaluierung des Plugins	37
5.1	Benutzeroberflächen von COntoDiff und Protégé OWL Diff	37
5.2	Auswertung der Ergebnisse	45
6	Kurzzusammenfassung und Ausblick	48
	Literaturverzeichnis	49
	Eidesstattliche Erklärung	52

1 Einleitung

1.1 Motivation

In der heutigen Zeit, in der immer mehr Datenquellen mit einer stetigen Zunahme an Informationen aus Bereichen, wie z.B. den Lebenswissenschaften oder der Informationsverarbeitung, zur Verfügung stehen, nehmen Ontologien [16] eine wichtige Rolle zur Beschreibung und Strukturierung von Domänenwissen ein. Eine Ontologie kann oft als gerichteter azyklischer Graph angesehen werden und wird zur Erfassung von Realweltobjekten und deren Beziehungen untereinander, z.B. im Semantic Web [2], genutzt. Mit ihrem standardisierten Vokabular aus Konzepten, Attributen und Relationen ist es möglich, Daten automatisiert nutzbar zu machen und verteilte Datenbestände gemeinsam zur Verfügung zu stellen.

Um Wissen aus mehreren Ontologien nutzen zu können, werden Mappings genutzt. Hierbei handelt es sich um eine Abbildung zwischen zwei Ontologien, bei der, z.B. durch einen (semi)automatischen Abgleich, festgehalten werden kann, welche Ontologiekonzepte aus Ontologie A den Konzepten aus Ontologie B entsprechen. Diese Art von Verknüpfung ist für die Integration mehrerer Ontologien oder zur Unterstützung übergreifender Analysen nötig. Weiterhin gibt es Mappings zwischen Instanzen oder Konzepten von Ontologien.

Annotationen dienen der semantischen Beschreibung der Eigenschaften von Objekten und werden dazu genutzt, Metadaten zu erfassen oder Assoziationen eines zu beschreibenden Objekts, zu den Begriffen eines Vokabulars bzw. den Konzepten einer Ontologie herzustellen. Beispielweise können Produkte in Online-Shops mit ihren Eigenschaften annotiert werden. Es gibt viele verschiedene Arten von Annotationen. In den Lebenswissenschaften, wie zum Beispiel in der Gene Ontology (GO) [4], werden Annotationen genutzt um Gene bzw. Genprodukte mit ihren Funktionen zu assoziieren. Weiterhin werden sie zur Indexierung von Publikationen, zum Beispiel in PubMed, mit Hilfe von Medical Subject Headings (MeSH) [29] genutzt.

Zur Entwicklung von Ontologien stehen verschiedene Sprachen mit unterschiedlicher Ausdrucksstärke zur Verfügung. Frame-Logic (F-Logic) [23] ist eine objektorientierte Ontologiesprache und zeichnet sich durch ihre große Ausdrucksstärke aus. Allerdings nutzt diese Sprache keine klassische Logik und ist unentscheidbar. Im Semantic Web kommen mehrere Sprachen zum Einsatz [25]. RDF Schema (RDF/S) [3], als Erweiterung vom Resource Description Framework (RDF) [27] zur Definition von Klassen, Beziehungen und Klassenhierarchien, und die Web Ontology Language (OWL) [30], als Erweiterung von RDF/S zur detaillierten Beschreibung von Klassen, z.B. Kardinalitäten und Disjunktheit, sind nur zwei Beispiele für solche Ontologiesprachen. In den Lebenswissenschaften sind vor allem Open Biomedical Ontologies (OBO) [11] im

Einsatz. Mit seinem einfachen, verständlichen Flatfile Format dient OBO zur kollaborativen Entwicklung von biomedizinischen Ontologien, wie z.B. der Gene Ontology.

Um diese und weitere Sprachen zur Entwicklung von Ontologien einsetzen zu können, stehen eine Reihe von Ontologieeditoren zur Verfügung [36]. Für das Erstellen, Bearbeiten und Warten von OBO Ontologien kann zum Beispiel der Open Source Editor OBO-Edit [8] eingesetzt werden. Dieser nutzt einen graphorientierten Ansatz und wird vom Gene Ontology Consortium (COG) [5] entwickelt. Protégé [10] ist aufgrund seiner einfachen Erweiterbarkeit ein sehr beliebter Ontologieeditor und wird in der Community von vielen Anwendern genutzt. Ebenso wie OBO-Edit, ist Protégé in Java [12] geschrieben und dadurch plattformunabhängig. Durch Plugins können nicht nur die Funktionalitäten des Editors erweitert werden, sondern auch die Unterstützung von verschiedenen Ontologiesprachen.

Ontologien müssen über die Zeit angepasst werden, um den Wissensstand möglichst aktuell und korrekt zu halten. Anpassungen können zum Beispiel bei neuem oder geänderten Domänenwissen, zur Behebung von initialen Designfehlern oder bei veränderten Anforderungen seitens der Benutzer nötig werden. Auch bei Änderungen der Richtlinien müssen Ontologien angepasst werden. Des Weiteren kann eine Migration in eine andere Ontologiesprache notwendig sein. Änderungsoperationen können in einfache Änderungen, wie z.B. das Hinzufügen, Löschen oder Ersetzen von Entitäten (Oberbegriff für Konzepte, Attribute und Relationen) und komplexe Änderungen, wie z.B. das Verschieben von Entitäten, das Hinzufügen oder Löschen von ganzen Subgraphen oder das Zusammenfügen von mehreren Konzepten, unterteilt werden.

Durch die Weiterentwicklung von Ontologien können verschiedene Probleme auftreten. In der Forschung können sich Analyseergebnisse auf ältere Versionen beziehen und müssen daher auf ihre Richtigkeit geprüft werden. Mappings müssen überarbeitet werden und es muss dafür gesorgt werden, dass Annotationen ihre Gültigkeit behalten. Aus diesen Problemen gehen mehrere Fragen hervor, die für den Benutzer von Interesse sein können. Beispielsweise könnte es wichtig sein, welche Änderungen zwischen zwei Versionen vorgenommen wurden und welche davon am häufigsten aufgetreten sind. Außerdem könnte es von Nutzen sein, zu wissen, wie sich einzelne Konzepte im Detail verändert haben und welche Ontologieteile stabil sind bzw. oft angepasst werden.

Die Beantwortung dieser und weiterer Fragen ist für Anwender und Entwickler der Ontologie wichtig. Beispielsweise müssen Nutzer Analysedaten validieren, da Forschungsergebnisse voneinander abhängen können. Der Entwickler kann sich über den aktuellen Stand der Entwicklung informieren und herausfinden, wo Anpassungen von Mappings und Annotationen nötig sind. Außerdem kann mit einer Speicherung der Versionsunterschiede die Ontologieverwaltung optimiert werden. Deswegen ist es wichtig, zwei Versionen einer Ontologie miteinander zu vergleichen und somit herausfinden zu können, welche Änderungen zwischen den Ontologieversionen stattgefunden haben.

Unterschiede zwischen zwei Versionen einer Ontologie, kurz als *Diff* bezeichnet, können mit der Hilfe von verschiedenen Algorithmen berechnet werden. Zum Beispiel bietet der PromptDiff-Algorithmus [31] die Möglichkeit einfache Änderungen, beispielsweise *created* und *deleted*, zwischen zwei Ontologieversionen zu bestimmen. Einen semantisch angereicherten Diff liefert der *Complex Ontology Diff* Algorithmus (COnToDiff) [19], der über den *Complex Ontology Diff Explorer* (CODEX) [18] erreichbar ist. COnToDiff stellt insbesondere einen kompakten Diff mit komplexen Änderungsoperationen zur Verfügung, der bereits mit guten Ergebnissen, auch im Vergleich mit PromptDiff, überzeugen konnte [19]. Leider steht COnToDiff bisher nur über ein Webtool (CODEX), jedoch nicht innerhalb eines Ontologieeditors zur Verfügung. COnToDiff ist somit bisher nicht direkt für Ontologieentwickler nutzbar, da diese auf ein Entwicklungswerkzeug angewiesen sind und der Gebrauch eines externen Tools sehr umständlich sein kann. Mit Hilfe von CODEX können Änderungen mit Informationen zu beteiligten Konzepten und Beziehungen zwar betrachtet werden, jedoch fehlt die Möglichkeit, durch die Hierarchie der Ontologie zu navigieren. Mit einem Ontologieeditor wird diese Navigation möglich und an Änderungsoperationen beteiligte Entitäten können direkt betrachtet werden. Da Protégé ein viel genutzter und weit verbreiteter Editor ist, ist es sinnvoll den COnToDiff-Algorithmus als Plugin in Protégé nutzen zu können und ihn somit einer großen Community zur Verfügung zu stellen. Neben der Betrachtung der Änderungsoperationen, bietet die Umsetzung als Plugin auch die Möglichkeit mit dem Editor zu interagieren und somit den Arbeitsablauf in hohem Maße zu verbessern.

1.2 Ziele der Arbeit

Im Rahmen der Arbeit soll ein Plugin für den Ontologieeditor Protégé entwickelt werden. Dieses Plugin soll die Möglichkeit bieten, zwei Versionen einer Ontologie, mit Hilfe des COnToDiff-Algorithmus, miteinander zu vergleichen. Als Ergebnis des Vergleichs soll dem Nutzer eine umfassende Statistik über die vorgefundenen Änderungsoperationen präsentiert und die Möglichkeit geboten werden, durch alle Änderungen zu navigieren. Dabei soll er sich jederzeit Konzepte in den geöffneten Ontologien anzeigen lassen können, die an einer ausgewählten Änderungsoperation beteiligt sind.

Um diese Anforderungen zu realisieren, werden folgende Hauptziele verfolgt:

- Erarbeitung eines Konzepts zur Umsetzung des Protégé-Plugins für COnToDiff. Hierbei werden Anforderungen an das Plugin und die benötigte Datenbank herausgearbeitet und die Auswahl eines geeigneten Plugintyps getroffen.
- Implementierung des COnToDiff-Algorithmus als Plugin für Protégé mit Hilfe des erarbeiteten Konzepts. Zusätzlich soll eine Benutzeroberfläche für das Einlesen

der zu untersuchenden Ontologien, sowie für die Ausgabe der Ergebnisse, erstellt werden.

- Durchführung einer beispielgestützten Evaluierung, um die Anwendbarkeit des Plugins zu zeigen. Außerdem werden die Funktionalitäten des COnToDiff-Plugins mit den Möglichkeiten von Protégé OWL Diff gegenübergestellt.

1.3 Aufbau der Arbeit

Die nachfolgende Arbeit gliedert sich wie folgt:

Kapitel 2 gibt eine Einführung zu Ontologien, wie diese sich verändern und mit welchen Programmen sie bearbeitet werden können. Des Weiteren werden der COnToDiff-Algorithmus und CODEX vorgestellt und die Arbeit thematisch eingeordnet.

Kapitel 3 stellt das Konzept für die Umsetzung des COnToDiff-Plugins vor. Insbesondere werden hierbei die Anforderungen an das Plugin ausgeführt, die verschiedenen Plugintypen von Protégé und der Aufbau der COnToDiff-Umsetzung näher betrachtet, sowie der Entwurf der Benutzeroberfläche vorgestellt.

Kapitel 4 beschreibt welche Techniken zur Umsetzung des erarbeiteten Konzepts genutzt wurden. Dabei wird näher drauf eingegangen, wie ein Plugin für Protégé erstellt wird, welche Anpassungen am COnToDiff-Algorithmus notwendig waren und wie die Benutzeroberfläche umgesetzt wurde.

Kapitel 5 beschäftigt sich mit der Evaluierung des entwickelten Plugins. Hierbei werden Tests mit einer Beispielontologie genutzt, um die Funktionen der Benutzeroberfläche vorzustellen und die gelieferten Ergebnisse des COnToDiff-Plugins denen des Protégé OWL Diffs gegenüberzustellen.

Abschließend folgt eine Zusammenfassung der Bachelorarbeit und ein Ausblick auf mögliche weitere Schritte.

2 Grundlagen & Verwandte Arbeiten

2.1 Ontologien

Ontologien sind in den 90er Jahren des 20. Jahrhunderts im Zusammenhang mit der Wissensrepräsentation populär geworden. Sie lassen sich dem Themenspektrum der künstlichen Intelligenz, aber auch der Datenbanken und allgemein der praktischen Informatik zuordnen. Der Begriff *Ontologie* wird heutzutage in zweierlei Bedeutung verwendet. Zunächst ist die Ontologie eine Disziplin der theoretischen Philosophie, die sich mit der Einteilung des Seienden und den Grundstrukturen der Wirklichkeit und der Möglichkeit beschäftigt.

In der Informatik wird eine Ontologie als formales Wissenssystem aufgefasst. Diese Auffassung entspricht der Definition von T. Gruber, der diesen Begriff in die Informatik und künstliche Intelligenz eingeführt hat. Er definiert eine Ontologie als eine “explizite, formale Spezifikation einer gemeinsamen Konzeptualisierung” (Übersetzung aus [15]). Das bedeutet, dass es sich bei einer Ontologie um ein abstraktes Modell (*Konzeptualisierung*) handelt, bei dem identifizierte und relevante Begriffe, sowie Beziehungen einer Domäne von Interesse maschinenverstehbar (*formal*) festgehalten werden. Dabei sind die Bedeutungen aller Begriffe definiert (*explizit*). Ontologien dienen also zum Erfassen und Darstellen von Wissen eines bestimmten, festgelegten Bereichs, wobei dieser zumeist einen Ausschnitt der Realität modelliert.

Eine solche Ontologie besteht im Allgemeinen aus Konzepten, die durch Relationen untereinander verbunden sind. Zusammen bilden diese einen gerichteten azyklischen Graph (DAG), der die Struktur der Ontologie repräsentiert. Spezielle Konzepte, die Wurzeln genannt werden, sind in der Hierarchie am weitesten oben vorzufinden und haben typischerweise keine Beziehung zu einem übergeordneten Konzept. Jedoch kann mehr als eine Wurzel existieren. In diesem Fall kann eine virtuelle Wurzel definiert werden. Somit kann man dann alle Wurzeln als Kinder der virtuellen Wurzel definieren. Mit der Hilfe von Attributen können Konzepte näher beschrieben werden. Zum Beispiel ist der Zugriffsschlüssel (eng. *accession number*) ein spezielles Attribut, welches ein Ontologiekonzept eindeutig identifiziert. Weitere typische Attribute sind der Name/das Label, eine Definition oder Synonyme für ein Konzept. Die am häufigsten auftretende Relation ist die *is_a*-Beziehung, die normalerweise die Grundstruktur einer Ontologie bildet [22]. Andere Relationen erweitern diese Basisbeziehung, z.B. *part_of*, *regulates* oder *located*. Außerdem können Ontologien unterschiedlich ausdrucksstark sein. Eine der einfachsten Notationen ist eine endliche Liste von Begriffen, wie z.B. ein Katalog. Sehr ausdrucksstarke Ontologien haben logische Restriktionen um z.B. Disjunktheit von Konzepten auszudrücken [26]. In Abbildung 1 sind zwei Versionen einer Beispielontologie zu sehen. Dabei befindet sich die alte Version auf der linken und die neue Version auf der rechten Seite. Das Konzept *brain* stellt in beiden Versionen

die Wurzel dar. Die durchgezogenen Linien bzw. Pfeile sind *is_a*-Relationen zwischen den Konzepten. Auf die Evolution von Ontologien wird im folgenden Kapitel 2.2 näher eingegangen.

Ontologien finden in vielen Bereichen ihre Anwendung. In der Wissenschaft werden sie vor allem dazu genutzt, Wissen bzw. Forschungsergebnisse strukturiert zu erfassen, um somit diese Daten für Untersuchungen nutzbar zu machen. Einfache Formen von Ontologien, z.B. informale *is_a* Hierarchien wie Produktkataloge, werden in Onlineshops und Suchmaschinen genutzt, weil die maschinenverstehbare Darstellung von Informationen nicht nur erlaubt, eine Internetpräsenz dynamisch zu erstellen, sondern auch eine semantische Suche ermöglicht.

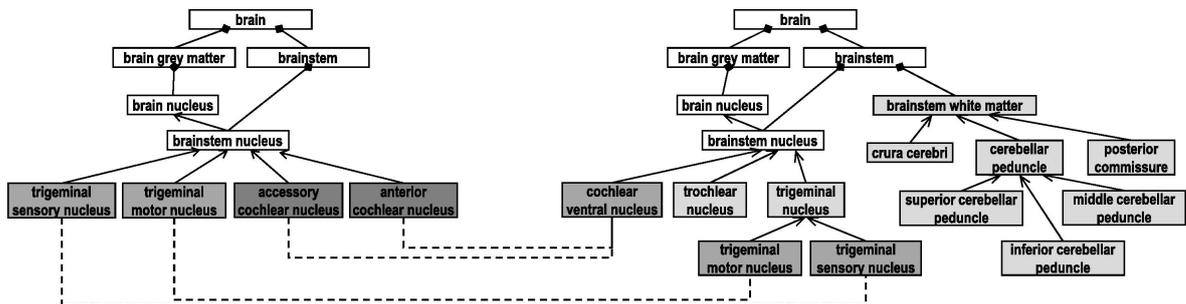


Abbildung 1: Ontologieevolution in Beispielontologie (aus [19])

2.2 Ontologieevolution

Die Ontologieentwicklung ist kein statischer Prozess, sondern ein dynamischer. Zunächst wird eine initiale Version einer Ontologie erstellt, welche später überarbeitet, verbessert und um weitere Details erweitert werden kann. Dieses Vorgehen ist notwendig, weil zum einen ständig neues Wissen, z.B. durch Forschung, hinzukommt und zum anderen eine Reihe von Gründen existieren, die direkt mit der Entwicklung einer Ontologie zusammenhängen.

Ontologien können beim ersten Erstellen noch Designfehler enthalten. Das heißt, dass sie nicht alle Nutzeranforderungen erfüllen und aus diesem Grund so angepasst werden müssen, dass alle Anforderungen, die der Nutzer an die Ontologie stellt, erfüllt werden können. Doch auch wenn alle Nutzeranforderungen zu Beginn der Entwicklung einer Ontologie bedacht und erfüllt werden, können diese sich im Laufe der Zeit ändern. Deswegen müssen auch im fortgeschrittenen Entwicklungszyklus Änderungen an einer Ontologie vorgenommen werden, um die neuen Anforderungen der Nutzer zu erfüllen. Des Weiteren kann man nicht davon ausgehen, dass das festgehaltene Wissen auf dem Stand zur Zeit der Erstellung der Ontologie bleibt. Häufig werden neue Versionen von Ontologien veröffentlicht. Durch neue Forschungsergebnisse in dem Bereich, mit dem sich die jeweilige Ontologie beschäftigt, kann es vorkommen, dass die erhobenen Daten veraltet sind. Aus diesem Grund müssen von Zeit zu Zeit Informationen entfernt

oder neues Wissen hinzugefügt werden, damit sich die Ontologie auf einem möglichst aktuellen Stand befindet [37].

Das Ziel der Ontologieevolution ist es, nach einer Reihe von Änderungen an einer konsistenten Ontologieversion, wieder einen konsistenten Zustand zu erreichen. Es existieren einfache und komplexe Änderungsoperationen (weitere Details in Kapitel 2.3), die es den Ontologieentwicklern ermöglichen, Änderungen an einer Ontologie vorzunehmen um neues oder geändertes Wissen in diese einzupflegen. Eine solche Änderung in einem Teil einer Ontologie kann Inkonsistenzen in einem anderen Teil hervorrufen. Aus diesem Grund kann die Ontologieevolution ein sehr kostenintensiver Prozess sein, der unter Umständen teurer als das Entwickeln einer Ontologie und der dafür benötigten Programme ist [35].

Bei der Entwicklung von Ontologien können nicht nur Entwickler und Nutzer unterschieden werden, denn um das Wissen einer Domäne von Interesse erfassen und strukturieren zu können, sind vor allem Experten gefragt, die zu einem bestimmten Teil einer Ontologie wichtige Aspekte liefern. Für diesen Zweck existieren Konsortien, wie das Gene Ontology Consortium (GOC) [5] oder das Plant Ontology Consortium (POC) [6], die es sich durch verschiedene Zusammenschlüsse zur Aufgabe gemacht haben, domänenspezifisches Wissen zusammenzutragen und es möglichst nutzerfreundlich unterschiedlichen Anwendungsbereichen zu Verfügung zu stellen. Im Normalfall findet die Arbeit an unterschiedlichen Teilen einer Ontologie kollaborativ statt. Diese können sich gegenseitig beeinflussen. Um die Konsistenz einer Ontologie während des Entwicklungszyklus wahren zu können, müssen sich alle Beteiligten jederzeit einen Überblick über die vorgenommenen Änderungen verschaffen können. Eine Möglichkeit dies zu gewährleisten wäre, eine Änderungshistorie zu führen und diese nach jeder Änderungsoperation zu erweitern. Der Nachteil bei diesem Vorgehen ist allerdings, dass nur einfache Änderungsoperationen, wie z.B. *add* und *delete*, geführt werden. Das erschwert bei einer Vielzahl von Änderungen das Verständnis über die Entwicklung der Ontologie. Über mehrere Ontologieversionen hinweg können schrittweise zusammenhängende Änderungen stattfinden, die durch eine kompakte Änderungsoperation, wie zum Beispiel das Hinzufügen eines Subgraphen, ausgedrückt werden können.

Ein anderer Ansatz ist es, zwei Ontologieversionen miteinander zu vergleichen und dem Anwender alle Änderungsoperationen aufzulisten. Diese können dann zu kompakteren Operationen zusammengefasst werden, um sich einen besseren Überblick über die Änderungen verschaffen zu können. Zu diesem Zweck stehen Algorithmen, wie COnto-Diff (siehe Kapitel 2.3) oder Promptdiff zur Verfügung.

Die Notwendigkeit eines Vergleichs zweier Ontologieversionen ist jedoch nicht nur in der aktiven Entwicklung einer Ontologie gegeben. Existieren bereits mehrere Ontologien, die sich mit demselben Wissensbereich beschäftigen, ist es unter Umständen sinnvoll, sich dem Wissen dieser zu bedienen, anstatt eine neue Ontologie zu entwickeln. Hierzu kann ein Mapping zweier oder mehrerer Ontologien genutzt werden, bei dem

durch eine Untersuchung auf Ähnlichkeiten, z.B. bei Konzepten mit gleichen Attributen, eine neue Ontologie berechnet wird, die das Wissen aller einbezogenen Ontologien vereint [21]. Da es sich um ein Mapping zwischen sich stetig entwickelnden Ontologien handelt, muss auch hier von Zeit zu Zeit überprüft werden, ob es noch seine Gültigkeit hat bzw. veraltet ist oder ob Anpassungen vorgenommen werden müssen. Außerdem kann ein Diff in Verbindung mit einem Anpassungsalgorithmus dazu verwendet werden, ein Mapping zwischen Ontologien semi-automatisch zu aktualisieren, indem dieser die notwendigen Anpassungen identifiziert [14].

Neben den Ontologie-Mappings ist es auch bei Annotationen notwendig diese regelmäßig auf ihre Gültigkeit zu überprüfen. Wird beispielweise ein Konzept einer Ontologie gelöscht, so müssen dessen Annotationen verschoben oder ebenso gelöscht werden. Außerdem können Anpassungen an Annotationen nötig werden, um neues Wissen widerzuspiegeln oder Inkonsistenzen zu eliminieren [9].

In Abbildung 1 kann man die Entwicklung einer Ontologie von der einen Version zur nächsten erkennen. Die weißen Konzepte bleiben unverändert, wohingegen bei allen anderen Konzepten verschiedene Änderungen stattfinden. Bei dem Konzept *brainstem* wird beispielsweise ein kompletter Subgraph hinzugefügt und *accessory cochlear nucleus* bzw. *anterior cochlear nucleus* werden in der neuen Version zu *cochlear ventral nucleus* zusammengeführt. Im folgenden Kapitel 2.3 wird unter anderem genauer auf diese und weitere Änderungsoperationen eingegangen.

2.3 COntoDiff

Zur Bestimmung der Änderungen von einer Ontologieversion zur nächsten gibt es eine Reihe von Möglichkeiten und Algorithmen. Jedoch unterstützen viele Umsetzungen nur einfache Änderungsoperationen, wie z.B. *add* und *delete*. Je nach Größe der Ontologie sind diese jedoch nicht ausreichend, um aussagekräftige Informationen über die Evolution einer Ontologie zu erhalten. Insbesondere bei großen Ontologien ist ein kompakter Diff unverzichtbar, weil man durch diesen komplexe Operationen entdecken kann, die sich aus einfachen Änderungen zusammensetzen. Damit fällt es um ein Vielfaches leichter, Zusammenhänge bei der Analyse einer Ontologie zu erkennen.

Bei COntoDiff [19] handelt es sich um einen Algorithmus zur Bestimmung eines ausdrucksstarken und invertierbaren *diff evolution mappings* aus zwei Versionen derselben Ontologie. Bei einem Evolutionsmapping handelt es sich um eine Menge von Änderungsoperationen, die nötig sind um eine Ontologieversion in die andere zu überführen. Der Ablauf der Berechnung ist in Abbildung 2 dargestellt. Zunächst werden zwei Versionen derselben Ontologie, hier mit O_{old} und O_{new} bezeichnet, eingelesen. Optional kann noch weiteres Hintergrundwissen, wie ein Wörterbuch, hinzugefügt werden, um das Ergebnis der *Match*-Phase zu verbessern. In dieser Phase, die z.B. mit Hilfe der Matchkomponente von GOMMA [24] ausgeführt werden kann, werden zusammengehö-

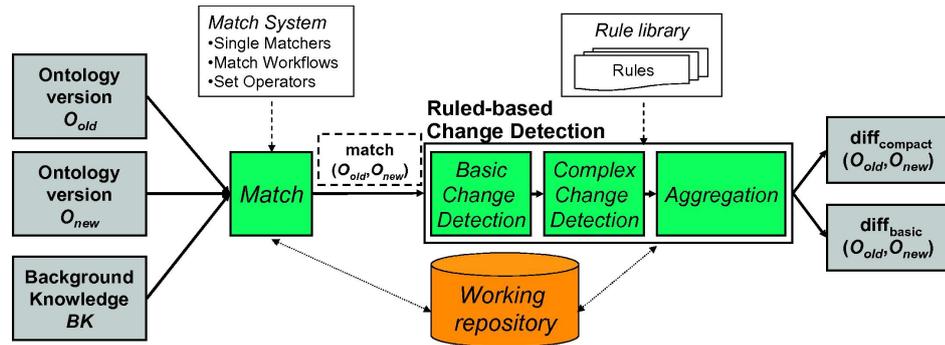


Abbildung 2: COntoDiff im schematischen Überblick (aus [19])

rende Konzepte aus beiden Ontologieversionen ermittelt und daraus ein *match mapping* erstellt. In Abbildung 1 stimmen die weißen und die durch eine gestrichelte Linie verbundenen Konzepte überein und werden deswegen jeweils als Match gekennzeichnet. Die folgenden Schritte nutzen dieses Mapping und laufen automatisch ab.

Im Anschluss an die Matchphase werden so genannte *Change Operation Generating (COG) Rules* angewendet, um zunächst den $diff_{basic}$ zu bestimmen. Hierbei werden Basis-Änderungsoperationen bestimmt, die sich auf ein einzelnes Konzept, ein Attribut oder eine Relation beziehen. Dazu gehören zum Beispiel *add* und *delete*. Im Anschluss werden die Regeln zur Berechnung des $diff_{compact}$ mit Hilfe des $diff_{basic}$ ausgeführt. Dabei werden die einfachen Änderungsoperationen zu einer kleineren Menge kompakter Operationen zusammengefasst. Damit der $diff_{compact}$ so kompakt wie möglich ist, werden nach der *Complex Change Detection* noch Aggregationsregeln angewendet, die rekursiv ausgeführt werden können. Zu den Änderungsoperationen des $diff_{compact}$ gehört z.B. das Ersetzen eines Konzepts durch ein anderes (*substitute*), das Zusammenführen mehrerer Konzepte (*merge*) oder das Einfügen bzw. Löschen eines ganzen Subgraphen (*addSubgraph/delSubgraph*). Hierbei ist noch zu beachten, dass jede Änderung invertierbar ist. Das heißt, dass zum Beispiel ein *split* durch einen *merge* wieder rückgängig gemacht werden kann, wenn das inverse Evolutionsmapping auf die neue Version O_{new} angewendet wird, um wieder O_{old} zu erzeugen. Eine Auswahl an möglichen Änderungsoperationen in COntoDiff ist in Tabelle 1 dargestellt. Das Ergebnis des COntoDiff-Algorithmus ist ein $diff_{basic}$, mit Basis-Änderungsoperationen, und ein semantisch angereicherter $diff_{compact}$, der kompakte Änderungsoperationen enthält.

Die in Abbildung 1 dargestellten Konzepte *accessory cochlear nucleus* und *anterior cochlear nucleus* würden von einem Diff mit Basisänderungen wahrscheinlich als gelöscht erkannt werden, obwohl es sich um einen *merge* zu *cochlear ventral nucleus* handelt. Die ursprüngliche Redundanz kann durch unterschiedliche Entwickler hervorgerufen worden sein und wurde durch das Zusammenführen der beiden Konzepte behoben. In so einem Fall könnten die Labels der Konzepte der alten Ontologieversion als Synonyme im neuen Konzept geführt werden und so Hinweise auf den *merge* liefern. Eine andere komplexe Änderungsoperation ist die Hinzufügung eines ganzen

Änderungsoperation	Beschreibung
<i>addC(c), delC(c)</i>	Hinzufügung/Löschung eines Konzepts <i>c</i>
<i>toObsolete(c), revokeObsolete(c)</i>	Setzen/Aufheben des „veraltet-Status“ eines Konzepts <i>c</i>
<i>substitute(c, c')</i>	Ersetzen eines Konzepts <i>c</i> durch ein anderes Konzept <i>c'</i>
<i>split(s, T)</i>	Aufspalten eines Quellkonzepts <i>s</i> in mehrere Zielkonzepte <i>T</i>
<i>merge(S, t)</i>	Zusammenführen mehrerer Quellkonzepte <i>S</i> in ein Zielkonzept <i>t</i>
<i>addSubGraph(c_root, C_Sub)</i>	Einfügen eines Subgraphen mit Wurzel <i>c_root</i> und den Konzepten <i>C_Sub</i>
<i>delSubGraph(c_root, C_Sub)</i>	Löschen eines Subgraphen mit Wurzel <i>c_root</i> und den Konzepten <i>C_Sub</i>
<i>addR(r), delR(r)</i>	Hinzufügung/Löschung einer Beziehung <i>r</i>
<i>move(c, P, P')</i>	Verschiebung eines Konzepts <i>c</i> von seinen Elternkonzepten <i>P</i> zu anderen Elternkonzepten <i>P'</i>
<i>addA(a), delA(a)</i>	Hinzufügung/Löschung eines Attributs <i>a</i>
<i>chgAttValue(c, att, v1, v2)</i>	Ändern des Attributwertes <i>v1</i> von Attribut <i>att</i> des Konzepts <i>c</i> zu Wert <i>v2</i>

Tabelle 1: Auswahl der COnToDiff-Änderungsoperationen (aus [13])

Subgraphen (*addSubgraph*) mit dem Konzept *brainstem white matter* als Wurzel. Vor allen in Ontologien der Lebenswissenschaften ist es nützlich, solche Änderungen an größeren Teilen einer Ontologie festzustellen. Weitere komplexe Änderungen beinhalten das Verschieben (*move*) der Konzepte *trigeminal sensory nucleus* und *trigeminal motor nucleus* von *brainstem nucleus* zum neuen, inneren Konzept *trigeminal nucleus* [19].

Für die Definition der Regeln nutzt die COnToDiff-Implementierung ein XML-basiertes Format, sowie eine MySQL-Datenbank (*working repository* in Abbildung 2) zur Anwendung dieser Regeln und Berechnung der Änderungsoperationen. Außerdem dient sie der Verwaltung der Operationen, dem Abspeichern von Zwischenergebnissen und der Aggregation zu komplexen Änderungsoperationen.

CODEX

Mit Hilfe des Complex Ontology Diff Explorers (CODEX) [18] kann der COnToDiff-Algorithmus über eine Weboberfläche genutzt werden. CODEX bietet eine Reihe von Möglichkeiten zum Import von Ontologien, wie zum Beispiel über das Einfügen von Text in dafür vorgesehene Textfelder, per Dateiimport vom Rechner oder über eine URL. Außerdem unterstützt die Implementierung das Laden von Ontologien im OBO- oder im OWL-Format. Nach dem Import der Ontologieversionen werden die Berechnungen des Algorithmus in einer Datenbank auf einem Webserver ausgeführt und anschließend die Ergebnisse in der Weboberfläche präsentiert. Bei der Ausgabe kann der Benutzer wählen, wie die erhobenen Daten präsentiert werden sollen. Auf der einen Seite kann er sich eine ausführliche Statistik zur Entwicklung der Ontologie von einer zur nächsten Version anzeigen lassen, die durch Kreisdiagramme visualisiert wird. Auf der anderen Seite besteht die Möglichkeit, durch die Änderungsoperationen zu navigieren und sich dabei anzeigen zu lassen, aus welchen Basis-Operationen sich kompakte Änderungen zusammensetzen. Außerdem kann der Benutzer nach bestimmten Entitäten filtern, durch die er navigieren möchte und hat die Möglichkeit sich eine TagCloud anzeigen zu lassen, die veranschaulicht, welche Änderungen am häufigsten aufgetre-

ten und welche Entitäten daran beteiligt sind. Des Weiteren können die Ergebnisse in Textform exportiert werden, um sie zur späteren Verwendung abspeichern zu können.

Leider bietet CODEX, als externes Tool, keine Möglichkeit den COnToDiff-Algorithmus in einer Entwicklungsumgebung, wie einem Ontologieeditor, zu nutzen und bietet selbst keine Darstellung der Ontologien. Dadurch fehlt die Möglichkeit, innerhalb einer geöffneten Ontologie, direkt zu den an einer Änderung beteiligten Entitäten zu springen oder nur Änderungsoperationen für Entitäten anzeigen zu lassen, die man aus dieser Ontologie auswählt. Außerdem ist der Nutzer auf eine Verbindung zu einer entfernten Datenbank angewiesen, um COnToDiff nutzen zu können und eine Integration in einen Ontologieeditor, wie Protégé, ist nicht ohne weiteres möglich.

2.4 Ontologieeditoren

Ontologien bieten, mit ihrem vereinheitlichten Vokabular, die Möglichkeit Wissen über eine Domäne von Interesse festzuhalten, sodass die abgespeicherten Informationen wiederverwendet und zwischen Anwendungen und verschiedenen Nutzergruppen geteilt werden können. Auch wenn eine Reihe von Untersuchungen und Ansätzen zum automatischen Erstellen von Ontologien [32], zum Beispiel aus Datenbanken [20], existieren, werden die meisten Ontologien manuell erstellt. Das geschieht mit der Hilfe von Editoren, die genau für diesen Zweck entwickelt wurden. Mit ihnen hat der Ontologieentwickler bzw. -nutzer die Möglichkeit, Ontologien zu prüfen, diese zu durchsuchen, zu chiffrieren und abzuändern, womit sie den Entwicklungs- und Wartungsaufwand reduzieren [37].

Um die Ontologieevolution in all ihren Schritten zu unterstützen, müssen Ontologieeditoren einige Anforderungen erfüllen [36]. Dazu gehört u.a., dass alle Änderungen, die bei der Entwicklung einer Ontologie auftreten können, vom Editor unterstützt und auch wieder rückgängig gemacht werden können. Für ein effizientes Arbeiten muss der Benutzer sich z.B. leicht zurechtfinden können und durch eine intuitive Bedienung in allen Phasen der Ontologieentwicklung unterstützt werden.

In den Lebenswissenschaften ist OBO-Edit [8] ein wichtiger open-source Ontologieeditor. Entwickelt vom Gene Ontology Consortium, findet er vor allem Anwendung in der Entwicklung einer Vielzahl von Open Biomedical Ontologies, wie z.B. der Cell Ontology [1].

Mit über 240000 registrierten Benutzern (Stand Mai 2014) und einer aktiven Weiterentwicklung stellt die Stanford University Protégé einer großen Community zur Verfügung. Der open-source Ontologieeditor zeichnet sich durch eine individuell anpassbare Benutzeroberfläche aus und kann beliebig mit Plugins erweitert werden. Außerdem findet er, im Gegensatz zu OBO-Edit, Anwendung bei der Entwicklung von Ontologien aller Domänen.

Diff-Funktionalität von Ontologieeditoren

Neben den aufgeführten Anforderungen, ist es wichtig, verschiedene Ontologieversionen miteinander vergleichen zu können, um z.B. herauszufinden, wie sich eine Ontologie über die Zeit entwickelt hat.

OBO-Edit bietet über den *Ontology Change Tracker* die Möglichkeit, sich einfache Änderungsoperationen zwischen zwei Ontologieversionen, wie *added/deleted*, *created/destroyed* oder *changed* in Textform anzeigen zu lassen. Außerdem kann man diese Änderungen exportieren und als Historie einer Ontologie laden. Mit dieser Historie kann man dann durch die Änderungen navigieren und zu den Entitäten in der geöffneten Ontologie springen, wenn diese noch vorhanden sind. Allerdings gibt es weder eine Statistik zu der Entwicklung der Ontologie, noch eine Möglichkeit die angezeigten Änderungsoperationen zu durchsuchen oder auf bestimmte Entitäten einzugrenzen. Des Weiteren bietet OBO-Edit nicht die Möglichkeit, zwei Ontologien mit einer Instanz des Programms zu öffnen. Somit ist es nicht möglich, über die Historie der neueren Version sich z.B. gelöschte Konzepte oder Beziehungen in der alten Ontologieversion durch einen Klick anzeigen zu lassen.

In Protégé steht seit der Version 4 das Tool *Protégé OWL Diff* zur Verfügung, welches auf dem Promptdiff-Algorithmus [31] basiert. Zuvor war der Vergleich von zwei Ontologieversionen nur mit Hilfe des Promptdiff-Plugins¹ möglich, welches zusätzliche Funktionen, wie das Zusammenführen von Ontologien oder das Extrahieren von Teilen einer Ontologie, bot. Außerdem hatte es sowohl eine Tabellenansicht der Änderungsoperationen, als auch eine tabellarische Ansicht zu bieten. Leider ist dieses Plugin nur mit Protégé 3 kompatibel und funktioniert nicht mit neueren Versionen des Editors. Um die Unterschiede zwischen zwei Ontologieversionen mit Protégé OWL Diff bestimmen zu können, muss die neuere Version der Ontologie zunächst in den Editor geladen werden. Anschließend führt man Protégé OWL Diff aus und wählt die ältere Version der geladenen Ontologie zur Diffberechnung. Präsentiert werden die Ergebnisse in tabellarischer Form und am unteren Rand des Ausgabefensters wird eine Statistik der aufgetretenen Änderungsoperationen angezeigt. Zusätzlich können die Ergebnisse nach bestimmten Entitäten durchsucht und die Auswahl der Änderungen so mit den in Protégé geöffneten Ontologien synchronisiert werden, dass beim Auswählen beteiligte Entitäten in der Klassenhierarchie der jeweiligen Ontologie selektiert werden. Letzteres ist sowohl in der alten, als auch in der neuen Ontologieversion möglich, insofern die alte Version beim Laden in einer neuen Arbeitsumgebung geöffnet wurde.

2.5 Abgrenzung der eigenen Arbeit

Der COntoDiff-Algorithmus bietet mit seinem kompakten Diff eine wertvolle Methode um Änderungen zwischen zwei Ontologieversionen zu bestimmen. Leider ist dieser

¹<http://protegewiki.stanford.edu/wiki/PROMPT>

bisher nur über die externe Webanwendung CODEX und nicht innerhalb eines Ontologieeditors nutzbar. CODEX bietet zwar viele nützliche Funktionen zum Import der Ontologieversionen und zur Ausgabe der Ergebnisse, stellt jedoch keine Darstellung der Ontologien und somit auch keine Möglichkeit der Interaktion mit diesen zur Verfügung. In dieser Arbeit soll COnToDiff in einem Plugin für den Ontologieeditor Protégé umgesetzt werden, um dem Anwender die Möglichkeit zu bieten, mit den geöffneten Ontologien zu interagieren. Dadurch können Entwickler von Ontologien den Algorithmus direkt in ihren Arbeitsablauf integrieren, ohne auf ein externes Werkzeug angewiesen zu sein. Durch die große Anzahl an registrierten Nutzern und einer aktiven Community bietet sich an, den Algorithmus als Plugin für Protégé umzusetzen. Nicht nur der Aspekt, dass dieser stetig weiterentwickelt wird, spricht hierbei für den Editor, sondern auch, dass Ontologien aller Domänen mit Protégé entwickelt werden. Auch wenn es bereits die Möglichkeit gibt, Ontologien in Protégé mit Hilfe von Protégé OWL Diff miteinander zu vergleichen, bietet dieses Werkzeug nur einen Diff mit Basisänderungsoperationen, wie *created* und *deleted*. Außerdem fehlt Protégé OWL Diff eine ausführliche und anschauliche Statistik, sowie die Möglichkeit die Analyseergebnisse zu exportieren. Des Weiteren schneidet der COnToDiff-Algorithmus im Vergleich mit dem Promptdiff-Algorithmus, auf dem Protégé OWL Diff basiert, sehr gut ab [19] und würde integriert in Protégé, die Community mit seinem kompakten Diff bereichern. Zusätzlich kann dadurch der Algorithmus mehr Benutzern bekannt gemacht werden (siehe Kapitel 4.5).

3 Konzept zur Entwicklung eines Protégé-Plugins für COntoDiff

Dieses Kapitel widmet sich dem Konzept für die Umsetzung des COntoDiff-Plugins in Protégé. Insbesondere werden die Anforderungen an das Plugin ausgeführt und die verschiedenen zur Verfügung stehenden Plugintypen von Protégé vorgestellt und eingeschätzt. Weiterhin wird der Aufbau der Erweiterung und der Entwurf der Benutzeroberfläche vorgestellt.

3.1 Anforderungen

Mit Hilfe des Complex Ontology Diff Explorers (CODEX) [18] steht der COntoDiff-Algorithmus dem Anwender über eine Weboberfläche zur Verfügung. Obwohl dem Anwender damit umfangreiche Analysefunktionen geboten werden, fehlen CODEX wichtige Funktionen, wie die Interaktion mit den Ontologieversionen. CODEX bietet keine Darstellung der Hierarchie einer Ontologie und der Benutzer ist aufgrund der Umsetzung mit einer entfernten Datenbank auf eine Internetanbindung angewiesen. Aufgrund dieser und weiterer Probleme können folgende Anforderungen für die Umsetzung des COntoDiff-Algorithmus innerhalb eines Protégé-Plugins ermittelt werden:

- Internetunabhängige Umsetzung
- Geringfügige Änderungen am COntoDiff-Algorithmus
- Flexibler Import von Ontologieversionen
- Unterstützung umfangreicher Diff- und Analyse-Funktionalitäten
- Interaktion des Plugins mit den in Protégé geöffneten Ontologien
- Export der Diff- und Analyseergebnisse

3.1.1 Internetunabhängige Umsetzung

Um den COntoDiff-Algorithmus zu nutzen, ist der Anwender von CODEX auf eine Internetanbindung angewiesen, da es sich hierbei um eine Webanwendung handelt und eine Verbindung zu einem Datenbankserver als Teil der internen Berechnungen des Algorithmus notwendig ist. Da es sich bei Protégé auch um eine Software handelt, die offline ausgeführt werden kann und dem Nutzer bei der Anwendung des angestrebten Plugins keine Einschränkungen, wie zum Beispiel die Notwendigkeit einer bestehenden Verbindung zu einer entfernten Datenbank, auferlegt werden sollen, soll die Berechnung des Diffs innerhalb vom Ontologieeditor auch vollkommen losgelöst von einer Internetanbindung möglich sein.

3.1.2 Geringfügige Änderungen am COnToDiff-Algorithmus

COnToDiff nutzt zur Anwendung der COG-Regeln (siehe Kapitel 2.3) und Berechnung der Änderungsoperationen eine MySQL-Datenbank und ist damit nicht lokal unabhängig. Um die lokale Ausführung des Algorithmus gewährleisten zu können, muss eine geeignete Datenbanklösung genutzt werden. Eine Umsetzung mit einer Datenbank auf einem externen Server soll aufgrund der zuvor erwähnten Einschränkungen des Nutzers, vermieden werden. Alternativ besteht die Möglichkeit der Nutzung einer lokalen Datenbanklösung. Auf der einen Seite könnte man davon ausgehen, dass der Nutzer eine MySQL-Datenbank auf seinem Rechner installiert hat oder ihn dazu anleiten diese zu installieren. Jedoch ist damit unnötiger Aufwand für den Endnutzer verbunden, der die Flexibilität des Plugins stark einschränkt. Aus diesem Grund ist eine Umsetzung mit einer eingebetteten Datenbank, die direkt in das Plugin integriert wird, sinnvoll. Dadurch besteht keine Notwendigkeit einer Installation von zusätzlicher Software. Allerdings besteht, je nach genutzter Datenbanklösung, die Notwendigkeit Anpassungen an den Queries des COnToDiff-Algorithmus vorzunehmen, da MySQL viele Funktionen zur Verfügung stellt, die über den SQL-Standard 92 [7] hinaus gehen.

3.1.3 Flexibler Import von Ontologieversionen

Durch den direkten Zugriff auf den Ontologieeditor Protégé und dessen Funktionalitäten, ergeben sich Möglichkeiten, die CODEX, als externes WebTool, nicht zur Verfügung stellen kann. Zu diesen Möglichkeiten zählen zum Beispiel das Öffnen von Ontologien oder einer neuen Arbeitsumgebung, das Laden einer bereits geöffneten Ontologie und der direkte Zugriff auf diese. Im einfachsten Fall werden eine oder beide Versionen einer Ontologie nicht in Protégé geöffnet, sondern werden mit Hilfe eines Dateidialogs des Editors ausgewählt, mit dessen Routinen geladen und nur die Ergebnisse der Auswertung ausgegeben. Das kann sinnvoll sein, wenn der Nutzer beispielsweise nur eine der beiden Ontologieversionen näher betrachten oder gar bearbeiten möchte. Des Weiteren soll es möglich sein, dass eine Ontologie aus der aktuellen Arbeitsumgebung von Protégé geladen werden kann. Das bringt den Vorteil, dass die Ontologie bereits vorher vom Editor geladen worden ist und dies nicht erneut vom COnToDiff-Plugin initiiert werden muss. Weiterhin soll die Möglichkeit bestehen eine oder beide Ontologieversionen in einer separaten Arbeitsumgebung zu öffnen, um diese zum Beispiel direkt gegenüberstellen zu können.

3.1.4 Unterstützung umfangreicher Diff- und Analyse-Funktionalitäten

Das COnToDiff-Plugin soll Funktionen ähnlich dem WebDiffTool CODEX bieten. Dazu gehört eine umfassende Statistik, die z.B. einen Vergleich der enthaltenen Konzepte, Attribute und Relationen der beiden untersuchten Ontologieversionen liefert und die vorgekommenen Änderungsoperationen mit Kreisdiagrammen gegenüberstellt. Der

kompakte Diff, der alle Änderungen so weit wie möglich zusammenfasst, und der Basisdiff werden hierbei separat dargestellt und miteinander verglichen. Des Weiteren soll es möglich sein, ausgehend von den kompakten Änderungsoperationen, durch alle Änderungen zu navigieren und sich somit nicht nur die zugehörigen Entitäten anzeigen zu lassen, sondern auch aufgeschlüsselt zu bekommen, aus welchen einfachen Operationen sich eine Mehrwertige zusammensetzt. Hinzu kommt die Möglichkeit, die Ausgabe der Ergebnisse auf bestimmte Entitäten zu beschränken und sich nur Veränderungen anzeigen zu lassen, in welche die angegebenen Konzepte involviert sind.

3.1.5 Interaktion des Plugins mit den in Protégé geöffneten Ontologien

Sind die gegenübergestellten Ontologieversionen ausgewertet und die Darstellung der Ergebnisse erfolgt, so soll eine direkte Interaktion zwischen den erhobenen Daten und den geöffneten Ontologien möglich sein. Es soll durch die Auswahl einer Änderungsoperation (bei einwertigen Operationen) bzw. einer beteiligten Entität (bei mehrwertigen Operationen) möglich sein, direkt zum betroffenen Konzept zu springen, um dieses in der Klassenhierarchie betrachten zu können. Diese Funktion soll nach Möglichkeit auch in die andere Richtung funktionieren. Durch das Auswählen einer oder mehrerer Entitäten im Baum einer der beiden Ontologien, sollen diese automatisch in eine Liste der Konzepte aufgenommen werden, nach denen gefiltert werden soll. Anschließend soll der Anwender sich nur Änderungen anzeigen lassen können, an denen diese Konzepte beteiligt sind.

3.1.6 Export der Diff- und Analyseergebnisse

Um nach der Diffberechnung weiter mit den erhobenen Daten arbeiten zu können, soll dem Nutzer die Möglichkeit geboten werden, die Ergebnisse zu exportieren. Dieser Export umfasst eine tabellarische Statistik über die vorgefundenen einfachen und komplexen Änderungsoperationen, sowie eine detaillierte Auflistung aller gefundenen Änderungen, wie sich diese zusammensetzen und welche Entitäten daran beteiligt sind. Der Benutzer soll die Ergebnisse kopieren oder in einer Datei abspeichern können.

3.2 Auswahl eines geeigneten Plugintyps

Für den Ontologieeditor Protégé stehen eine Reihe von Plugintypen zur Verfügung. Je nach Auswahl können verschiedene Teile der Benutzeroberfläche erweitert, angepasst oder für neue Funktionen genutzt werden. Nachfolgend wird ein kurzer Überblick über die verschiedenen Typen gegeben und diskutiert, in wie weit der jeweilige Plugintyp für die Implementierung des COntoDiff-Plugins in Protégé geeignet ist.

Um bereits einen Großteil der in Frage kommenden Plugins eingrenzen zu können, müssen diese erst einmal grob in zwei Kategorien eingeteilt werden. Auf der einen Seite

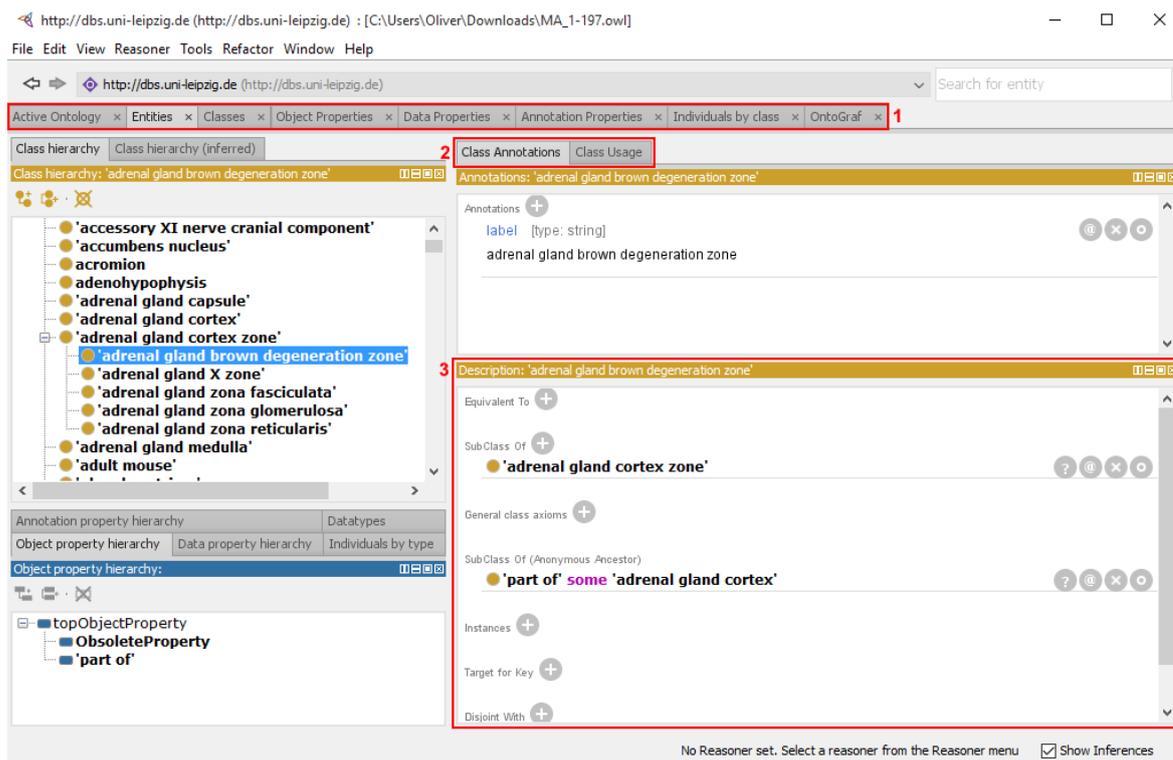


Abbildung 3: Bestandteile der Benutzeroberfläche von Protégé
(1 – Workspace Tabs, 2 – View Tabs, 3 – View)

gibt es die *OWL Plugins*. Dabei handelt es sich um Erweiterungen für Protégé, die über die Kernfunktionalitäten des Ontologieeditors hinausgehen. Sie können dazu genutzt werden, Reasoner zu implementieren. Dabei handelt es sich um eine Software, mit der es aufgrund von Axiomen oder Behauptungen möglich ist, logische Schlussfolgerungen zu ziehen. Des Weiteren stehen in dieser Kategorie Plugintypen zur Verfügung, die es beispielsweise ermöglichen, Änderungen am Erscheinungsbild von Entitäten in der Benutzeroberfläche vorzunehmen. Allerdings handelt es sich beim COnToDiff-Plugin um ein zusätzliches Werkzeug, welches nicht an die OWL Funktionalitäten von Protégé gebunden sein soll. An der vorhandenen Visualisierung von Ontologien soll nichts verändert werden und das Plugin soll eine eigene Benutzeroberfläche zur Verfügung stellen, die speziell für die Repräsentation der Diff-Ergebnisse geeignet ist.

Als zweite Kategorie von Plugins können die *Core Plugins* benannt werden. Diese können genutzt werden um die Grundfunktionen des Ontologieeditors Protégé für das eigene Plugin zu nutzen, vorhandene Darstellungen von Kernbestandteilen anzupassen oder den Editor um grundlegende Funktionen zu erweitern.

Im Folgenden werden einige Core Plugintypen genauer betrachtet und es wird darauf eingegangen, in wie weit sie sich für die Umsetzung des COnToDiff-Algorithmus als Protégé-Plugin eignen.

WorkspaceTab

Hierbei handelt es sich um die Reiter, die zu jeder Zeit in der Arbeitsumgebung sichtbar sind und Grundansichten zur Verfügung stellen. In Abbildung 3 – 1 sind einige dieser Reiter zu sehen. Mit Hilfe eines solchen Plugins ist es möglich, weitere Grundfunktionen zur Verfügung zu stellen, die für den Benutzer jederzeit sichtbar und erreichbar sind. Ein WorkspaceTab kann aus Views (siehe Abschnitt ViewComponent), sowie aus weiteren Tabs mit Views bestehen.

Dieser Plugintyp ist prinzipiell geeignet, da man durch einen WorkspaceTab eine Vielzahl von View-Komponenten nutzen kann. Durch die Kombination mit weiteren Tabs stehen umfangreiche Möglichkeiten zur Gestaltung eines Plugins zur Verfügung. Allerdings wäre das COnToDiff-Plugin mit einem Reiter in einer Arbeitsumgebung an diese gebunden und die Ergebnisse könnten nicht im Workspace der anderen Ontologieversion eingesehen werden.

ViewComponent

Hierbei handelt es sich um den häufigsten genutzten Plugintyp, der zur Implementierung von *Views* (Abb. 3 – 3) genutzt werden kann. Views sind die Bestandteile der WorkspaceTabs (siehe Abschnitt WorkspaceTab) und können jederzeit zu einem dieser Tabs oder zu Tabs von Views (Abb. 3 – 2) hinzugefügt werden

Des Weiteren bieten sie mit Hilfe der Schaltflächen in der oberen rechten Ecke weitere Funktionalitäten. Einerseits können Views auf Wunsch dupliziert werden. Hierbei kann man sich zwischen zwei Möglichkeiten der Anordnung des neuen Views in Bezug auf den alten View entscheiden: Entweder werden die beiden Sichten untereinander oder nebeneinander angeordnet und nehmen hierbei keinen zusätzlichen Platz in der Arbeitsumgebung ein. Der ursprüngliche View wird also in der Höhe oder Breite halbiert. Zusätzlich kann ein View auch als frei bewegliches Objekt von der aktuellen Arbeitsumgebung entkoppelt werden, um es zum Beispiel bei der Auswahl eines anderen WorkspaceTabs weiterhin sehen zu können. Eine weitere Funktion ist das Schließen eines Views. Hierbei wird wieder mehr Platz für andere Benutzerumgebungselemente freigegeben.

Views bieten viele Möglichkeiten Daten übersichtlich darzustellen, allerdings ist ein einzelner View für die Vielzahl an angestrebten Funktionalitäten des COnToDiff-Plugins nicht ausreichend und kann deswegen ohne weitere Elemente nicht für die Implementierung in Frage kommen.

EditorKitFactory

Mit diesem Plugintyp ist es möglich neue Menüeinträge in der Menüleiste des Ontologieeditors Protégé einzufügen. Durch die Nutzung eines Menüeintrags ist es möglich, dem Nutzer das COnToDiff-Plugin von einer zentralen Stelle aus zur Verfügung zu

stellen, da die Menüpunkte jederzeit und von jeder Ansicht aus nutzbar sind. Außerdem besteht hierbei die Möglichkeit ein neues Fenster zu öffnen, welches losgelöst von einer bestimmten Arbeitsumgebung genutzt werden kann. Das hat den Vorteil, dass im Nachhinein weitere Ontologien geöffnet werden und die Ergebnisse eines Vergleichs zwischen zwei Ontologieversionen jederzeit eingesehen werden können, ohne auf andere wichtige Informationen verzichten zu müssen. Somit ist es zum Beispiel möglich, beide Ontologien mit ihren Klassenhierarchien nebeneinander geöffnet zu haben und zusätzlich deren Diff angezeigt zu bekommen. Mit diesem Plugintyp ist es auch möglich einen neuen Menüpunkt der Menüleiste hinzuzufügen. Allerdings existiert bereits ein Menüpunkt namens Tools, der sich sehr gut für die Integration von COntoDiff eignet.

Zusammenfassend kann man sagen, dass Protégé eine Vielzahl an Plugintypen bietet und jedes Teil des Editors in irgendeiner Weise angepasst werden kann. Ob das nun Anpassungen an der Benutzeroberfläche oder im Hintergrund sind. Im Endeffekt kommen für die Umsetzung des COntoDiff-Algorithmus allerdings nur wenige Typen in Frage. Das Hinzufügen eines neuen Menüeintrags in das Tools-Menü bietet die besten Möglichkeiten für die Umsetzung, da das COntoDiff-Plugin dadurch zentral erreichbar ist, eine eigene Benutzeroberfläche umgesetzt und ein eigenständiges Fenster für die Ergebnisausgabe zur Verfügung gestellt werden kann.

3.3 Schematischer und struktureller Aufbau

In Abbildung 4 sieht man die Infrastruktur von CODEX und dem COntoDiff-Plugin. Die linke Seite zeigt unterschiedliche Clients, die mit Hilfe der Webapplikation CODEX den COntoDiff-Algorithmus nutzen. Die Applikation ist auf einem Webserver installiert und für die Berechnungen des Algorithmus wird ein gemeinsames Repository verwendet, welches sich ebenfalls auf dem Server befindet. Die rechte Seite zeigt zwei Instanzen von Protégé mit dem COntoDiff-Plugin, die lokal auf einem Rechner ausgeführt werden. Beide Instanzen greifen auf dieselbe Datenbank zu und können parallel und isoliert den COntoDiff-Algorithmus ausführen.

Der Ontologieeditor Protégé ist so konzipiert, dass die Entwicklung von Plugins und das Einbinden dieser Erweiterungen möglichst einfach realisiert werden kann. Hierzu ist Protégé modular aufgebaut. Das bedeutet, dass zunächst ein Grundgerüst des Entwicklungstools mit elementaren Grundfunktionen existiert. Zu diesen Funktionen gehören zum Beispiel die Benutzeroberfläche, Einstellungsdialoge und Repository für die Ontologien. Unterstützt wird der modulare Aufbau von Protégé durch das OSGi Framework² für die Plugin-Infrastruktur. Hierbei handelt es sich um ein komponentenbasiertes Framework, was es ermöglicht, auf einfachem Wege dynamische Softwarelösungen in Java umzusetzen. Dadurch kann Protégé durch mitgelieferte Plugins, die im

²<http://www.osgi.org/Technology/HomePage>

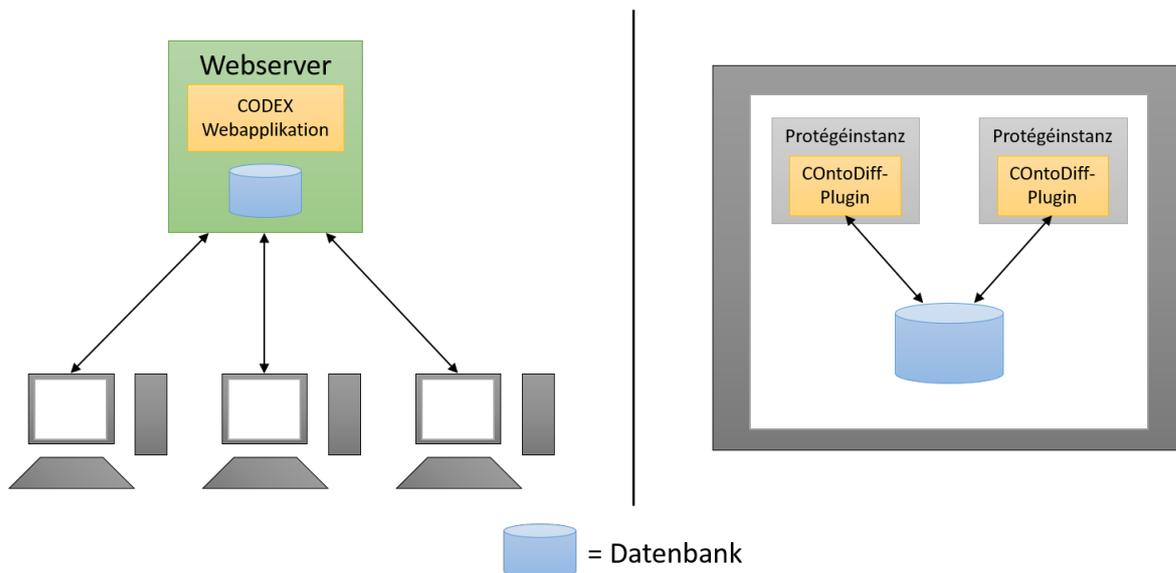


Abbildung 4: Infrastruktur von CODEX (links) und dem COntoDiff-Plugin (rechts)

Zusammenspiel mit den Grundfunktionen das fertige Softwareprodukt ausmachen, um weitere wichtige Funktionalitäten erweitert werden. Darüber hinaus wird den Nutzern, die sich an der Entwicklung von zusätzlichen Anwendungsmöglichkeiten des Editors beteiligen möchten, die Möglichkeit geboten, eigene Erweiterungen zu entwickeln. Diese können anschließend dem Entwicklungsteam von Protégé zur Verfügung gestellt oder mit anderen Nutzern geteilt und ausgetauscht werden (siehe Kapitel 4.5).

Um die internetunabhängige Ausführung des COntoDiff-Algorithmus innerhalb des Plugins zu gewährleisten, ist es nötig, eine eingebettete Datenbanklösung zu nutzen. Dadurch muss der Nutzer nicht selbstständig eine Datenbank auf seinem Rechner installieren und benötigt kein Detailwissen darüber, wie das COntoDiff-Plugin intern funktioniert. Ihm bleibt verborgen, dass eine Datenbank bei den Berechnungen mit einbezogen wird. Im COntoDiff-Plugin wird H2 Database³ als eingebettete Datenbanklösung verwendet (siehe Kapitel 4.3). Bei H2 gibt es verschiedene Möglichkeiten bzw. Verbindungsmodi um eine Verbindung zur Datenbank herzustellen.

Im eingebetteten Modus wird die Verbindung zur Datenbank innerhalb einer Java Virtual Machine (JVM) [28] mit Hilfe von Java Database Connectivity (JDBC) [17] realisiert. Der Nachteil hierbei ist, dass die Datenbank zur gleichen Zeit in nur einer JVM genutzt werden kann. Das bedeutet, dass eine weitere Anwendung nicht auf die Datenbank zugreifen kann.

Im Servermodus wird die Datenbank ferngesteuert und mit Hilfe von JDBC oder ODBC [34], von einer Anwendung geöffnet. Hierfür muss ein Server in derselben oder einer anderen JVM gestartet werden, auf den dann mehrere Anwendungen zur gleichen Zeit zugreifen, und zur selben Datenbank eine Verbindung herstellen können. Das setzt

³<http://h2database.com/>

allerdings entweder einen externen oder lokalen Server voraus, der zusätzlich verwaltet werden muss.

Der letzte Modus ist eine Kombination aus dem eingebetteten und dem Servermodus. Die erste Anwendung stellt eine Verbindung zur Datenbank im eingebetteten Modus her und startet gleichzeitig einen lokalen Server, der es anderen Anwendungen ermöglicht, eine Verbindung zur selben Datenbank herzustellen. Hierbei sind parallele Verbindungen zur Datenbank möglich und es ist keine Verwaltung eines zusätzlichen Servers nötig.

Das COntoDiff-Plugin nutzt die Kombination aus dem eingebetteten und dem Servermodus. Beim ersten Ausführen des Plugins wird die Datenbank, falls sie noch nicht existiert, erstellt und eine Verbindung im eingebetteten Modus zu dieser hergestellt. Zur gleichen Zeit wird ein lokaler Server innerhalb der JVM gestartet. Das ermöglicht anderen Instanzen des Plugins, auf die selbe Datenbank zuzugreifen und somit mehrere Berechnungen des COntoDiff-Algorithmus parallel auszuführen. Die parallele Ausführung wird durch die Nutzung von unterschiedlichen Schemas gewährleistet. Jede Instanz erstellt vor der Berechnung ihr eigenes Schema, mit einem zufällig generierten Präfix (siehe Kapitel 4.3). Die Verbindung zur Datenbank bleibt solange bestehen, bis Protégé beendet wird. Sollte eine weitere Ausführung des Algorithmus erwünscht sein, steht die Datenbankverbindung bereits zur Verfügung, sodass alle nachfolgenden Diffberechnungen wesentlich schneller ablaufen. Nach dem Beenden von Protégé wird die Verbindung zur Datenbank getrennt und der Server heruntergefahren. Eine andere Instanz, die bisher als Client mit Hilfe des Servers mit der Datenbank verbunden war, stellt anschließend automatisch eine Verbindung im eingebetteten Modus her und startet den Server erneut.

3.4 Entwurf der Benutzeroberfläche

Beim Entwurf für die Benutzeroberfläche des COntoDiff-Plugins müssen vor allem die in Kapitel 3.1 erarbeiteten Anforderungen berücksichtigt werden. Hierzu wird im Folgenden der typische Ablauf beim Nutzen des Plugins in Protégé veranschaulicht und auf die einzelnen Funktionen der Oberfläche näher eingegangen.

3.4.1 Starten des COntoDiff-Plugins

Um den COntoDiff-Algorithmus im Ontologieeditor Protégé nutzen zu können, muss dieser über den Menüeintrag Tools unter dem Punkt COntoDiff in der Menüleiste von Protégé gestartet werden. Um was für einen Plugintyp es sich hierbei handelt und warum dieser gewählt wurde, wurde in Kapitel 3.2 näher erläutert. Die Menüleiste befindet sich am oberen Bildschirmrand (siehe Abbildung 3). Es ist dem Nutzer überlassen, ob er vor dem Starten des COntoDiff-Plugins eine Ontologie in Protégé öffnet.

3.4.2 Einlesen der Ontologieversionen

Nach dem Auswählen des COntoDiff-Menüeintrags öffnet sich dem Benutzer ein Dialog zum Importieren zweier Ontologieversionen, die mit Hilfe des COntoDiff-Algorithmus auf Änderungen untersucht werden sollen. Der Entwurf dieses Dialogs ist in Abbildung 5 zu sehen. Das Fenster besteht aus zwei Teilen, jeweils für die ältere und neuere Ontologieversion. Im oberen Abschnitt dieser beiden Elemente befindet sich eine Combobox, mit der man aus den zwei dargestellten Optionen *Import From File* zum Dateiimport und *Import From Current Workspace*, dem Import aus der aktuellen Arbeitsumgebung, wählen kann. Diese Umsetzung hat den Vorteil, dass zu einem späteren Zeitpunkt einfach weitere Möglichkeiten für das Importieren von Ontologieversionen, wie zum Beispiel aus weiteren Dateiformaten (siehe Kapitel ??), hinzugefügt werden können. Entscheidet der Nutzer sich dafür, eine Ontologie aus einer Datei einzulesen, kann er den Dateipfad entweder manuell eingeben oder über die Schaltfläche *Browse* mit Hilfe eines Dateiauswahldialogs von Protégé durch die Ordnerstruktur des Computers navigieren und die gewünschte Datei auswählen. Anschließend hat der Nutzer die Möglichkeit, aus einer Reihe von Optionen zu wählen, wie mit der zu ladenden Ontologie verfahren werden soll. Zur Berechnung des Diffs zwischen den beiden Ontologieversionen ist es nicht nötig, die Ontologien in Protégé zu öffnen. Der Anwender kann sich im einfachsten Fall, mit der Option *Don't Load Ontology*, lediglich die Ergebnisse des Algorithmus anzeigen lassen. Als zweite Option kann dieser *Load Ontology Into Current Workspace* auswählen, womit die Ontologie in der aktuellen Arbeitsumgebung geöffnet werden soll. Hierbei muss jedoch beachtet werden, dass eine bereits geöffnete Ontologie bei diesem Vorgang geschlossen wird. Außerdem werden dem Nutzer nur sinnvolle Optionen zur Auswahl angeboten. Zum Beispiel wird verhindert, dass beide Ontologieversionen im aktuellen Workspace geöffnet werden. Es ist nämlich nur möglich, eine Ontologie pro Workspace zu öffnen. Außerdem soll verhindert werden, dass eine Version in der Arbeitsumgebung geöffnet und die andere aus derselben Umgebung geladen wird, da sie in diesem Fall identisch wären. Das Gleiche gilt für das Laden beider Ontologien aus der aktuellen Arbeitsumgebung. Wie in Abbildung 5 zu erkennen ist, wird

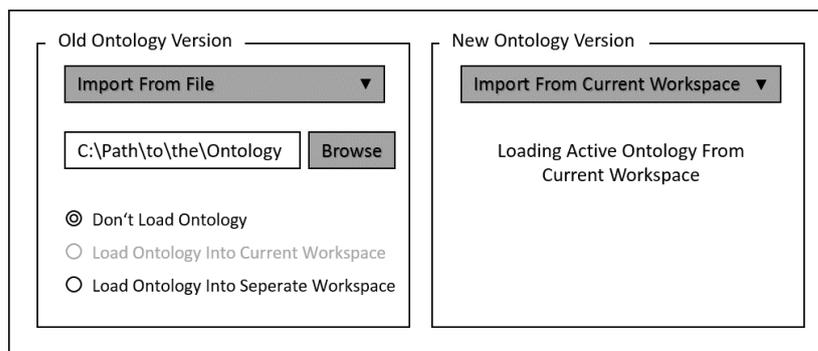


Abbildung 5: Entwurf für den Dialog zum Einlesen der zu vergleichenden Ontologieversionen

die entsprechende Option bei einem Konflikt ausgegraut und ist nicht wählbar. Durch die Option *Load Ontology Into Seperate Workspace*, hat der Anwender die Möglichkeit, die jeweilige Ontologieversion in einer neuen Arbeitsumgebung zu öffnen. Dadurch ist es möglich, beide Versionen der Ontologie nebeneinander anzuordnen und während der Betrachtung des Diffs, durch diese zu navigieren. Der Importdialog bietet somit einen flexiblen und benutzerfreundlichen Import der Ontologieversionen (siehe Anforderung Kapitel 3.1.3).

3.4.3 Fortschritt der Berechnung

Nachdem der Nutzer ausgewählt hat auf welche Art er die Ontologieversionen einlesen möchte und wie mit den eingelesenen Ontologien verfahren werden soll, wird der COntoDiff-Algorithmus ausgeführt. Während der Ausführung des Algorithmus wird eine Fortschrittsanzeige eingeblendet. Der Gesamtfortschritt wird mit Hilfe eines Fortschrittsbalkens visuell dargestellt. Zusätzlich wird der aktuelle Berechnungsschritt des Algorithmus eingeblendet. Dieser Vorgang kann jederzeit abgebrochen werden, indem das Fenster geschlossen wird.

3.4.4 Ausgabe der Ergebnisse

Im Anschluss an die Berechnung des Diffs, zwischen den beiden Ontologieversionen, werden dem Nutzer die Ergebnisse präsentiert. Hierfür wird ein neues Fenster geöffnet (siehe Abbildung 6), deswegen muss die Arbeitsumgebung, in der das Plugin gestartet wurde, nicht geöffnet bleiben. Das bedeutet, dass die Ergebnisse solange einsehbar sind, wie ein Workspace geöffnet ist. Hierbei spielt es keine Rolle, um welchen Workspace es sich handelt. Das Ausgabefenster ist in mehrere Reiter unterteilt, was es dem Nutzer ermöglicht, auf alle Funktionen komfortabel zugreifen zu können. Im Folgenden werden die Inhalte der einzelnen Tabs näher erläutert.

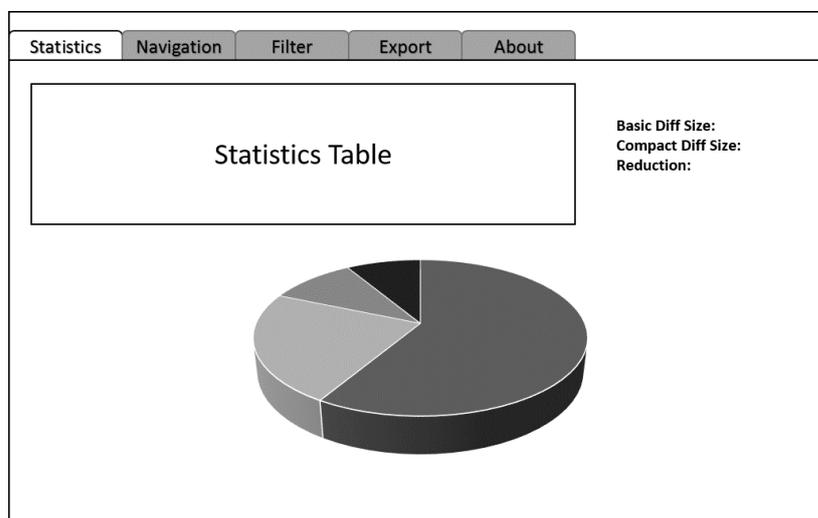


Abbildung 6: Entwurf für die Ausgabe der Statistik

Statistik

Nach der Berechnung der Änderungen zwischen den beiden eingelesenen Ontologieversionen wird dem Nutzer zunächst eine ausführliche Statistik präsentiert (siehe Abbildung 6). Im oberen Abschnitt befindet sich eine Tabelle mit einer Statistik zu den vorgefundenen Konzepten, Relationen und Attributen und wie diese sich von einer Version zur anderen entwickelt haben. Neben der Tabelle wird aufgeführt, wie groß $diff_{basic}$ und $diff_{basic}$ sind (siehe Kapitel 2.3) und daraus berechnet, in wie weit die Größe des Diffs durch die kompakten Änderungsoperationen reduziert werden konnte. Im unteren Abschnitt der Statistik befinden sich zwei Kreisdiagramme, die illustrieren, welche und wie viele Änderungsoperationen jeweils aufgetreten sind. Durch zwei Checkboxen oberhalb der Kreisdiagramme kann die Darstellung der Operationen weiter eingeschränkt werden. So ist es möglich Änderungen an Attributen und/oder Relationen auszuschließen. Dadurch wird dem Anwender eine wichtige Möglichkeit zum Analysieren der Difffergebnisse geboten (siehe Anforderung Kapitel 3.1.4).

Navigation

Mit der Navigation (siehe Abbildung 7) kann der Benutzer des COntoDiff-Plugins durch alle aufgetretenen Änderungen navigieren. Hierfür steht eine Auflistung aller Änderungsoperationen und deren Anzahl auf der linken Seite des Reiters zur Verfügung. Zusätzlich hat er die Möglichkeit, sich im Baum anzeigen zu lassen, aus welchen einfachen Änderungen eine komplexe Operation zusammengesetzt ist. Außerdem wird angezeigt, welche Änderung im Detail stattgefunden hat. Bei der Änderung eines Attributs wird zum Beispiel die ID der betroffenen Entität, das Attribut welches geändert wurde und der alte, sowie der neue Wert angezeigt. Bei der Auswahl einer Änderungsoperation im Baum, wird dem Nutzer auf der rechten Seite eine Tabelle eingeblendet. Diese Tabelle zeigt die Entitäten, die an einer Operation beteiligt sind. Hierzu wird die jeweilige ID der Entität dargestellt und zusätzlich ihr Label dazu ausgegeben. Die Breite dieser Darstellung ist vom Nutzer frei verstellbar und ein Ausblenden ist einfach durch verschieben an den rechten Rand möglich. Durch das Einfügen einer Checkbox am unteren Rand der Navigation wird es möglich, auszuwählen, ob die Auswahl einer Änderung mit den Arbeitsumgebungen synchronisiert werden soll. Das bedeutet, dass die beteiligten Entitäten in den geöffneten Ontologien ausgewählt werden, wenn eine Änderungsoperation im Baum durch einen Klick markiert wird. Da die Auswahl bei einer Operation, bei der mehrere Entitäten involviert sind, nicht eindeutig ist, kann der Nutzer diese mit Hilfe der verschiebbaren Tabelle konkretisieren. Dadurch wird die Interaktion mit den in Protégé geöffneten Ontologien ermöglicht (siehe Anforderungen Kapitel 3.1.4 und 3.1.5).

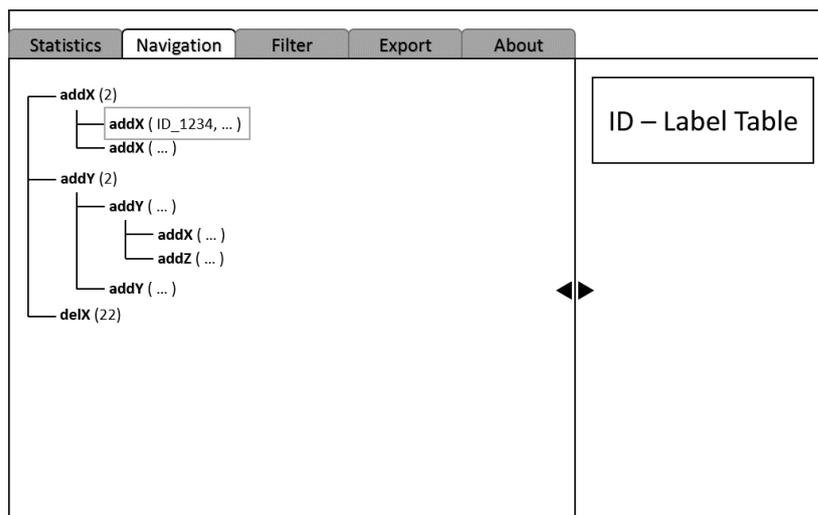


Abbildung 7: Entwurf für die Navigation durch die gefundenen Änderungen

Nach bestimmten Entitäten filtern

Weiterhin können Nutzer die Ergebnisse der Diff-Berechnung nach bestimmten Entitäten filtern. Mit Hilfe des Reiters *Filter* ist es möglich, sich alle Änderungsoperationen ausgeben zu lassen, an denen eine bestimmte Entität beteiligt ist. Durch den Tab für die Navigation kann sich der Benutzer zwar einen guten Überblick über die vorgefundenen Änderungen verschaffen, hat jedoch keine Möglichkeit, ein bestimmtes Konzept zu finden. In Abbildung 8 kann man den Entwurf für die Eingabe der Entitäten sehen, nach denen gefiltert werden soll. In der Abbildung kann man in der Mitte ein Textfeld erkennen. Dort werden die Zugriffsschlüssel der Entitäten untereinander eingetragen. Damit der Nutzer diese nicht manuell eingeben muss, kann er auch die Schaltfläche *Select in Workspace/Navigation* nutzen. Damit öffnet sich ein zusätzliches Fenster, welches ein weiteres Textfeld enthält. In dieses Textfeld wird beim Auswählen einer Entität in einer Arbeitsumgebung oder dem Reiter *Navigation* der entsprechende Zu-

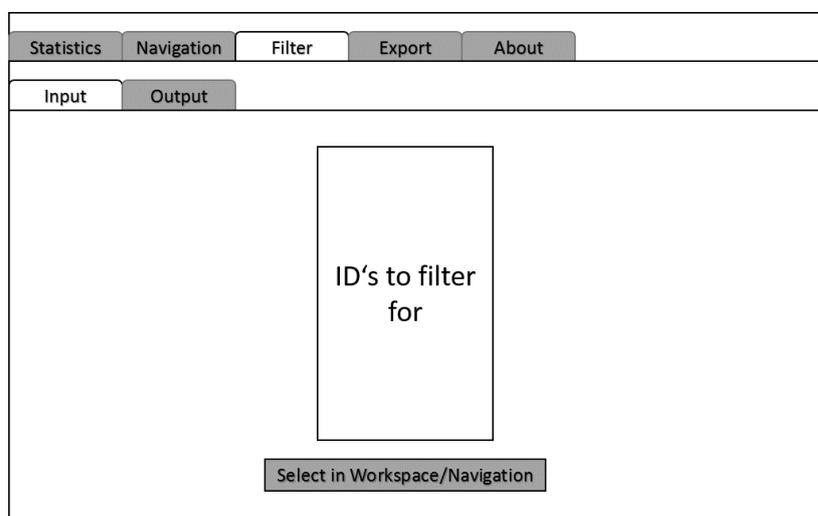


Abbildung 8: Entwurf der Eingabe für das Filtern von bestimmten Entitäten

griffsschlüssel eingetragen. Da sich die beiden Textfelder gegenseitig beeinflussen, wird jede Änderung in einem Feld in das andere übertragen. Der Vorteil des zusätzlichen Fensters ist, dass der Nutzer die Möglichkeit hat, in den Navigationsreiter der Ausgabe des COntoDiff-Plugins zu wechseln, um dort die Entitäten auszuwählen. Dabei hat er jederzeit im Blick, welche Zugriffsschlüssel bereits in der Liste vorhanden sind. Ähnlich funktioniert es, eine Entität für die Auswahl aus einer geöffneten Arbeitsumgebung hinzuzufügen. Durch das Hilfsfenster, was immer im Vordergrund bleibt, kann die COntoDiff-Ausgabe in den Hintergrund verschoben und bequem eine Entität aus einer geöffneten Ontologie gewählt werden. Zusätzlich wird verhindert, dass derselbe Zugriffsschlüssel mehrmals eingetragen werden kann.

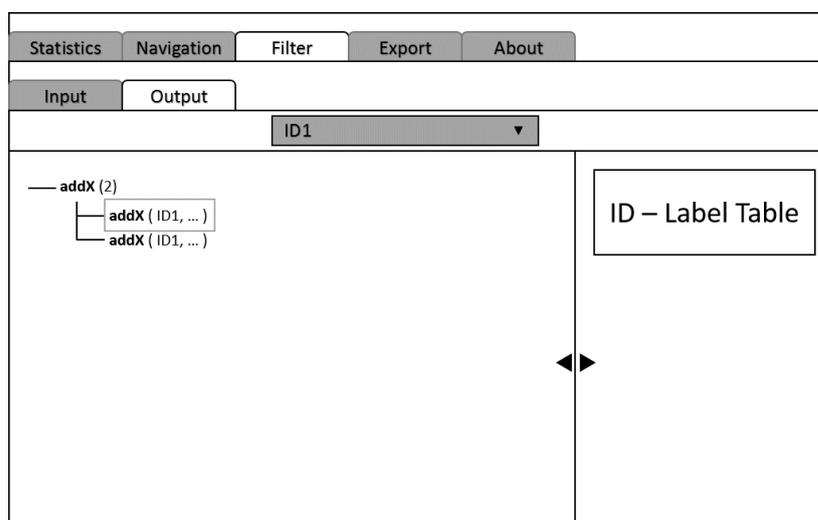


Abbildung 9: Entwurf der Ausgabe für die gefilterten Entitäten

Nachdem der Nutzer die Liste mit den Zugriffsschlüsseln gefüllt hat, kann er im Tab für die Ausgabe betrachten, an welchen Änderungen diese beteiligt sind. Wie in Abbildung 9 zu sehen ist, befindet sich im oberen Teil der Ausgabe eine Combobox. Mit dieser Combobox kann der Nutzer einen Zugriffsschlüssel auswählen und bekommt anschließend alle Änderungsoperationen angezeigt, an denen dieser beteiligt ist. Auch hier ist es, wie in der Navigation (siehe Kapitel 3.4.4), möglich sich die Labels zu den entsprechenden Zugriffsschlüsseln, mit Hilfe der verschiebbaren Tabelle, anzeigen zu lassen. Die Möglichkeit die Auswahl einer Operation mit den Arbeitsumgebungen zu synchronisieren, ist bei dieser Ansicht ebenso gegeben. Damit sind dem Anwender weitere Analysemöglichkeiten der Ontologieversionen und die Interaktion mit den in Protégé geöffneten Ontologien geboten (siehe Anforderungen Kapitel 3.1.4 und 3.1.5).

Exportieren der Ergebnisse

Um auch später wieder auf die Ergebnisse einer Diffberechnung zugreifen zu können, hat der Nutzer die Möglichkeit diese zu kopieren oder in einer Textdatei abzuspeichern. In Abbildung 10 ist der Entwurf für den Reiter *Export* zu sehen. Im oberen Bereich wird

eine Übersicht über alle möglichen Änderungen, sowie ihre Häufigkeit ausgegeben. Außerdem wird, wie in der Statistik (siehe Kapitel 3.4.4), zwischen $diff_{basic}$ und $diff_{compact}$ unterschieden. Im unteren Bereich werden alle komplexen Änderungen mit ihren zugrundeliegenden Basisoperationen aufgeschlüsselt. Über die beiden Schaltflächen *Save to file* kann der Benutzer die Daten über einen Dateidialog abspeichern. Damit hat der Anwender die Möglichkeit die Diff- und Analyseergebnisse zu exportieren (siehe Anforderung Kapitel 3.1.6).



Abbildung 10: Entwurf für den Export der Ergebnisse

Informationen über die Änderungsoperationen

Damit sich der Nutzer einen Überblick über die möglichen Änderungsoperationen verschaffen und sich darüber informieren kann, was diese im Einzelnen bedeuten, existiert ein weiterer Tab namens *About*. Es handelt sich hierbei einfach um eine Seite, die mit Hilfe von HTML [33] mit formatiertem Text befüllt wird. Die Änderungsoperationen werden als Überschriften dargestellt, unter denen der Beschreibungstext zur jeweiligen Operation zu finden ist.

4 Implementierung des COntoDiff-Plugins

Dieses Kapitel beschäftigt sich mit der Umsetzung des COntoDiff-Algorithmus in einem Plugin für Protégé. Hierbei wird näher darauf eingegangen, wie ein Protégé-Plugin erstellt wird und welche Architektur für das COntoDiff-Plugin genutzt wurde. Außerdem werden verschiedene Datenbanklösungen vorgestellt und es wird diskutiert, welche sich für die Implementierung am besten eignet. Des Weiteren wird ein kurzer Überblick über die genutzten Mittel zur Umsetzung der Benutzeroberfläche gegeben. Ferner wird ausgeführt, auf welche Weise das COntoDiff-Plugin veröffentlicht werden kann. Beim letzten Punkt handelt es sich um zukünftige Arbeit.

4.1 Erstellen eines Protégé-Plugins

Zum Erstellen eines Plugins für den Ontologieeditor Protégé wird Java benötigt, welches im besten Fall durch eine geeignete Entwicklungsumgebung, wie Eclipse⁴, ergänzt wird. Um das angestrebte Plugin in Protégé testen zu können, muss man den Editor in Eclipse einbinden. Hierfür gibt es verschiedene Möglichkeiten, die im Wiki⁵ von Protégé ausgeführt werden. Für Version 5 des Ontologieeditors kann der Quellcode mit Hilfe von Git⁶ ausgecheckt und mit Maven⁷ kompiliert werden. Allerdings kann man Protégé auch ohne den Quellcode in Eclipse ausführen, indem man die aktuelle Version bezieht und die enthaltenen Bibliotheken in die Entwicklungsumgebung als Plugins importiert.⁸ Nachdem man Eclipse konfiguriert hat um Protégé ausführen zu können, muss zunächst ein minimales Plugin erstellt werden. Wenn dieses Minimalplugin lauffähig ist und erfolgreich beim Start von Protégé mit eingebunden wurde, können die notwendigen Anpassungen vorgenommen werden um z.B. den gewünschten Plugintyp zu nutzen. Für das COntoDiff-Plugin wurde ein Menüplugin ausgewählt, was mit Hilfe der *EditorKitFactory* umgesetzt werden kann (siehe Kapitel 3.2). Eine ausführliche Anleitung zum Erstellen eines minimalen Plugins⁹ und der Umsetzung eines Menüplugins¹⁰ sind im Wiki von Protégé zu finden.

4.2 Architektur des Plugins

Das Protégé-Plugin für den COntoDiff-Algorithmus basiert auf einer 3-Schichten Architektur (siehe Abbildung 11). Das Backend besteht aus einer eingebetteten Daten-

⁴<https://eclipse.org/>

⁵http://protegewiki.stanford.edu/wiki/Protege4DevDocs#Protege_4_OWL_editor_in_Eclipse_or_IntelliJ

⁶<https://git-scm.com/>

⁷<https://maven.apache.org/>

⁸http://protegewiki.stanford.edu/wiki/CompileProtege5InEclipseWithMaven#Configure_Eclipse_without_Protege_sources

⁹<http://protegewiki.stanford.edu/wiki/PluginAnatomy>

¹⁰http://protegewiki.stanford.edu/wiki/PluginAnatomy#Adding_Menu_Plugins

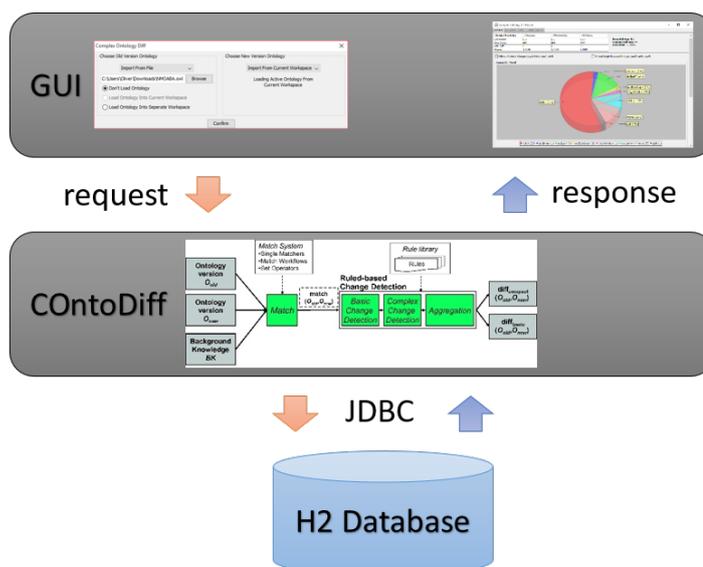


Abbildung 11: 3-Schichten Architektur des COntoDiff-Plugins

bank (siehe Kapitel 4.3), die die geeigneten Datenstrukturen für die Berechnungen des COntoDiff-Algorithmus zur Verfügung stellt. Die COntoDiff-Schicht ist in Java umgesetzt und für die Ausführung des Algorithmus zuständig. Mit Hilfe der COG-Regeln und der Datenbank werden hier logische Schritte angewendet um die Diffs, den $diff_{basic}$ und $diff_{compact}$ (siehe Kapitel 2.3), aus zwei eingelesenen Ontologieversionen zu berechnen. Das Frontend wurde mit Java Swing umgesetzt und lediglich für die Kreisdiagramme wurde eine zusätzliche Bibliothek namens JFreeChart (siehe Kapitel 4.4) genutzt. Über die Benutzeroberfläche hat der Anwender die Möglichkeit die Berechnung des COntoDiff-Algorithmus zu konfigurieren und sich nach der Ausführung durch die Ergebnisausgabe zu navigieren.

4.3 Anpassungen am ContoDiff-Algorithmus

Damit die internetunabhängige Umsetzung (siehe Anforderung Kapitel 3.1.1) gelingt und nur geringfügige Änderungen am COntoDiff-Algorithmus (siehe Kapitel Anforderung 3.1.2) notwendig sind, wurde sich für eine eingebettete Datenbanklösung entschieden. Das größte Hindernis bei der Umsetzung von COntoDiff in einem Plugin mit einer eingebetteten Datenbank war, dass der Algorithmus ursprünglich mit einer MySQL-Datenbank umgesetzt wurde und deswegen eine Vielzahl von Funktionen zum Einsatz kommen, die über den SQL-Standard 92 [7] hinausgehen und deswegen möglicherweise nicht von anderen Datenbanklösungen unterstützt werden. Dazu gehören beispielweise *REPLACE*¹¹ und *GROUP_CONCAT*¹², auf die im Folgenden näher eingegangen wird. Leider existiert keine eingebettete Datenbanklösung von MySQL, sondern nur die Möglichkeit

¹¹<http://dev.mysql.com/doc/refman/5.7/en/replace.html>

¹²http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html#function_group_concat

sich über einen eingebetteten Client mit einer externen MySQL-Datenbank zu verbinden. Bei den Recherchen zu geeigneten Datenbanklösungen, die vollständig in eine Java Applikation eingebettet werden können haben sich *Apache Derby*¹³ und *H2 Database*¹⁴ als erfolgsversprechende Kandidaten ergeben.

Apache Derby

Auch wenn Derby die Funktion `GROUP_CONCAT` nicht unterstützt, da sie nicht zum SQL-Standard 92 gehört, kann diese durch eine selbst definierte Benutzerfunktion umgesetzt werden. Jedoch sind bereits bei der Schemadefinition einige Probleme aufgetreten. Zum Beispiel ist es in MySQL bei der *CREATE TABLE*-Anweisung möglich durch *IF NOT EXISTS* eine Tabelle nur zu erstellen, wenn die Tabelle mit entsprechendem Namen noch nicht existiert. Derby unterstützt *IF NOT EXISTS* nicht und somit kann diese Abfrage nicht über den SQL-Befehl selbst erfolgen, sondern muss in den Javacode verlegt werden. Beim Versuch eine Tabelle zu erstellen, die bereits existiert, wird ansonsten ein Fehler ausgelöst. Zusätzlich kann man in MySQL durch die Anweisung *KEY* einen Index während des Erstellens einer Tabelle definieren. Mit Derby mussten diese Anweisungen alle entfernt und die Indizes durch *CREATE INDEX*-Anweisungen nachträglich hinzugefügt werden.

Nach diesen und weiteren kleinen Anpassungen funktionierte das Erstellen des Datenbankschemas mit Apache Derby. Allerdings traten weitere Probleme bei der Berechnung der Diffs des COntoDiff-Algorithmus auf. Dieser benutzt die MySQL-Funktion *REPLACE* an vielen Stellen und auch, wenn eine ähnliche Funktion *MERGE*¹⁵ in Derby existiert, kann diese nicht auf dieselbe Weise genutzt werden. *REPLACE* überschreibt den Inhalt einer Zeile in einer Tabelle, wenn der selbe Schlüssel gefunden wurde und fügt eine neue Zeile ein, wenn dem nicht so ist. Derbys *MERGE* geht im Grunde genauso vor, jedoch ist es mit dem Befehl nur möglich zwei Tabellen miteinander zu vereinen und es kann nicht der Inhalt einer einzelnen Tabelle geändert werden. Eine andere Möglichkeit *REPLACE* umzusetzen wäre die Nutzung von *INSERT*- und *UPDATE*-Anweisungen. Jedoch wäre der Aufwand hierfür sehr hoch gewesen und die Anforderung möglichst wenige Änderungen an COntoDiff vorzunehmen (siehe Kapitel 3.1.2), hätte nicht erfüllt werden können. Aus diesem Grund wurde sich gegen Apache Derby entschieden.

H2 Database

H2 unterstützt nicht nur den `GROUP_CONCAT`-Befehl¹⁶, sondern verfügt auch über einen MySQL-Kompatibilitätsmodus, der beim Erstellen der Verbindung zur Datenbank ak-

¹³<https://db.apache.org/derby/>

¹⁴<http://h2database.com/>

¹⁵<http://db.apache.org/derby/docs/10.11/ref/rrefsqjmerge.html>

¹⁶http://h2database.com/html/functions.html#group_concat

```
CREATE TABLE IF NOT EXISTS 8588871541368749334_evol_versions
(
  ...
  value1 VARCHAR(2000) DEFAULT NULL,
  value2 VARCHAR(2000) DEFAULT NULL,
  KEY value1 (value1),
  KEY value2 (value2),
  ...
)

CREATE TABLE IF NOT EXISTS 8588871541368749334_evol_change
(
  ...
  value1 VARCHAR(2000) DEFAULT NULL,
  value2 VARCHAR(2000) DEFAULT NULL,
  KEY value1 (value1),
  KEY value2 (value2),
  ...
)
```

Listing 1: Ausschnitt der Tabellendefinition in MySQL

tiviert werden kann. Durch diesen Modus waren nur sehr wenige Anpassungen notwendig, auf die im Folgenden näher eingegangen wird.

In Listing 1 ist ein Ausschnitt der Tabellendefinition zu sehen, wie sie mit MySQL umgesetzt wurde. Vor dem Erstellen der Tabellen wird jedem Tabellennamen ein 19-Stelliges Präfix mit einem Zufallsgenerator erstellt und vor den eigentlichen Tabellennamen gestellt. Bei der Erstellung der Tabelle mit Hilfe der `CREATE TABLE`-Anweisung werden auch gleich alle nötigen Indizes durch die Anweisung `KEY` initialisiert. Hierbei fällt auf, dass sowohl dieselben Spaltennamen, als auch die selben Indexnamen in beiden Tabellen auftreten. Da MySQL die Indizes pro Tabelle führt ist das hier kein Problem.

In H2 mussten sowohl Änderungen an den Präfixen, als auch an den Indizes vorgenommen werden (siehe rot markierte Stellen in Listing 2). Da in H2 ein Tabellename zwingend mit einem Buchstaben beginnen muss und nicht mit einer Zahl, Sonderzeichen oder ähnlichem beginnen darf, ist dem 19-stelligen, zufallsgenerierten Präfix ein *P* vorangestellt. Außerdem führt H2 die Indexnamen global und nicht pro Tabelle. Aus diesem Grund wurde pro Tabelle den Indizes ein Kürzel vorangestellt um diese der Tabelle zweifelsfrei zuzuordnen. Zusätzlich musste auch bei den Indizes das Präfix vorangestellt werden, um den Mehrbenutzerbetrieb zu gewährleisten. Im Falle einer parallelen Ausführung des COntoDiff-Algorithmus von mehreren Protégéinstanzen, würde die Erstellung der Tabellen scheitern, weil die Indexnamen bereits von der ersten Instanz des Plugins benutzt worden wären.

Des Weiteren existiert die Funktion `REPLACE` auch in H2 nicht. Allerdings gibt es die `MERGE`-Anweisung¹⁷, die im Gegensatz zur Funktion in Derby, in H2 so umgesetzt ist, dass man sie ohne große Anpassungen als Ersatz für `REPLACE` verwenden kann. Die beiden Befehle unterscheiden sich darin, dass `REPLACE` beim Auffinden des gleichen Schlüssels zuerst ein `DELETE` und dann ein `INSERT` und `MERGE` nur ein `UPDATE` ausführt. Dadurch kann es vorkommen, dass bei der `REPLACE`-Anweisung nicht übergebene Variablen beim `INSERT` auf ihren Standardwert gesetzt werden, wobei sie bei

¹⁷<http://www.h2database.com/html/grammar.html#merge>

```
CREATE TABLE IF NOT EXISTS P8588871541368749334_evol_versions
(
  ...
  value1 VARCHAR(2000) DEFAULT NULL,
  value2 VARCHAR(2000) DEFAULT NULL,
  KEY P8588871541368749334_ver_value1 (value1),
  KEY P8588871541368749334_ver_value2 (value2),
  ...
)

CREATE TABLE IF NOT EXISTS P8588871541368749334_evol_change
(
  ...
  value1 VARCHAR(2000) DEFAULT NULL,
  value2 VARCHAR(2000) DEFAULT NULL,
  KEY P8588871541368749334_wor_value1 (value1),
  KEY P8588871541368749334_wor_value2 (value2),
  ...
)
```

Listing 2: Ausschnitt der Tabellendefinition in H2

einem UPDATE ihren Wert beibehalten. Durch eine Erweiterung der MERGE-Befehle um betroffene Variablen, konnte diesem Problem allerdings entgegengewirkt werden.

Mit Hilfe der H2 Database war es also möglich den COntoDiff-Algorithmus, trotz seiner ursprünglichen Umsetzung mit einer MySQL-Datenbank, mit einer eingebetteten Datenbanklösung umzusetzen (siehe Anforderung Kapitel 3.1.1). Außerdem waren nur geringfügige Änderungen an den Queries des Algorithmus nötig (siehe Anforderung Kapitel 3.1.2).

4.4 Erstellung der Benutzeroberfläche

Bei der Umsetzung der Benutzeroberfläche wurde zunächst untersucht, ob sich Funktionen von Protégé zur Wiederverwendung eignen. Protégé stellt zwar eine Reihe von anpassbaren Elementen, wie zum Beispiel Views (siehe Kapitel 3.2), zur Verfügung, diese sind jedoch für die Visualisierung von Ontologien und deren Bestandteile ausgelegt. Beispielweise ist in Abbildung 3 auf der linken Seite die Klassenhierarchie einer Ontologie abgebildet, die dem angestrebten Baum für die Navigation bei der Ausgabe der Ergebnisse von COntoDiff ähnelt (siehe Abbildung 7). Allerdings ist dieser Baum bereits stark angepasst worden um den Anforderungen einer geeigneten Visualisierung einer Ontologie gerecht zu werden und so verhält es sich auch mit den anderen Darstellungsmöglichkeiten. Aus diesem Grund ist es nicht sinnvoll Elemente von Protégé für die Umsetzung des COntoDiff-Plugins zu nutzen.

Die Benutzeroberfläche des Protégé-Plugins für COntoDiff wurde mit Hilfe von Java Swing¹⁸ umgesetzt. Swing bietet die wichtigsten Elemente zur Umsetzung einer geeigneten Oberfläche zum Konfigurieren von COntoDiff und Visualisieren der Ergebnisse. Zu diesen Elementen zählen im einfachsten Fall Buttons, Checkboxes und Comboboxen. Darüber hinaus ist auch die Darstellung von umfangreichen Informationen mit

¹⁸<http://docs.oracle.com/javase/tutorial/uiswing/index.html>

Hilfe von Tabellen und Bäumen möglich. Außerdem ist keine zusätzliche Bibliothek nötig, wodurch der Umfang der Implementierung gering gehalten werden kann.

Für die Umsetzung der Kreisdiagramme, die in der Statistik der Ergebnisausgabe von COntoDiff das Verhältnis zwischen den Änderungsoperationen veranschaulichen, war Java Swing nicht ausreichend. Um eine ansprechende und übersichtliche Darstellung realisieren zu können, musste auf eine zusätzliche Bibliothek zurückgegriffen werden. JFreeChart¹⁹ bietet umfangreiche Möglichkeiten zur individuellen Darstellung von Diagrammen und ist, bis auf das Entwicklerhandbuch, frei erhältlich. Außerdem war es damit möglich die Kreisdiagramme so umzusetzen, dass diese während der Laufzeit abgeändert werden können. Der Anwender hat nämlich über zwei Checkboxen die Möglichkeit, die angezeigten Änderungsoperationen einzuschränken, indem er Änderungen an Attributen und/oder Relationen ausschließt (siehe Kapitel 5).

4.5 Veröffentlichung des COntoDiff-Plugins

Die Nutzer des Editors Protégé, die das COntoDiff-Plugin in ihrer Arbeit mit Ontolgien gerne nutzen wollen, müssen die Möglichkeit bekommen das Plugin zentral zu beziehen. Auf der einen Seite besteht die Möglichkeit im Wiki²⁰ von Protégé mit Hilfe eines neuen Eintrags das eigene Plugin vorzustellen und dem Anwender auf diesem Weg die eigene Erweiterung zur Verfügung zu stellen. Bei der Nutzung dieses Kanals tritt allerdings ein grundlegendes Problem auf: Der Anwender muss sich selbst darüber informieren, ob eine neue Version des Plugins existiert und anschließend die Installation selbst vornehmen.

Allerdings gibt es noch eine weitere Möglichkeit. Bei jedem Start von Protégé wird überprüft, ob für die installierten Plugins neue Versionen vorhanden sind und der Nutzer kann dann entscheiden ob und welche Erweiterungen er auf den neusten Stand bringen möchte. Um diese Funktion zu gewährleisten, muss das entsprechende Plugin für das automatische Update²¹ konfiguriert werden. Hierfür muss eine Datei im Internet hinterlegt werden, die alle Informationen zur aktuellen Version der Erweiterung und dem Ort, von wo das Plugin bezogen werden kann, enthält. Die URL zu dieser Updatedatei wird dann im Manifest der Erweiterung hinterlegt. Die Informationen, der im Netz hinterlegten Datei, werden dann beim Start von Protégé oder beim manuellen Abfragen von Updates, überprüft und die dort eingetragene Version, mit der Versionsnummer des installierten Plugins verglichen. Bei einer Abweichung wird die neue Version dann vom hinterlegten Ort bezogen und automatisch installiert.

Zusätzlich zum Aktualisieren von Erweiterungen hat Protégé noch eine weitere Funktion um den Nutzern einen möglichst unkomplizierten Umgang mit Plugins zu bieten und den Aufwand von Suche und Installation so gering wie möglich zu halten: Es

¹⁹<http://www.jfree.org/jfreechart/>

²⁰http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library

²¹<http://protegewiki.stanford.edu/wiki/EnablePluginAutoUpdate>

besteht die Möglichkeit, verfügbare Plugins mit zusätzlichen Informationen zur jeweiligen Erweiterung, direkt in Protégé anzuzeigen. Hierfür steht der Menüeintrag *Check for plugins...* unter dem Punkt *File* zur Verfügung. Über diesen Dialog können nicht nur neue Plugins installiert, sondern auch bereits vorhandene Erweiterungen aktualisiert werden. Um diese Möglichkeit zu nutzen, kann ein eigenes Register angegeben werden, in dem Links zu den beschriebenen Updatedateien hinterlegt sind. Außerdem ist es möglich dort weitere Register, wie das zentrale Register von Protégé, einzutragen, um die Liste an verfügbaren Plugins zu erweitern. Diese Umsetzung hat den Nachteil, dass der Endnutzer den Pfad bzw. die URL zur Registerdatei selbst festlegen muss. Deswegen eignet sich dieses Vorgehen nur, wenn man zum Beispiel mehrere Plugins in einem Unternehmen allen Nutzern zur Verfügung stellen möchte. Allerdings besteht auch die Möglichkeit, die eigene Erweiterung in das zentrale Register von Protégé eintragen zu lassen.

Das COntoDiff-Plugin sollte also einerseits im Wiki von Protégé mit Hilfe der Plugin Bibliothek veröffentlicht und auf der anderen Seite so konfiguriert werden, dass ein automatisches Update ohne Umwege möglich ist. Zusätzlich sollte sich darum bemüht werden, dass es in das zentrale Pluginregister von Protégé aufgenommen wird um es Nutzern auf einem einfachen Weg zur Verfügung stellen zu können. Außerdem können mit dieser Umsetzung weitere Nutzer auf COntoDiff aufmerksam gemacht werden. Hierbei handelt es sich allerdings um zukünftige Arbeit.

5 Evaluierung des Plugins

In diesem Kapitel wird das COntoDiff-Plugin Mit Hilfe einer Beispielontologie evaluiert. Hierbei werden die Funktionen der Benutzeroberfläche des Plugins ausführlich beschrieben und mit Screenshots veranschaulicht. Zusätzlich werden die Funktionalitäten von Protégé OWL Diff beschrieben. Außerdem werden die Ergebnisse des COntoDiff-Algorithmus und von Protégé OWL Diff ausgewertet und gegenübergestellt.

5.1 Benutzeroberflächen von COntoDiff und Protégé OWL Diff

Die Benutzeroberflächen vom COntoDiff-Plugin und Protégé OWL Diff bieten dem Anwender die Möglichkeit eine Diffberechnung innerhalb des Ontologieeditors Protégé auszuführen. Im Folgenden werden die Funktionen der Ein- und Ausgabe der beiden Werkzeuge betrachtet und gegenübergestellt.

Importieren der Ontologieversionen

Der Importdialog des COntoDiff-Plugins bietet dem Anwender verschiedene Möglichkeiten die beiden Ontologieversionen zur Diffberechnung zu laden. In Abbildung 12 ist zu sehen, dass die alte Version der Ontologie mittels der Option *Import From File* der Checkbox im oberen Abschnitt aus einer Datei importiert wird. Falls der Benutzer eine Ontologie bereits in der Arbeitsumgebung von Protégé geöffnet hat, kann er diese, mit Hilfe der Option *Import From Current Workspace*, von dort importieren. Diese Auswahl wurde in der Abbildung für die neue Ontologieversion getroffen. Beim Import einer Ontologie aus einer Datei stehen dem Anwender zusätzliche Funktionen zur Verfügung. Mit der Option *Don't Load Ontology* kann sich der Benutzer dazu entscheiden, sich nur die Ergebnisse ausgeben zu lassen, ohne die Ontologie in irgendeiner Form in Protégé zu öffnen. Mit Hilfe von *Load Ontology Into Current Workspace* kann die Ontologieversion in der aktuellen Arbeitsumgebung, von der aus das Plugin gestartet wurde, geöffnet werden. Um den Anwender vor Fehleingaben zu bewahren, steht diese Option nur zur Verfügung, wenn die andere Ontologieversion weder aus dem aktuellen Workspace importiert, noch in diesem geöffnet wird. Zusätzlich wird verhindert, dass beide Versionen der Ontologie aus der aktuellen Arbeitsumgebung importiert werden

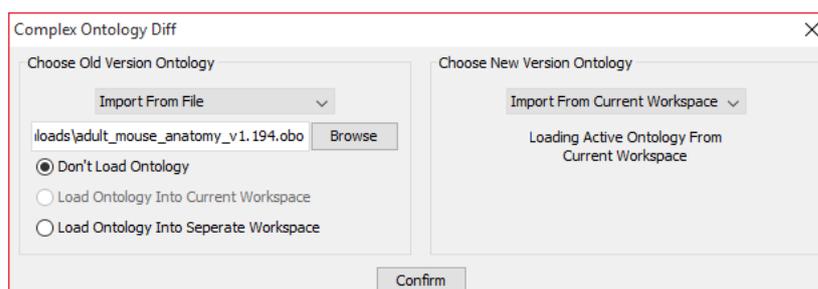


Abbildung 12: Dialog zum Importieren von Ontologieversionen in COntoDiff

können, da diese in diesem Fall identisch wären. Außerdem kann der Benutzer sich dazu entscheiden, die Ontologie in einem zusätzlichen Workspace zu öffnen, indem er *Load Ontology Into Seperate Workspace* auswählt.

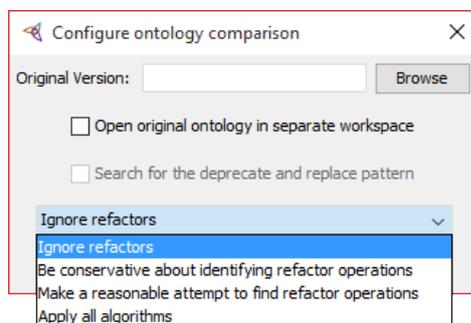


Abbildung 13: Dialog zum Importieren von Ontologieversionen in Protégé OWL Diff

Der Importdialog von Protégé OWL Diff ist weniger intuitiv aufgebaut. In Abbildung 13 ist zu sehen, dass nur eine Version der Ontologie als Datei importiert werden kann. *Original Version* bezeichnet hier die alte Ontologieversion. Die neue Version der Ontologie muss bereits in der aktuellen Arbeitsumgebung von Protégé geöffnet sein um die Diffberechnung sinnvoll ausführen zu können. Außerdem ist es, wie beim COnToDiff-Plugin möglich, die alte Ontologieversion in einen zusätzlichen Workspace zu laden. Zusätzlich können weitere Algorithmen zum Finden von strukturellen Änderungen (*refactor operations*) angewendet werden. Diese haben bei der genutzten Ontologie allerdings keine Verbesserungen des Ergebnisses hervorbringen können.

Statistik

Nach dem Ausführen des COnToDiff-Algorithmus mit den zuvor importierten Ontologieversionen, steht dem Anwender eine ausführliche Statistik zur Verfügung. Ein Teil der Ausgabe ist in Abbildung 14 zu sehen. Die Tabelle befindet sich im oberen Bereich des Statistiktabs und bietet einen detaillierten Überblick über die beiden Versionen der Ontologie. Hier werden nicht nur die genauen Vorkommen von Konzepten, Attributen und Beziehungen beziffert, sondern auch die Differenz dieser Werte und deren Wachstum angegeben. Neben der Tabelle werden die Größen von $diff_{basic}$ und $diff_{compact}$

Version Statistics	#Concepts	#Relationships	#Attributes
Old Version	2882	3626	9095
New Version	2924	3683	9241
Abs. Diff	42	57	146
Growth	1,0146	1,0157	1,0161

Basic Diff Size: 311
Compact Diff Size: 63
Reduction: 20,257%

Abbildung 14: Statistik der Unterschiede zwischen den Ontologieversionen – Tabelle

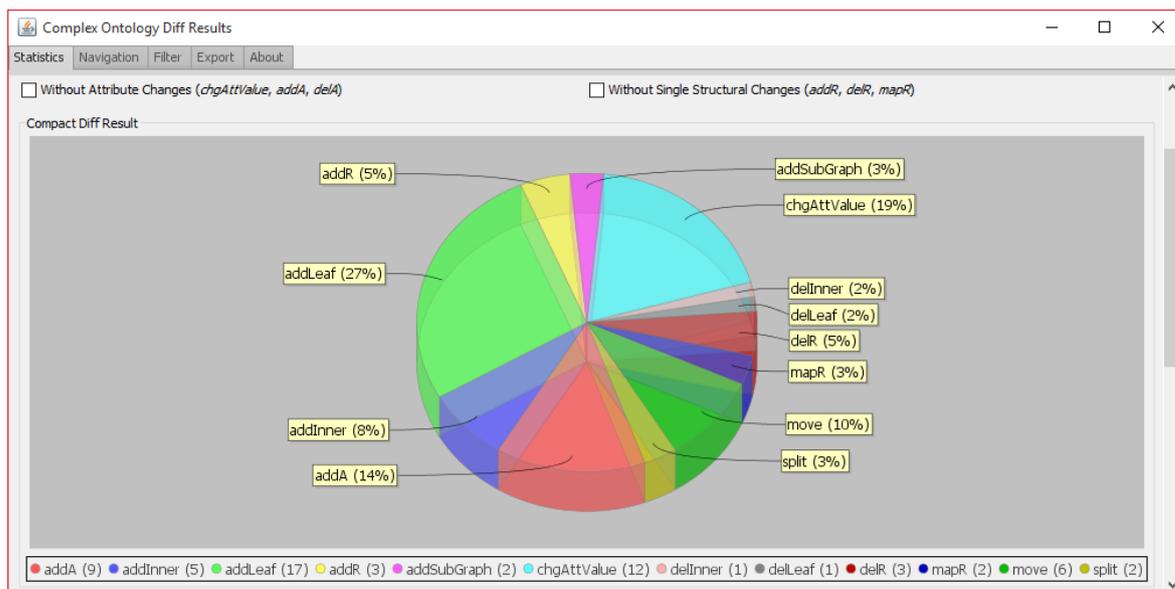


Abbildung 15: Statistik der Unterschiede zwischen den Ontologieversionen – Kompakter Diff mit allen Änderungsoperationen

angezeigt und die Reduktion der Änderungsoperationen angegeben, die beim Erstellen des $diff_{compact}$ aus dem $diff_{basic}$ erfolgte. In Abbildung 15 ist der Bereich unterhalb der Tabelle dargestellt. Dort ist ein Kreisdiagramm zu sehen, welches das Verhältnis zwischen den unterschiedlichen Änderungsoperationen illustriert. Am Diagramm selbst sind Labels dargestellt, die den Anteil der Operationen in Prozent angeben und unter dem Diagramm befindet sich eine Legende mit der genauen Anzahl an Vorkommen der jeweiligen Änderungsoperation. Mit den Checkboxes oberhalb des Kreisdiagramms können die angezeigten Operationen weiter eingeschränkt werden. In Abbildung 16

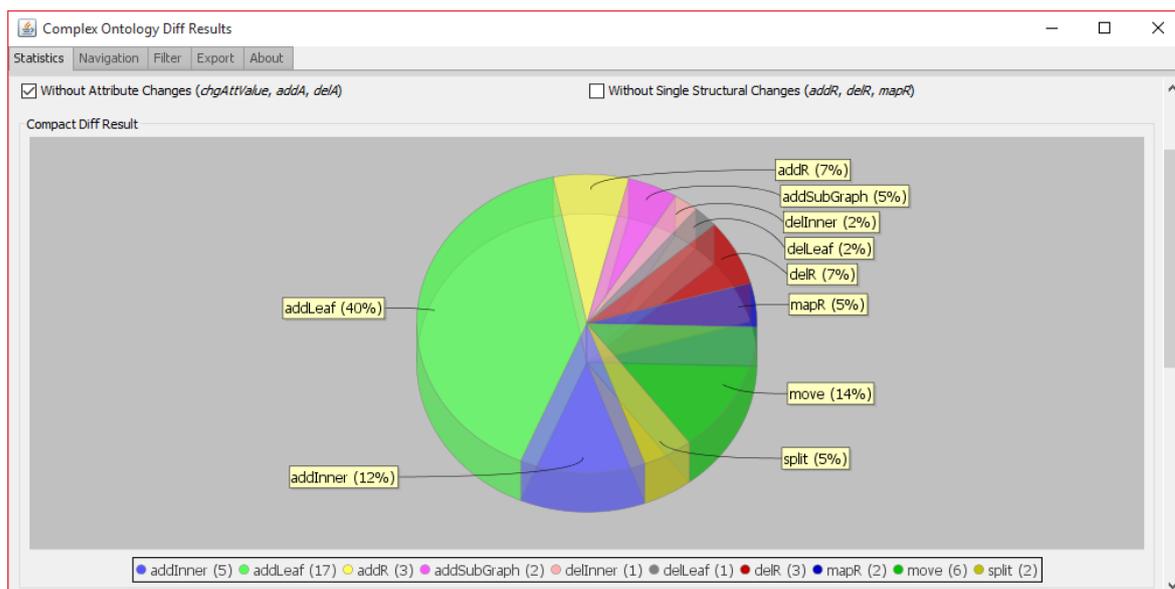


Abbildung 16: Statistik der Unterschiede zwischen den Ontologieversionen – Kompakter Diff ohne Attributänderungen

wurde *Without Attribute Changes* gewählt. Das Kreisdiagramm passt sich in Echtzeit der Auswahl an und stellt die gewünschten Informationen übersichtlich dar. Zusätzlich können auch strukturelle Änderungen ausgeschlossen werden. Unter dem Diagramm für den kompakten Diff wird ein Kreisdiagramm für den Basisdiff angezeigt, welches mit Hilfe der Checkboxes ebenso in Echtzeit angepasst werden kann.

Im Protégé OWL Diff findet der Nutzer, im Gegensatz zur ausführlichen Statistik vom COntoDiff-Plugin, lediglich eine kurze Auflistung der vorgekommenen Änderungen (*created*, *deleted*, *renamed* und *modified*) in der unteren Leiste des Ausgabefensters vor.

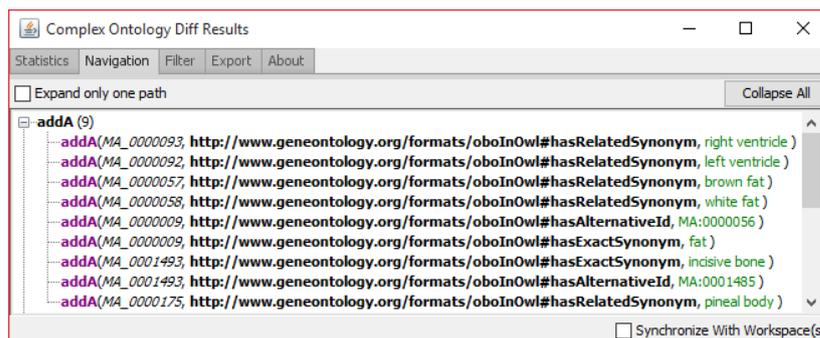


Abbildung 17: Navigation mit aufgefalteten *addA*-Änderungsoperationen

Navigation durch die Änderungsoperationen

Im Reiter Navigation hat der Anwender die Möglichkeit sich alle aufgetretenen Änderungsoperationen im Detail anzusehen. In Abbildung 17 kann man sehen, dass bei der Darstellung ein Baum gewählt wurde, in dem auf oberster Ebene alle Operationen aufgelistet sind. Zum Beispiel treten in der Abbildung 9 Attributhinzufügungen (*addA*) auf. Beim Auffalten der jeweiligen Änderungsoperation kann man diese entweder im Detail betrachten oder es wird aufgeschlüsselt, aus welchen weniger kompakten Operationen diese zusammengesetzt ist. In Abbildung 17 sind die *addA*-Operationen im Detail abgebildet und setzen sich aus der betroffenen Entität, dem hinzugefügten Attribut und dessen Wert zusammen. In Abbildung 19 (Mitte) kann man hingegen bei einer Verschiebung (*move*) sehen, dass sich diese aus dem Entfernen (*delR*) und Hinzufügen (*addR*) von Relationen zusammensetzt. Um dem Anwender die Benutzung der

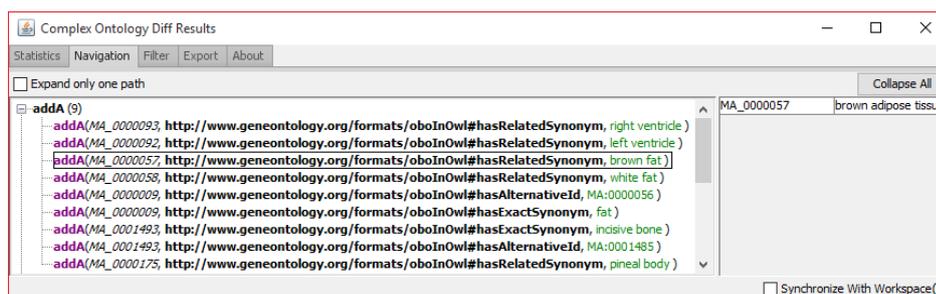


Abbildung 18: Navigation mit Auswahl der Entität *brown adipose tissue*

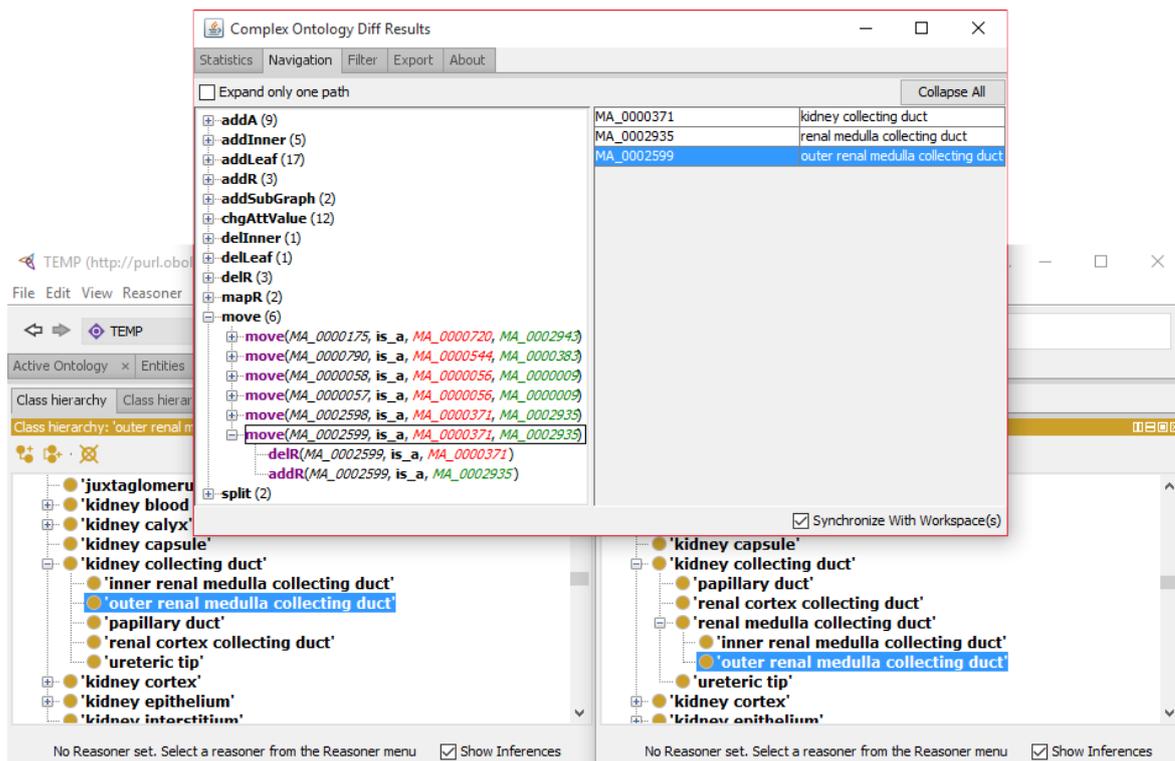


Abbildung 19: Synchronisation der Navigation mit den in Protégé geöffneten Ontologieversionen

Navigation so einfach wie möglich zu machen, findet man oberhalb des Baumes zwei Schaltflächen. Mit Hilfe von *Expand only one path* kann der Benutzer in großen Bäumen die Übersicht behalten, indem immer nur ein Pfad auf einmal aufgeklappt wird. Zuvor geöffnete Pfade werden sofort geschlossen, wenn ein anderer Pfad geöffnet wird. Mit der Schaltfläche *Collapse All* kann der gesamte Baum auf einmal zusammengeklappt werden. Während im Baum nur die Zugriffsschlüssel der Entitäten dargestellt werden, gibt es auch die Möglichkeit sich deren entsprechende Labels anzeigen zu lassen. In Abbildung 18 ist zu sehen, dass bei der Auswahl der Änderungsoperation *addA* des Konzepts *MA_000057* eine verschiebbare Tabelle eingeblendet wird, die den Zugriffsschlüssel der Entität mit dessen Label (*brown adipose tissue*) anzeigt.

Eine wichtige Funktion des COnToDiff-Plugins ist die Interaktion mit den Ontologien, die in Protégé geöffnet wurden. Mit der Checkbox *Synchronize With Workspace(s)* hat der Anwender die Möglichkeit durch das Auswählen einer Änderungsoperation, eine beteiligte Entität in den geöffneten Arbeitsumgebungen zu selektieren. Sind an einer Operation mehrere Entitäten beteiligt, kann der Benutzer aus der Labeltabelle auswählen, welche Entität selektiert werden soll. In Abbildung 19 sind zwei Workspaces zu sehen. In der linken Arbeitsumgebung ist die alte und in der Rechten die neue Ontologieversion geöffnet, in der Mitte ist die Navigation der COnToDiff-Ausgabe zu sehen. Bei der Auswahl der *move*-Änderungsoperation wird die Tabelle für die beteiligten Entitäten *MA_0000371*, *MA_0002935* und *MA_0002599* mit deren Labels angezeigt. Durch die aktive Synchronisation mit den Workspaces und dem Auswählen der Entität

MA_0002599 in der Tabelle, wird das Konzept *outer renal medulla collection duct* in den beiden Visualisierungen der Ontologien selektiert und auch die Klassenhierarchie dementsprechend aufgefaltet. Wenn der Anwender nun *MA_0000371* in der Tabelle auswählt, verschiebt sich die Selektion in beiden Arbeitsumgebungen auf das Konzept *kidney collecting duct*. Würde eine beteiligte Entität in einer der Ontologieversionen noch nicht existieren, würde bei der Auswahl die Selektion in dieser Version nicht verändert werden. Mit dieser Funktionalität hat der Anwender die Möglichkeit, Änderungen direkt in den geöffneten Ontologieversionen zu betrachten.

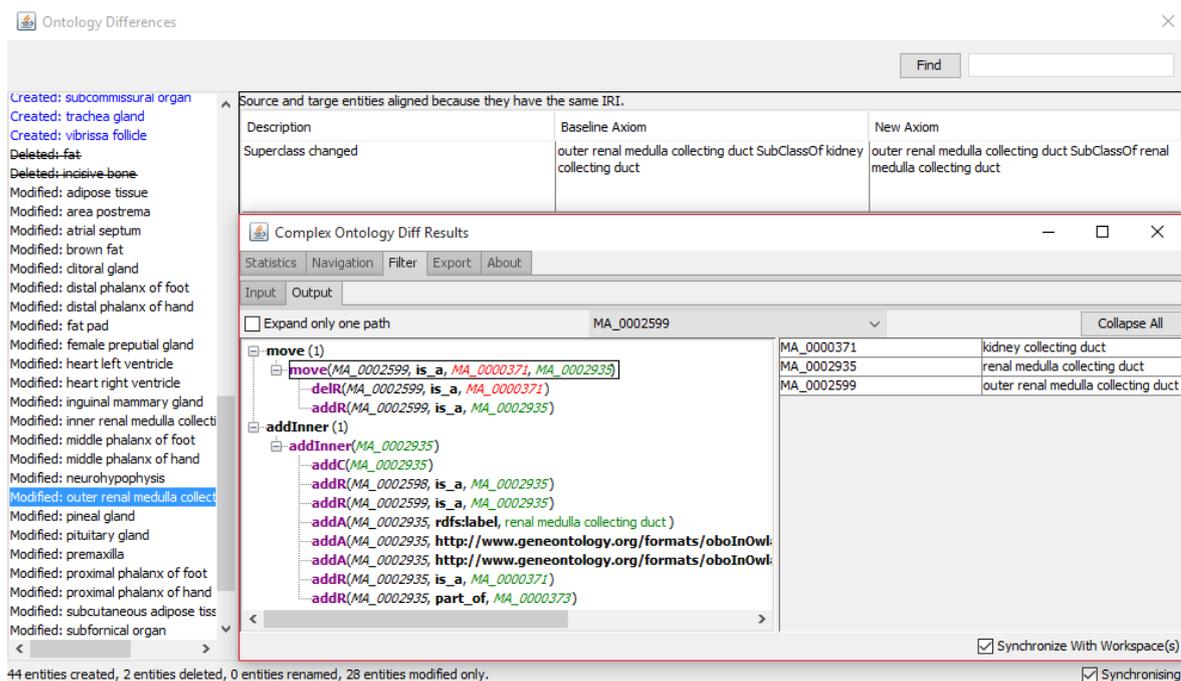


Abbildung 20: Vergleich der Änderungsoperationen für *outer renal medulla collection duct* (*MA_0002599*) in Protégé OWL Diff und COnToDiff

In Abbildung 20 sieht man, dass die Änderungsoperationen in Protégé OWL Diff auf der linken Seite lediglich untereinander aufgelistet sind. Beim Auswählen einer Änderung werden deren Details in Tabellenform in der rechten Seite des Ausgabefensters dargestellt. Auch wenn diese Darstellung aufgrund der einfachen Änderungsoperationen des Tools ausreicht, kann es bei vielen Änderungen unübersichtlich werden und die Entwicklung der Ontologie ist schwerer nachzuvollziehen. Protégé OWL Diff verfügt zwar auch über die Möglichkeit, die Änderungsoperationen mit den geöffneten Ontologien zu synchronisieren, allerdings funktioniert die Selektion nur mit der Auswahl auf der linken Seite. Die Tabelle auf der rechten Seite bietet keine Möglichkeit dort aufgeführte Entitäten so auszuwählen, dass diese in den Ontologien selektiert werden. Wenn der Anwender beispielsweise das Konzept *kidney collecting duct* in den Ontologieversionen ausgewählt haben möchte, bietet das Werkzeug keine Möglichkeit dazu, da das Konzept nur indirekt an Änderungen beteiligt ist.

Nach bestimmten Entitäten filtern

Über den Reiter Filter im Ausgabefenster des COntoDiff-Plugins hat der Anwender die Möglichkeit, sich Änderungsoperationen für bestimmte Entitäten anzeigen zu lassen. In Abbildung 21 sind bereits drei Zugriffsschlüssel in das Textfeld *ID's To Filter For* eingetragen. Der Benutzer kann die Liste entweder manuell fortsetzen oder Entitäten aus einer Arbeitsumgebung bzw. aus dem Navigationstab des Plugins auswählen, indem er die Schaltfläche *Select In Workspace/Navigation* nutzt. Dabei öffnet sich ein zusätzliches Fenster, welches immer im Vordergrund bleibt und dieselbe Liste von Zugriffsschlüsseln enthält. In Abbildung 22 kann man sehen, dass man dann zur Navigation wechseln kann und weiterhin sieht, welche Entitäten bereits in die Filterliste eingetragen wurden. Nach der Auswahl des Konzepts *adipose tissue* wird dessen Zugriffsschlüssel in die Liste übernommen. Auf die gleiche Weise können Zugriffsschlüssel durch die Auswahl von Entitäten in der Klassenhierarchie in den Arbeitsumgebungen von Protégé zur Filterliste hinzugefügt werden. Erst wenn der Anwender die Schaltfläche *Done* betätigt, wird das zusätzliche Fenster geschlossen und das automatische Befüllen der Liste beendet. Nach dem Bestätigen mittels der Schaltfläche *Confirm* können im Reiter Output die Ergebnisse betrachtet werden. Die Ausgabe ist hier ähnlich dem Navigationstab, mit der Ausnahme, dass sich in der Mitte eine zusätzliche Combobox befindet. In Abbildung 23 sieht man, dass der Benutzer über diese Combobox auswählen kann, für welche Entität die Änderungsoperationen angezeigt werden sollen. Daraufhin wird ein Baum dargestellt, der nur die Operationen enthält, an denen die ausgewählte Entität beteiligt ist. Abgesehen davon, findet man die gleichen Funktionen, wie bei der Navigation vor. (siehe Abschnitt Navigation durch die Änderungsoperationen)

In Abbildung 20 kann man sehen, dass sich in Protégé OWL Diff eine Suchfunktion in der oberen rechten Ecke befindet. In das Textfeld muss ein Label als Suchbegriff

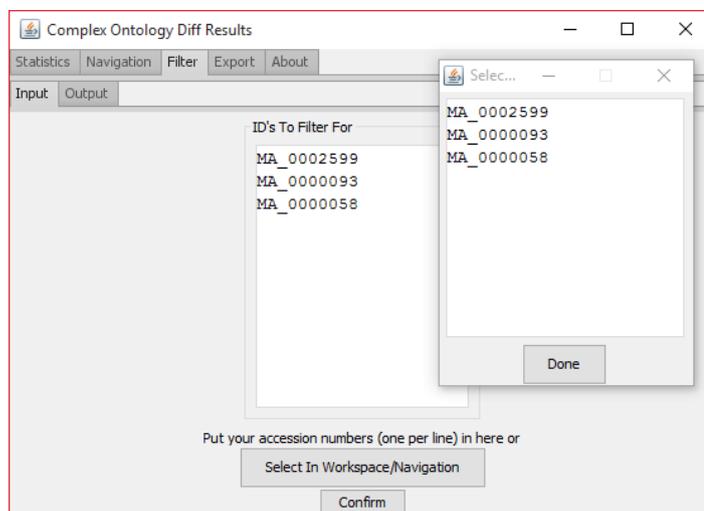


Abbildung 21: Importdialog für das Filtern von Entitäten

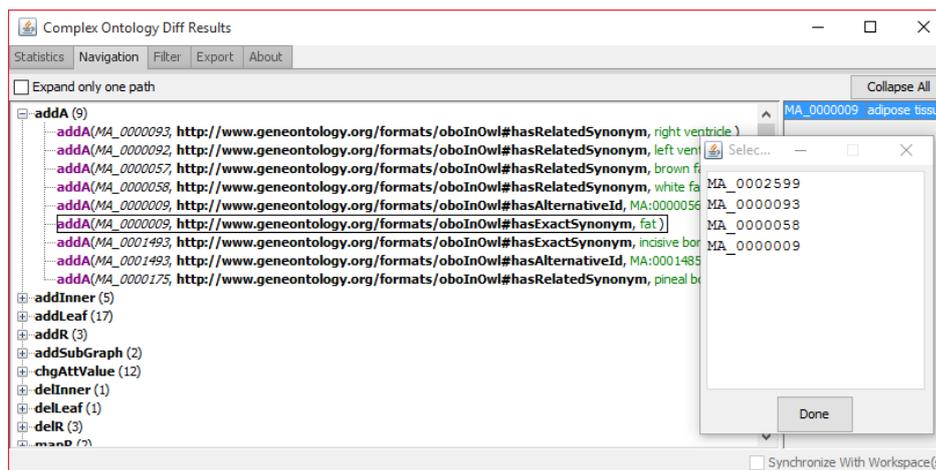


Abbildung 22: Auswahl von zu filternden Entitäten aus der Navigation

eingetragen werden. Nach dem Betätigen der Schaltfläche *Filter* wird ein zusätzliches Fenster mit einer Liste der gefundenen Entitäten eingeblendet. Der Anwender hat dann die Möglichkeit eine Entität aus dieser Auflistung zu markieren. Nach dem Markieren wird die entsprechende Änderung in der Ergebnisausgabe ausgewählt, jedoch nur, wenn die Synchronisation zuvor eingeschaltet wurde. Außerdem müssen mehrere Suchen gestartet werden, um verschiedene Entitäten zu finden. Wenn die Suche nur ein Ergebnis anzeigt, kann dieses nur einmal ausgewählt und nicht wieder abgewählt werden. Das Problem hierbei ist, dass die Synchronisation nur funktioniert, wenn eine neue Auswahl getroffen wird und das ist in diesem Fall nicht möglich. Außerdem können Entitäten, die nur an Änderungsoperationen beteiligt sind, nicht gesucht werden, da nur die linke Auflistung in die Suche einbezogen wird. Das betrifft zum Beispiel *kidney collecting duct* (siehe Abschnitt Navigation durch die Änderungsoperationen).

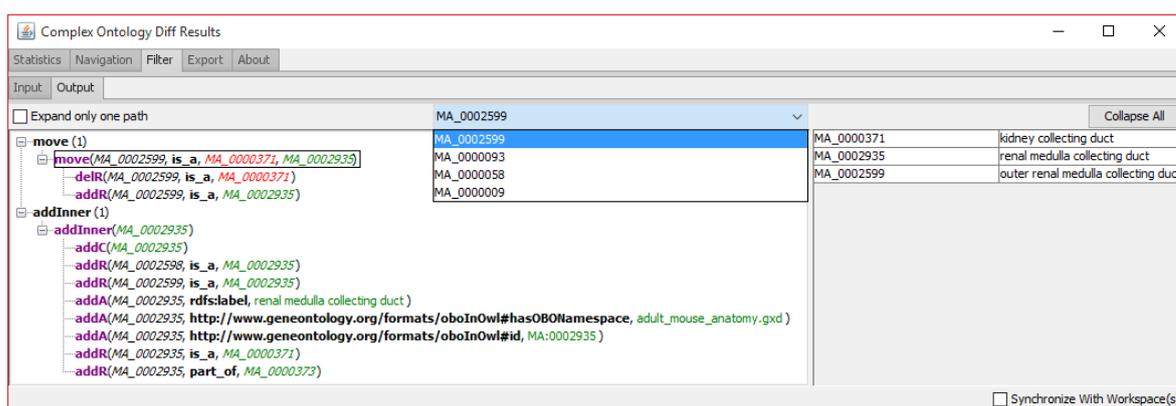


Abbildung 23: Ausgabe der Änderungsoperationen der gefilterten Entitäten

Exportieren der Ergebnisdaten

Über den Reiter *Export* hat der Anwender des COnToDiff-Plugins die Möglichkeit, die Ergebnisse des Vergleichs zweier Ontologieversionen für die weitere Verwendung

zu exportieren. In Abbildung 24 ist im Bereich *Overall Statistics* eine Statistik der Änderungsoperationen für den *diff_{basic}* und den *diff_{compact}* zu sehen. Hier werden alle möglichen Operationen aufgeschlüsselt und ausgegeben, wie oft diese jeweils in der Berechnung gefunden wurden. Im Abschnitt *All Changes* werden alle Änderungsoperationen detailliert in Textform dargestellt. Da es sich hierbei um Textfelder handelt, kann der Benutzer einzelne Teile kopieren oder die Ausgabe vor dem Abspeichern bearbeiten. Über die Schaltfläche *Save to file* können die Ergebnisdaten über einen Dateidialog abgespeichert werden.

In Protégé OWL Diff gibt es leider keine Möglichkeit die Ergebnisse zu exportieren. Die Darstellung der Änderungsdetails als Tabelle bietet keine Funktionalitäten zum Kopieren der Daten.

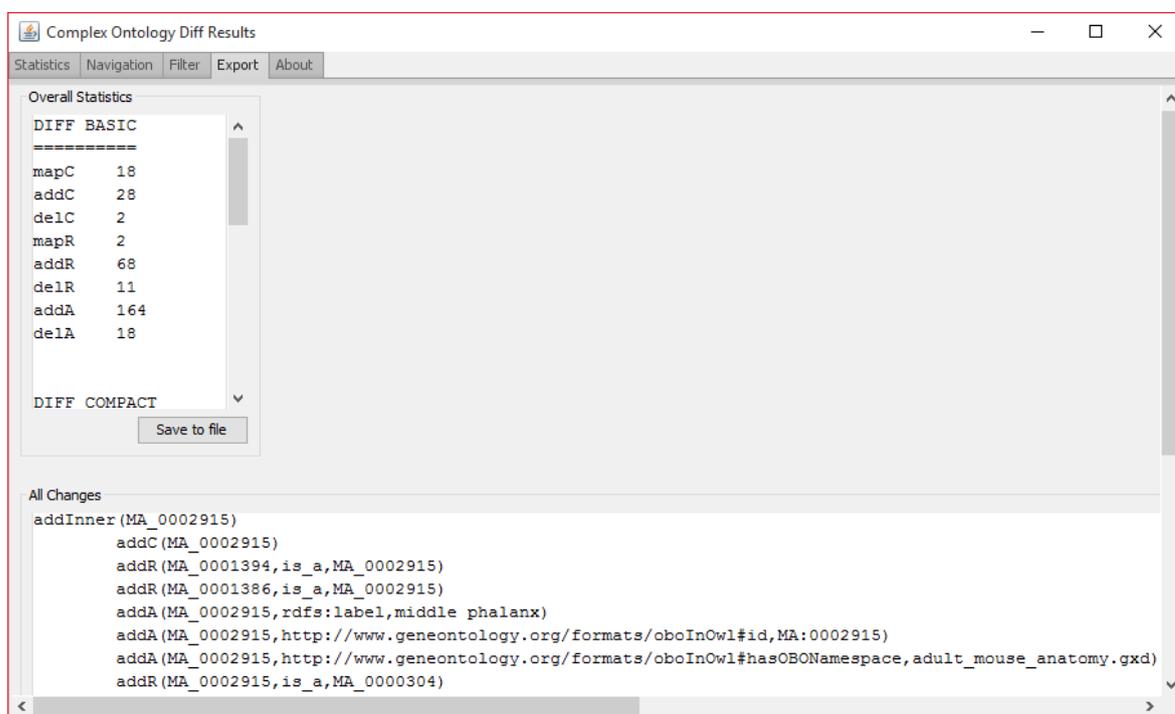


Abbildung 24: Exportdialog zum Abspeichern der Ergebnisse

5.2 Auswertung der Ergebnisse

Um die Ergebnisse des COntoDiff-Algorithmus und Protégé OWL Diff gegenüberzustellen werden die ausgegebenen Änderungsoperationen näher betrachtet. In Tabelle 2 kann man sehen welche Operationen der *diff_{basic}* und *diff_{compact}* enthalten. Protégé OWL Diff stellt lediglich eine Statistik zu den Änderungsoperationen zur Verfügung, die in der Spalte *Protégé OWL Diff_{compact}* zu sehen sind. Die Werte in der Spalte *Protégé OWL Diff_{basic}* müssen vom Anwender selbst erhoben werden. Die Spalten von Protégé OWL Diff wurde zum besseren Verständnis weitestgehend an das Vokabular von COntoDiff angeglichen. Beispielsweise sind alle Hinzufügungen, wie *addC*, *addR*

diff _{basic}		diff _{compact}		Protégé OWL Diff _{basic}		Protégé OWL Diff _{compact}	
Änderungsoperation	Anzahl	Änderungsoperation	Anzahl	Änderungsoperation	Anzahl	Änderungsoperation	Anzahl
addC	28	addInner	5	addC	44	created	44
delC	2	addLeaf	17	delC	2	deleted	2
mapC	18	addSubGraph	2	addR	61	modified	28
addR	68	delInner	1	delR	5		
delR	11	delLeaf	1	superclass changed	8		
mapR	2	split	2	addA	150		
addA	164	addR	3	delA	6		
delA	18	delR	3	annotation changed	12		
		mapR	2				
		move	6				
		addA	9				
		chgAttValue	12				
Summe	311	Summe	63	Summe	288	Summe	74

Tabelle 2: Vergleich der gefundenen Änderungsoperationen im $diff_{basic}$ und $diff_{compact}$ von COnToDiff und dem Diff von Protégé OWL Diff

und *addA*, in Protégé OWL Diff lediglich mit *added* bezeichnet. Die zusammenfassende Änderungsoperation *created* enthält nur Hinzufügungen, *deleted* nur Löschungen und *modified* kann sowohl *added*, *deleted*, *annotation changed* oder *superclass changed* enthalten. Es ist zu erkennen, dass der $diff_{compact}$ etwas kompakter als die zusammengefassten Änderungsoperationen vom OWL Diff ausfällt. Außerdem kann der Nutzer, mit den aussagekräftigen Bezeichnungen von COnToDiff, einfacher Analysen an der Entwicklung einer Ontologie vornehmen. Es können allerdings auch Gemeinsamkeiten festgestellt werden. So entspricht *annotation changed* von Protégé OWL Diff der Operation *chgAttValue* in COnToDiff, was auch an der gleichen Vorkommenshäufigkeit zu erkennen ist. *superclass changed* ist einem *move* in COnToDiff ähnlich. Die im OWL Diff gefundenen Änderungen sind in COnToDiff jedoch in *move* und *mapR* vorzufinden.

In Abbildung 20 kann man ein Beispiel für eine *superclass changed* Operation sehen. Diese wird dem *move* von COnToDiff gegenübergestellt. Man kann erkennen, dass es sich bei der Änderungsoperation in Protégé OWL Diff um eine Basisänderung handelt, wohingegen ein *move* eine zusammengesetzte Operation aus der Löschung und der Hinzufügung einer Beziehung ist. Aus diesem Grund sind in Tabelle 2 im Vergleich zum $diff_{basic}$, weniger *addR*- und *delR*-Operationen beim OWL Diff vorzufinden. Außerdem ist zu erkennen, dass für *outer renal medulla collecting duct* lediglich eine Änderungsoperation in Protégé OWL Diff ausgegeben wird und das Filtern mit dem COnToDiff-Plugin zusätzlich Änderungen anzeigt, an denen dieses Konzept beteiligt ist. Um im OWL Diff dieselben Daten zu erhalten, muss man weiter in den Änderungsoperationen suchen und bekommt keine übersichtliche Auflistung aller Änderungen, an denen dieses Konzept beteiligt ist.

Protégé OWL Diff liefert einen guten Diff mit einfachen Änderungsoperationen und ist für den schnellen Vergleich von Ontologieversionen gut geeignet. Allerdings bietet es kaum Analysefunktionalitäten und ist umständlich zu handhaben. Mit der direkten Einbindung in Protégé und den Interaktionsmöglichkeiten mit der Klassenhierarchie

von in Protégé geöffneten Ontologien hat es einige Vorteile, die COntoDiff nur teilweise erfüllen kann. Das COntoDiff-Plugin schneidet gegenüber Protégé OWL Diff mit einer guten Benutzerführung, einer ausführlichen Statistik und leicht verständlichen, kompakten Änderungsoperationen gut ab und bietet dem Anwender bei der Ontologieentwicklung ein nützliches Werkzeug, welches in Zukunft weiter verbessert werden wird.

6 Kurzzusammenfassung und Ausblick

Die vorliegende Arbeit beschäftigt sich mit der Umsetzung des COnToDiff-Algorithmus innerhalb eines Plugins für den Ontologieeditor Protégé. Dabei wurde die bisherige Implementierung des Algorithmus so angepasst, dass COnToDiff ohne die Notwendigkeit einer externen Datenbank ausgeführt werden kann. Außerdem wurde eine eigene Benutzeroberfläche entworfen und umgesetzt. Über diese Oberfläche stehen dem Anwender umfangreiche Möglichkeiten zum Importieren der Ontologieversionen zur Verfügung. Außerdem bietet sie die geeignete Darstellung für die Ergebnisausgabe mit einer ausführlichen Statistik und der Unterstützung umfangreicher Diff- und Analysefunktionalitäten. Des Weiteren ermöglicht das COnToDiff-Plugin die Interaktion mit der Klassenhierarchie einer in Protégé geöffneten Ontologie und vereinfacht damit die Analyse und Entwicklung von Ontologien aus allen Domänen. Gegenüber Protégé OWL Diff kann COnToDiff vor allem durch seinen kompakten Änderungsoperationen und zusätzlichen Funktionen, wie dem Export der Analysedaten, überzeugen.

Dennoch gibt es einige Einschränkungen für das COnToDiff-Plugin. Bisher gibt es nur die Möglichkeit eine Ontologie aus der Arbeitsumgebung zu importieren, in der das Plugin gestartet wurde. Das bedeutet, dass mindestens eine der beiden Ontologieversionen aus einer Datei geladen werden muss. Des Weiteren ist es bisher nicht möglich, Zugriffsschlüssel für das Filtern von Entitäten aus Arbeitsumgebungen hinzuzufügen, die nicht direkt in Verbindung mit dem Plugin stehen. Dabei handelt es sich um Umgebungen, die nicht vom COnToDiff-Plugin geöffnet wurden. Außerdem gibt es Einschränkungen beim COnToDiff-Algorithmus selbst, wie zum Beispiel die fehlende Abdeckung von OWL-Konstrukten. Dazu gehören beispielsweise disjunkte Konzepte.

In Zukunft können die aufgeführten Funktionen für die weitere Verbesserung der Bedienung des Plugins hinzugefügt werden. Außerdem soll die Veröffentlichung des COnToDiff-Plugins für Protégé realisiert und das Analyseergebnis durch den Einsatz eines verbesserten Parsers weiter verbessert werden.

Literaturverzeichnis

- [1] Jonathan Bard, Seung Y Rhee, and Michael Ashburner. An ontology for cell types. *Genome biology*, 6(2):R21, 2005.
- [2] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.
- [3] Dan Brickley and Ramanathan V Guha. RDF vocabulary description language 1.0: RDF schema. 2004.
- [4] Gene Ontology Consortium et al. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Research*, 32(suppl 1):D258–D261, 2004.
- [5] Gene Ontology Consortium et al. Gene ontology consortium: going forward. *Nucleic Acids Research*, 43(D1):D1049–D1056, 2015.
- [6] Plant Ontology Consortium et al. The Plant OntologyTM consortium and plant ontologies. *International Journal of Genomics*, 3(2):137–142, 2002.
- [7] Chris J Date and Hugh Darwen. *A guide to the SQL Standard: a user's guide to the standard relational language SQL*, volume 55822. Addison-Wesley Longman, 1993.
- [8] John Day-Richter, Midori A Harris, Melissa Haendel, Suzanna Lewis, et al. OBO-Edit—an ontology editor for biologists. *Bioinformatics*, 23(16):2198–2200, 2007.
- [9] Mary E Dolan, Li Ni, Evelyn Camon, and Judith A Blake. A procedure for assessing go annotation consistency. *Bioinformatics*, 21(suppl 1):i136–i143, 2005.
- [10] John H Gennari, Mark A Musen, Ray W Ferguson, William E Grosso, Monica Crubézy, Henrik Eriksson, Natalya F Noy, and Samson W Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-computer studies*, 58(1):89–123, 2003.
- [11] GO. The OBO Flat File Format Specification. Website, November 2004. Version 1.2
http://oboformat.googlecode.com/svn/trunk/doc/GO.format.obo-1_2.html, abgerufen am 22.06.2015.
- [12] James Gosling and Henry McGilton. The Java language environment. *Sun Microsystems Computer Company*, 2550, 1995.
- [13] Anika Groß. Evolution von ontologiebasierten Mappings in den Lebenswissenschaften. 2013.
- [14] Anika Groß, Julio Cesar Dos Reis, Michael Hartung, Cédric Pruski, and Erhard Rahm. Semi-automatic adaptation of mappings between life science ontologies. In *Data Integration in the Life Sciences*, pages 90–104. Springer, 2013.
- [15] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.
- [16] Tom Gruber. Ontology. *Encyclopedia of database systems*, pages 1963–1965, 2009.

-
- [17] Graham Hamilton, Rick Cattell, Maydene Fisher, et al. *JDBC Database Access with Java*, volume 7. Addison Wesley, 1997.
- [18] Michael Hartung, Anika Gross, and Erhard Rahm. CODEX: exploration of semantic changes between ontology versions. *Bioinformatics*, 28(6):895–896, 2012.
- [19] Michael Hartung, Anika Groß, and Erhard Rahm. COnto-Diff: generation of complex evolution mappings for life science ontologies. *Journal of biomedical informatics*, 46(1):15–32, 2013.
- [20] Chen He-ping, He Lu, and Chen Bin. Research and implementation of ontology automatic construction based on relational database. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 5, pages 1078–1081. IEEE, 2008.
- [21] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The knowledge engineering review*, 18(01):1–31, 2003.
- [22] Graham Kemp. *Data Integration in the Life Sciences*. Springer, 2010.
- [23] Michael Kifer and Georg Lausen. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In *ACM SIGMOD Record*, volume 18, pages 134–146. ACM, 1989.
- [24] Toralf Kirsten, Anika Gross, Michael Hartung, Erhard Rahm, et al. GOMMA: a component-based infrastructure for managing and analyzing life science ontologies and their evolution. *J. Biomedical Semantics*, 2(6), 2011.
- [25] Markus Kuhnt. Ontologiesprachen im Kontext des Semantik Web. Technical report, Technical report, Universität Ulm, 05.02, 2003.
- [26] Ora Lassila and Deborah McGuinness. The role of frame-based representation on the semantic web. *Linköping Electronic Articles in Computer and Information Science*, 6(5):2001, 2001.
- [27] Ora Lassila and Ralph R Swick. Resource description framework (RDF) model and syntax specification. 1999.
- [28] Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley. *The Java virtual machine specification*. Pearson Education, 2014.
- [29] Carolyn E Lipscomb. Medical subject headings (MeSH). *Bulletin of the Medical Library Association*, 88(3):265, 2000.
- [30] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C Recommendation*, 10(10):2004, 2004.
- [31] Natalya Fridman Noy, Mark A Musen, et al. Promptdiff: A fixed-point algorithm for comparing ontology versions. *AAAI/IAAI*, 2002:744–750, 2002.
- [32] Thanh Tho Quan, Siu Cheung Hui, Alvis Cheuk M Fong, and Tru Hoang Cao. Automatic generation of ontology for scholarly semantic web. In *The Semantic Web-ISWC 2004*, pages 726–740. Springer, 2004.

-
- [33] Dave Raggett, Arnaud Le Hors, Ian Jacobs, et al. Html 4.01 specification. *W3C recommendation*, 24, 1999.
 - [34] Robert Signore, Michael O Stegman, and John Creamer. *The ODBC solution: Open database connectivity in distributed environments*. McGraw-Hill, Inc., 1995.
 - [35] Ljiljana Stojanovic. *Methods and tools for ontology evolution*. 2004.
 - [36] Ljiljana Stojanovic and Boris Motik. *Ontology evolution within ontology editors*. In *Proceedings of the OntoWeb-SIG3 Workshop*, pages 53–62, 2002.
 - [37] York Sure. *On-to-knowledge Ontology based Knowledge Management Tools and their Application*. *KI*, 16(1):35–37, 2002.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig

12.10.2015

Ort

Datum

Unterschrift